Une nouvelle approche pour la conception créative: De l'interprétation du dessin à main levée au prototypage d'interactions non-standard

Stéphane Huot

UNIVERSITÉ DE NANTES

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATÉRIAUX

Année : 2005 N^0 B.U. :

Thèse de Doctorat de l'Université de Nantes

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

Stéphane HUOT

le 12 Juillet 2005

à l'École Nationale Supérieure
des Techniques Industrielles et des Mines de Nantes

Une nouvelle approche pour la conception créative : De l'interprétation du dessin à main levée au prototypage d'interactions non-standard

Composition du jury :

Président	:	**Henri BRIAND**	Professeur, École Polytechnique de l'Université de NANTES
Rapporteurs	:	**Pierre LECLERCQ**	Professeur, Université de LIÈGE
	:	**Philippe PALANQUE**	Professeur, Université Paul SABATIER, TOULOUSE 3
Examinateurs	:	**Claudie FAURE**	Chargée de recherche CNRS, ENST PARIS
	:	**Gérard HÉGRON**	Professeur, École d'architecture de NANTES
	:	**Cédric DUMAS**	Maître assistant, École des Mines de NANTES
Invité	:	**Jean-Daniel FEKETE**	Chargé de recherche INRIA, Université PARIS Sud

Directeur de thèse	:	**Gérard HÉGRON**
Laboratoire	:	CERMA, CNRS UMR 1563, Université de NANTES
Co-encadrant	:	**Cédric DUMAS**
Laboratoire	:	École des Mines de NANTES

N^o ED 0366-

Remerciements

Comme le veut la tradition, je vais tenter de satisfaire au difficile exercice de la page des remerciements, peut-être la tâche la plus ardue de ces années de thèse. Non qu'exprimer ma gratitude envers les personnes en qui j'ai trouvé un soutien soit contre ma nature, bien au contraire. La difficulté tient plutôt dans le fait de n'oublier personne. C'est pourquoi, je remercie par avance ceux dont le nom n'apparaît pas dans cette page et qui m'ont aidé d'une manière ou d'une autre. Ils se reconnaîtront. Pour les autres, non merci. Ils se reconnaîtront aussi...

La première personne que je tiens à remercier est Cédric DUMAS, mon *chef*, qui a su me laisser la liberté nécessaire à l'accomplissement de mes travaux, tout en y gardant un œil critique et avisé. Nos continuelles oppositions, contradictions et confrontations ont sûrement été la clé de notre travail commun. Plus qu'un encadrant ou un collègue, je crois avoir trouvé en lui un ami qui m'a aidé aussi bien dans le travail que dans la vie lorsque j'en avais besoin. *Kakebuke* à toi et à ta famille.
Si il est beaucoup de personnes que l'on qualifie à tort de sage ou de savant, ce n'est pas le cas avec Gérard HÉGRON. Il sait, invente, transmet ; il écoute, comprend, tempère. Il a toujours montré de l'intérêt pour mes travaux et répondu à mes sollicitations lorsque le besoin s'en faisait sentir. J'espère que cette thèse sera un remerciement suffisant au soutien et à la confiance sans cesse renouvelée dont il a fait preuve en mon égard.
Je les remercie particulièrement tous les deux d'avoir fait de moi leur *Padawan* lorsque le conseil des *Jedi* émettait des doutes quant à mes aptitudes.

Je remercie Pierre LECLERCQ et Philippe PALANQUE de m'avoir fait l'honneur d'être les rapporteurs de cette thèse. J'éprouve un profond respect pour leur travail et leur parcours, ainsi que pour leurs qualités humaines. Le regard critique, juste et avisé qu'ils ont porté sur mes travaux ne peut que m'encourager à être encore plus perspicace et engagé dans mes recherches.
Merci à Henri BRIAND, Claudie FAURE et Jean-Daniel FEKETE d'avoir accepté de faire partie de mon jury. Leurs remarques et commentaires ont jeté un courant d'air frais qui m'a permis de clarifier certains points de ce manuscrit.
Je remercie tout particulièrement Jean-Daniel pour nos fréquentes discussions sur de nombreux sujets, allant du futur de l'IHM au Rock'n Roll, mais aussi pour son implication dans mes travaux. Sa manière bien à lui de pousser continuellement à la remise en question m'a été d'une aide précieuse.

« Faire une thèse » est une échappée solitaire au milieux de compagnons de fortune (ou d'infortune). On ne se rend pas toujours compte à quel point ils peuvent être importants dans le travail et dans la vie, jusqu'au jour où nos chemins se séparent.
Je ne remercierais pas Pierre DRAGICEVIC, Frédéric JOURDAN, Didier BOUCARD et Mohammad GHONIEM. Je me contenterais de regretter les moments que l'on a passés autour des incomparables *Morteaux/Pommes de terre* de Fred. et Nathalie, les plans *bière/guitare* avec Piotr, les délires *Polovolovoï* de Daïdieure et la voix calme et détendue de Moh. *« May the Spiral be with us, guys »*.

Merci à Christian COLIN pour l'intérêt qu'il a porté à mon travail, ainsi qu'à Pierre MACÉ et Louis-Paul UNTERSTELLER pour ce qu'ils sont, ce qu'ils savent et ce qu'ils donnent.

Je remercie *en vrac* les membres (passé et présent) du Département Informatique de l'EMN, dirigé par Pierre COINTE. Tout particulièrement Narendra « la modestie », Philippe « la raquette », Martin « la souris à un bouton », Remi « le maître du jeu », Hervé « ClassLoader », Andres « la raclette », Yann-Gaël « Design Pattern », Mohammed « le cousin », Abdallah « le prince », Samir « le bon », Gustavo « le smile », Christine « l'indispensable », Sylvie « l'indispensable aussi ». Merci aussi à ceux qui s'y sont arrêté un instant dans leur parcours et dont la rencontre a été plus qu'un simple « bonjour-au revoir » : Stéphane « Allez les verts » CONVERSY, Geoffrey « Mais y sont où les canaris » SUBILEAU, Frédéric « le spéléo. » FÜRST, Jairo « Yé lé toue » ROCHA. Merci aussi à ceux du fond du couloir, Christian « la classe » JANIN, Laurent « pète un plomb » MENU-KERFORN, Kaddour « pas toujours de bonne humeur » FELLAH et tous les membres du SIC, ainsi qu'à Anthony « j'ai l'image et le son » DIAZ.

Merci aux membres du Département Informatique de l'IUT de Nantes pour leur accueil, leur soutien et leur confiance renouvelée lors des derniers mois de ma thèse. Merci à Maurice, Sébastien, Jean, Dalila, Anne, Noëlle, Christine, Pierre, Nassim, Henri, Thierry, etc.
Merci aussi aux personnels du laboratoire CERMA de l'École d'Architecture de NANTES, tout particulièrement à Daniel SIRET (Oui, je sais Daniel, le terme d'architecture n'est pas adapté aux systèmes informatiques...).

Cela va de soi, je remercie évidemment ma famille pour son irremplaçable et inconditionnel soutien. Ils ont été présents pour écarter les doutes, soigner les blessures et partager les joies. Cette thèse est un peu la leur, aussi. Merci Raymonde, Jean-Jacques, Véronique, Antoine, David, Marie, Lucas.
Merci aussi à Bertrand d'être toujours à côté de moi.

Un soir de décembre 1996... Quel chemin parcouru depuis ce temps là où on a fait germer LES SUPER HÉROS ! Merci à Benjamin, Bruno, Eloi, Fabrice, Florian, Germain, Jérôme et Lionel pour notre Skak'n Roll (et tout ce que l'on met dedans) sans lequel j'aurais eu du mal à garder l'équilibre. Merci aussi à toute la tribu qui nous suit dans cette aventure qui, je l'espère, n'est pas prête de s'arrêter.

Loin, loin, loin, loin, loin...

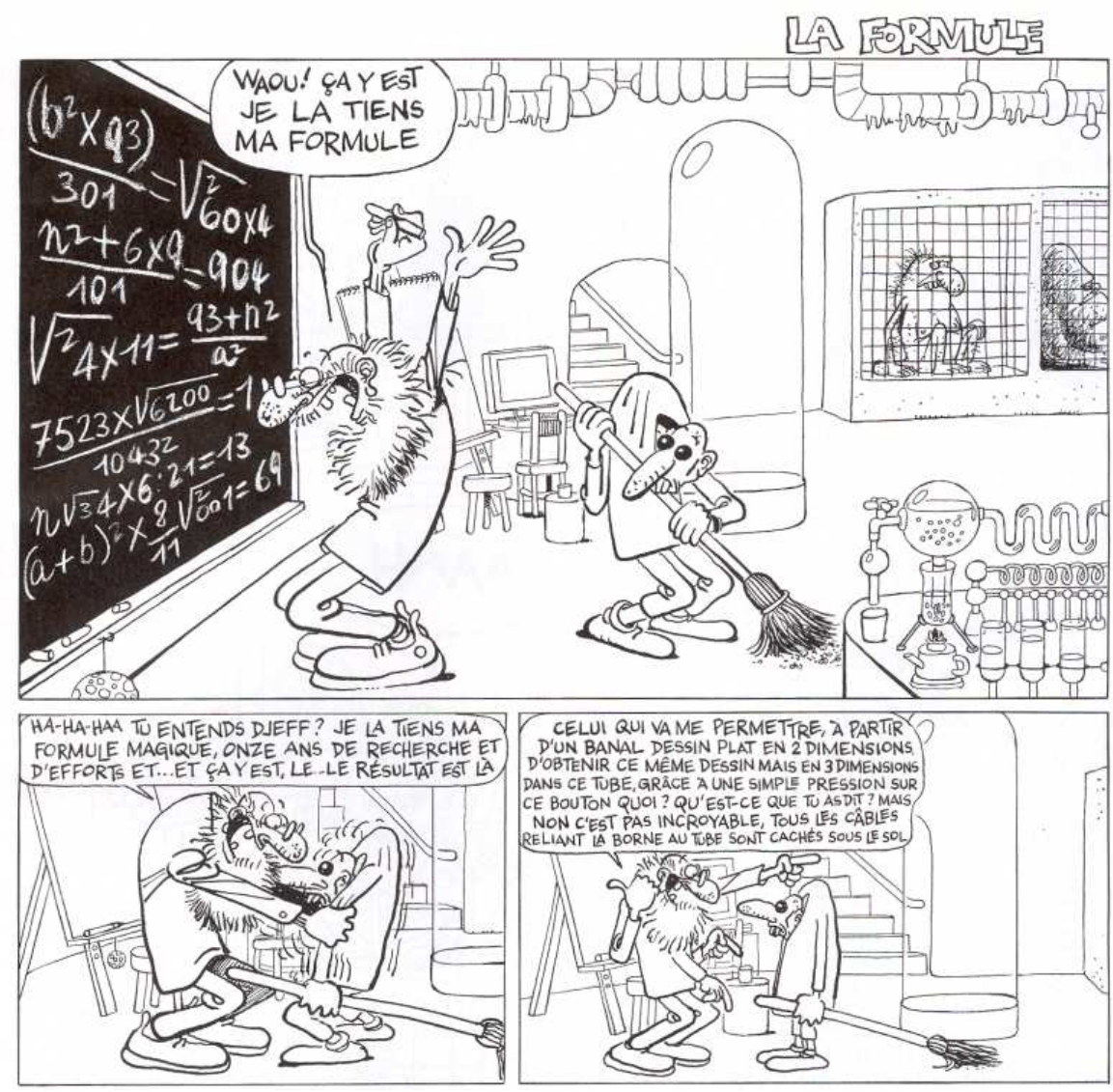
Le plus fort de mes remerciements est pour Vanessa. Merci de m'avoir tenu la main jusqu'aux dernières lignes de ce mémoire. En plus de me supporter dans des conditions « normales », ce qui n'est déjà pas une mince affaire je le conçois, elle a du subir mon humeur de *barbu en fin de thèse* et se mettre en retrait. Et pourtant, quelle présence indispensable. Merci d'être là tous les jours.

Enfin, merci aux Beatles sans qui je n'aurais jamais tenu le coup tout au long des jours et des nuits de rédaction de ce mémoire...
« ... I've got no kick against modern jazz
Unless they try to play it too darn fast
And lose the beauty of the melody
Until they sound just like a symphony

That's why I go for that rock and roll music
Any old way you choose it... »

À Georges Béchade, mon grand-père.

Table des matières



L'auteur de bandes dessinées EDIKA *n'aurait-il pas un jour rencontré Pierre* MACÉ *? Image tirée de Edika. La Formule, Fluide Glacial numéro 334, mai 2004, pp. 57-66. © Éditions Audie 2004.*

Introduction

En 1963, Ivan SUTHERLAND *présente* SKETCHPAD *[Sutherland, 1963], premier système de dessin interactif sur ordinateur.*

Historique et contexte

« Il n'y a pas de raison pour quiconque d'avoir un ordinateur à la maison. » déclarait Ken OLSON, président de *Digital Equipement Corp.* en 1977. Évidemment, cette prise de position peut porter à sourire à une époque où les ordinateurs, et l'informatique d'une manière générale, ont pris une place importante dans beaucoup des activités professionnelles, culturelles, artistiques et de loisir de notre vie courante. C'est à cette même époque, dans les années 70, que les chercheurs de *Xerox* travaillaient sur le système informatique qui allait inspirer le paradigme des interfaces actuelles et contribuer largement à la démocratisation de l'informatique dans tous les milieux : **Xerox Star**. Et c'est aussi en 1970 qu'un laboratoire du *CNRS* présente **EUCLID**, l'un des premiers systèmes de Conception Assistée par Ordinateur tridimensionnel, issu du *« besoin de modéliser des formes tridimensionnelles complexes, pour y greffer des calculs d'aérodynamique ou d'hydrodynamique »* (selon l'entreprise DATAVISION qui développe ce logiciel, cité dans [Lavoisy, 2000]).

Ces événements ne présentent a priori aucun lien, si l'on écarte leur simultanéité. Pourtant, il s'avère que ce sont des points clés dans la genèse de nos travaux. En effet, Pierre MACÉ et Louis-Paul UNTERSTELLER, forts de leurs expériences de mathématicien et d'architecte, ont émis l'idée commune de rendre la construction de modèles 3D plus proche de ses utilisateurs potentiels, plus naturelle par l'utilisation du dessin en perspective. Et ce alors même qu'ils collaboraient à la conception

d'*EUCLID* dont la ligne directrice était au contraire de travailler *« directement sur l'objet à concevoir... et non sur sa représentation symbolique, le dessin »* (toujours selon les mêmes sources). Cette préoccupation pressentait déjà la démocratisation de l'outil informatique et son utilisation potentielle dans des contextes professionnels ou personnels, impliquant des modes d'interaction plus adaptés aux utilisateurs et à leurs aptitudes. Précisons tout de même qu'à l'époque, les cartes perforées étaient une modalité d'entrée courante et les tables traçantes un périphérique de sortie usuel pour les activités graphiques. Nous noircissons peut être le tableau, mais à cette époque, l'interaction pour le dessin sur ordinateur n'allait pas beaucoup plus loin que le prototype (révolutionnaire au demeurant) SKETCHPAD datant de 1963 [Sutherland, 1963] (voir la figure en début de ce chapitre).

Ce n'est qu'une vingtaine d'années plus tard que Pierre MACÉ et Gérard HÉGRON initiaient le projet **GINA** (**G**éométrie **I**nteractive et **Na**turelle) à l'École des Mines de Nantes. Évidemment, de nombreux systèmes de modélisation 3D et de *CAO*[(1)] avaient été proposés depuis les premières idées : des *enfants* d'EUCLID, orientés sur la **manipulation** *plus ou moins* **directe** d'un objet 3D, des *cousins* de SKETCHPAD, fondés sur un paradigme de **dessin vectoriel**. Pourtant, bien que de plus en plus performants et fonctionnels, aucun de ces systèmes ne s'écartait du paradigme d'interaction stéréotypé introduit par *Xerox Star* (une souris, des boutons, des menus, etc.). Un paradigme qui a permis la standardisation des interfaces graphiques et donc favorisé leur développement, mais au détriment de leur utilisabilité et de leur adaptabilité à des tâches bien spécifiques. Mais plus important encore, aucun ne considérait la démarche spécifique des acteurs de la conception : manipuler et dessiner des concepts, et non assembler des entités géométriques.

Les deux objectifs de GINA

Les hypothèses initiales de GINA étaient donc bien encore d'actualité et d'autres travaux de recherche proposaient des approches similaires : dessiner pour construire des scènes en 3D. Dans ce contexte, l'originalité du projet tenait en grande partie dans l'utilisation de la perspective (contrairement aux autres approches qui imposaient des restrictions de point de vue) et dans des outils mathématiques novateurs pour l'élévation 3D de l'objet esquissé (une approche algébrique).

Un objectif mathématique
Le premier objectif fut d'étudier les formalismes mathématiques adaptés et de concevoir un noyau fonctionnel permettant d'obtenir un modèle numérique 3D à partir de sa projection en 2D et de l'énoncé de contraintes géométriques. Cela a donné lieu à deux thèses successives, qui ont abouti à un solveur de contraintes mathématiques original et performant : à partir d'une projection 2D et des contraintes géométriques associées à ses éléments (parallélismes de droites, coplanarités, etc.), une scène 3D peut être construite [Kuzo, 1999; Sosnov, 2003].

Un objectif interactif
La seconde étape, qui a initiée nos travaux, fut alors d'étudier les modalités et techniques d'interaction à mettre en œuvre pour une utilisation **interactive** de ce noyau mathématique, l'idée directrice étant d'utiliser le **dessin** pour la saisie *naturelle* de la projection perspective 2D. Mais plus que la réalisation d'une interface de saisie de dessins et de contraintes, notre ligne directrice fut surtout d'étudier l'intégration de tels outils dans le contexte **créatif** des premières phases de la conception appelées **conception créative**.

(1) Conception Assistée par Ordinateur

De la signification de la lettre *C* dans *CAO*

Les systèmes actuels de **CAO** tendent à assister les concepteurs en tout point d'un projet, de la création à la fabrication du produit. On trouve ainsi des systèmes informatiques complets, permettant par exemple toutes sortes de simulations, générant des plans de construction, contrôlant des machines outils et cela dans divers domaines : la mécanique, le design, l'architecture, etc. Leurs apports sont indéniables, que ce soit pour réduire les coûts (il n'est plus nécessaire de construire des prototypes de simulation ou des maquettes de présentation), améliorer la communication entre les différents acteurs ou unifier l'environnement de conception (échange et archivage des différentes étapes du projet, etc.).

Pourtant, ces systèmes n'ont toujours pas trouvé leur place dans les premières phases de la conception, celles où ne sont encore manipulées que des prémices de solutions au problème posé. Une partie de notre travail consiste justement à identifier *quelle pourrait être cette place*, mais surtout *comment donner cette place à des outils informatiques*. Car si l'on évoque l'adéquation des logiciels actuels à une tâche de conception créative, le constat est plutôt amer. Ils imposent de manipuler des données précises, en totale correspondance avec leurs structures de données internes : spécifier des positions, des transformations et des relations mathématiques de manière plus ou moins interactive. Mais est-ce en accord avec le mode de pensée du concepteur ? Nous montrerons que ce besoin de précision, cette démarche *constructive*, implique de fixer trop rapidement les choses, de faire des choix dès le début de la conception et limite de fait l'expression de la créativité, l'itération progressive dans un espace de solutions issues d'idées, d'associations, de recoupements, d'hésitations et de découvertes.

Alors, même si l'on peut associer le terme de **Construction** à celui de **Conception** pour la lettre *C* de *CAO*, il est certain que nous ne pouvons pas encore lui substituer celui de **Conception Créative**.

Vers des *Outils* informatiques pour la création

Il est pourtant reconnu par les concepteurs eux-mêmes que l'informatique pourrait être un atout dans le processus créatif, offrant à la fois des moyens d'analyse et de description pour aider dans la démarche créative, des outils de recoupement et de visualisation pour aider à la résolution de problèmes et à l'émergence de nouvelles solutions. C'est dans ce sens que les domaines de la psychologie ergonomique et de l'informatique ont associés leurs efforts : mieux comprendre les processus cognitifs engagés dans l'activité de **conception créative** pour proposer des systèmes informatiques plus enclins à les supporter. Deux tendances majeures se sont alors affirmées :

1. Axer les outils sur les **savoirs** du domaine, et étudier alors leur modélisation et leur manipulation par un système informatique. Même si ces méthodes commencent à être efficaces pour manipuler des concepts au niveau du système, elles se heurtent encore à des problèmes pour leur présentation à l'utilisateur dans un contexte créatif : comment ne pas orienter l'activité de manière stéréotypée, ni la perturber, tout en essayant d'y apporter une valeur ajoutée ? Des réponses existent à ces problèmes mais elles restent anecdotiques et peu intégrées à des environnements de conception à grande échelle.
2. Étudier les **savoir-faire** du domaine de la tâche pour en tirer parti dans les outils logiciels. C'est une approche misant sur l'interaction entre l'utilisateur et le système, visant à le replacer dans un environnement *créatif* proche de ses habitudes ustensilaires afin de rendre *transparent* l'usage de l'informatique. Ces approches *ergonomiques* rencontrent le plus souvent des problèmes techniques, liés à la difficulté à traiter et à interpréter des actions qui sortent du cadre de la simple commande, mais aussi à la rigidité des systèmes actuels en terme d'interaction.

Bien sur, ces deux approches ne sont pas disjointes et la majorité des travaux actuels tendent à les associer. Il reste toutefois de nombreuses voies à explorer dans chacune d'elles avant de pouvoir proposer des environnements informatiques complets de **support à la création**.

Nos travaux de recherche s'inscrivent dans la démarche *savoir-faire*[(2)] et notre objectif dans cette thèse est d'étudier un nouveau mode d'utilisation des logiciels dans la démarche des concepteurs afin de proposer de véritables **outils** de conception et non de simples assistants. Comme le suggèrent nos remarques précédentes, il est important de se placer dans un cadre précis, un domaine et une tâche identifiés afin de proposer un outil adéquat. C'est pourquoi nous nous intéressons en particulier à la conception de modèles numériques en trois dimensions dans les premières phases d'un projet architectural, un apport attendu des systèmes informatiques mais encore hors d'atteinte avec les logiciels de CAO traditionnels. Toutefois, notre approche ne consiste pas non plus à reporter la créativité au niveau de la machine, mais à définir les principes interactifs qui peuvent permettre à l'utilisateur de tirer parti du système pour s'ouvrir de nouvelles voies créatives.

Nous pensons que la réussite de cette entreprise passe par l'**intégration** de l'informatique dans la démarche du concepteur et que le **dessin**, outil figuratif majeur de la démarche créative, est le mode de communication à mettre en œuvre pour réaliser ce rapprochement. Beaucoup d'autres travaux ont mis en avant cette notion d'*interface de dessin*. Nous prônons par contre une approche plus complète, basée à la fois sur la construction d'un véritable **environnement** interactif de conception (dont l'emploi est le plus transparent et naturel possible) et une démarche la moins contrainte qui soit : un **système de dessin à main levée**.

Nos contributions

Dans notre thèse, nous introduisons un système de dessin pour la conception de modèles 3D : **SVALABARD**. Nous proposons une **étude** originale sur le dessin d'architecture en perspective dont les résultats, associés à des hypothèses de conception issues de l'état de l'art, ont permis de définir des **lignes directrices** pour la conception d'un système de dessin libre, ainsi que de réaliser des filtres bas niveau de **traitement du dessin à main levée**.

Le système que nous avons réalisé à partir de ces études évoque la métaphore d'une **table à dessin virtuelle**, issue d'un paradigme original de **feuilles d'interaction** et d'un environnement **instrumenté**. L'emploi cohérent d'interactions avancées (interaction bimanuelle, reconnaissance de gestes, périphériques d'entrée non standard) au service de la démarche du concepteur et dans un environnement adaptable vise selon nous à une meilleure intégration de l'outil dans le processus créatif. Ces travaux permettent tout d'abord d'envisager l'étude des apports de l'informatique dans les premières phases de la conception d'un projet architectural. Mais plus que cela, nous montrerons aussi que l'architecture logicielle utilisée peut permettre l'adaptation et l'ouverture de notre démarche à d'autres domaines de la conception créative.

Des environnements rigides pour la conception des interfaces et des interactions

Car en effet, ces travaux nous ont aussi conduit à apprécier les environnements actuels pour le développement d'applications interactives. Réaliser une application comme celle que nous avons proposée soulève de nombreux problèmes techniques, liés principalement à la rigidité des architectures

[(2)] Mais nous proposerons tout de même des pistes pour leur association avec des méthodes axées sur le *savoir*.

logicielles actuelles : en terme de gestion des dispositifs d'entrée, de richesse des sorties, de description des interactions et d'intégration de différents paradigmes dans un même modèle d'interaction.

C'est pourquoi, dans notre thèse, nous proposons aussi un **nouveau modèle d'architecture logicielle** pour le développement d'applications interactives : les ***graphes combinés***. En misant sur une description séparant clairement l'interaction et les graphismes dans des formalismes hautement modulaires et en fournissant les mécanismes nécessaires à leur combinaison dynamique, ce modèle permet d'atteindre un niveau de flexibilité et de généricité indispensable à l'intégration de nombreux paradigmes d'interaction dans un environnement unifié. Nous présentons sa réalisation dans la boîte à outils **MAGGLITE**, qui a non seulement permis la réalisation de SVALABARD, mais qui ouvre aussi de nombreuses perspectives pour le **prototypage** et la démocratisation **d'interactions non-standard**.

Organisation de ce mémoire

PREMIÈRE PARTIE - SVALABARD, une table à dessin virtuelle pour le dessin libre. Application à la modélisation 3D
Dans la première partie de ce mémoire, nous cernons le problème global de l'utilisation d'outils informatiques comme support aux activités créatives et nous proposons une solution envisageable dans le cadre précis de la modélisation 3D pour l'architecture.

Dans le **chapitre 1 - Conception, créativité et modélisation 3D**, nous identifions la place des représentations figuratives (le dessin) dans les démarches *créatives*, en particulier dans la conception architecturale. Nous y constatons la place actuelle de l'informatique et nous cernons les apports envisageables à ce niveau.

Dans le **chapitre 2 - Créativité, modélisation 3D et IHM**, nous confrontons les observations du premier chapitre à l'usage des logiciels de CAO actuels, soulevant en particulier les points de leur inadéquation avec les premières phases du projet architectural. Nous dressons alors un état de l'art des solutions déjà proposées.

Dans le **chapitre 3 - Du trait à l'interprétation du dessin**, nous définissons, à partir d'une *étude du trait* et des observations des chapitres précédents, des lignes directrices pour la conception d'un véritable *outil* informatique de modélisation 3D supportant la tâche créative de l'architecte.

Dans le **chapitre 4 - Une table à dessin virtuelle : SVALABARD**, nous introduisons le système SVALABARD. En évoquant une métaphore de *table à dessin virtuelle*, cet outil rompt avec les conventions des IHM de modélisation 3D en s'appuyant sur les méthodes de l'utilisateur et en intégrant des techniques d'interaction avancées.

Dans le **chapitre 5 - Discussion**, nous discutons les apports de nos travaux dans le domaine des interfaces supportant la créativité. En particulier, nous anticipons l'impact que peut avoir un tel système sur ses éventuels utilisateurs et leur démarche créative, avant d'aborder quelques perspectives.

SECONDE PARTIE - MAGGLITE, une boîte à outil post-WIMP avancée
Dans la seconde partie de ce mémoire, nous proposons un nouveau modèle d'architecture logicielle pour la conception d'applications interactives, issu de nos besoins pour la réalisation de SVALABARD.

Dans le **chapitre 6 - Une évolution nécessaire**, nous cernons les problèmes techniques que soulève la réalisation de SVALABARD. Nous identifions les limites des outils actuels de développement d'interfaces et de gestion des interactions par rapport à nos besoins.

Dans le **chapitre 7 - Les graphes combinés**, nous proposons un nouveau modèle d'architecture logicielle pour la conception d'applications interactives, flexible et modulaire tant au niveau de l'architecture graphique que de la description de l'interaction.

Dans le **chapitre 8 - La boîte à outils MAGGLITE**, nous décrivons la boîte à outils MAGGLITE, réalisée à partir de notre modèle. Nous présenterons ses principaux composants et outils, avant de décrire et commenter quelques exemples d'utilisation, en particulier pour la réalisation de SVALABARD.

Dans le **chapitre 9 - Positionnement et discussions**, nous positionnons notre approche par rapport aux contributions majeures et discutons ses apports ainsi que ses limites. Pour finir, nous développons quelques perspectives d'évolution et d'utilisation de MAGGLITE.

Dans l'**annexe A**, nous détaillons les méthodes et algorithmes de traitement en temps réél du dessin que nous avons proposés. L'**annexe B**, apporte des compléments d'information sur l'étude présentée dans le chapitre 3. Dans l'**annexe C**, nous présentons ICON+, l'extension d'ICON que nous avons réalisée. Enfin, dans l'**annexe D**, nous détaillons les mécanismes de MAGGLITE et leur application pour l'extension et l'utilisation de la boîte à outils.

Première partie

SVALABARD, une table à dessin virtuelle pour le dessin libre. Application à la modélisation 3D.

Chapitre 1

Conception, créativité et modélisation 3D

« Rien ne se construit sans être, ouvertement ou non, dessiné »

F. HAMMOUTÈNE.

Sommaire

1.1 Introduction

L'objectif de ce chapitre est de cerner dans quelles proportions des systèmes informatiques peuvent assister les activités de *conception architecturale*, en particulier dans leurs premières phases. Pour cela, nous nous appuyons sur différents travaux des domaines de la psychologie cognitive, de l'informatique et de l'architecture. Cette synthèse offre une base de réflexion sur la place que peuvent tenir des systèmes informatiques dans de telles activités créatives.

Nous partirons d'un modèle général des aspects cognitifs de la conception, qui assimile cette activité à la résolution de problèmes mal définis, la caractérisant alors de *conception créative*. Cette notion de créativité sera développée au travers des mécanismes qu'elle met en jeu : *l'exploration*, *la génération de solutions* et *l'évaluation*. Mais surtout, nous verrons l'aspect essentiel que tient le dessin dans cette activité : un support figuratif, un outil de la pensée.

En quoi cette caractérisation des mécanismes cognitifs peut-elle alors influencer et permettre une meilleure adaptation des outils informatiques à la créativité ? Mais aussi, en quoi les outils informatiques peuvent aider et apporter à la créativité ? Ces deux questions sont évidemment intimement liées. Nous y répondrons d'abord dans le contexte général de la conception créative. Ensuite, nous spécialiserons notre étude au domaine de la conception architecturale.

Cette spécialisation fera intervenir une définition plus contextuelle de la conception, liée au spécificités du domaine et en particulier dans son rapport au dessin. Finalement, nous dresserons un état des lieux sur l'emploi de l'informatique en architecture, avant de cerner quels en sont les apports envisageables.

1.2 La conception

D'après le dictionnaire de l'Académie française :

> **CONCEPTION** N. F. XIIe SIÈCLE, AU SENS PHYSIOLOGIQUE. EMPRUNTÉ DU LATIN *conceptio*, « ACTION DE CONTENIR », PUIS *« conception »*, ET, EN LATIN CHRÉTIEN, « IDÉE, PENSÉE ».
>
> **I**. Action de concevoir un être vivant(...)
>
> **II**. FIG. Activité de l'esprit en vue de la compréhension ou de l'élaboration de quelque chose. **1**. CLASS. Faculté de concevoir, de former des idées générales. Puissance, faiblesse de conception. *Il manifeste une grande clarté de conception. Il a la conception facile, vive, lente.* **2**. Action de former le concept d'un objet et, par ext., d'appréhender un objet par la pensée ; action de former dans son esprit, d'imaginer, d'inventer. *Cette notion n'est pas susceptible d'une conception claire et distincte.* (...) *La conception d'un projet. Un projet au stade de la conception. Suivre un programme d'équipement de la conception à la réalisation. Les défauts de conception d'une machine. Un appareil de conception ancienne, récente.* SPÉCIALT. Idée qui guide la création d'une œuvre et assure son unité. *La conception et la facture d'un tableau. Une mise en scène, un décor de théâtre d'une conception originale. Un roman d'une conception audacieuse.* **3**. Ce que l'esprit crée, produit. *Une des plus belles, des plus prodigieuses conceptions de l'esprit humain.* SPÉCIALT. Doctrine, théorie. *La conception platonicienne de l'État.* (...) *Des conceptions juridiques, architecturales. Les partisans, les tenants, les adversaires d'une conception*. *Par ext*. Manière de considérer, d'envisager(...) **4**. INFORM. Conception assistée par or-

> dinateur ou C.A.O., ensemble des techniques informatiques qui permettent l'élaboration d'un produit nouveau. La conception assistée par ordinateur a de nombreuses applications dans l'industrie automobile et l'aéronautique.

Si nous en excluons le sens propre, cette définition du terme *conception* dénote des aspects *imaginatifs* et *créatifs* de l'acte de conception. Dans le cadre qui nous intéresse, la conception est l'*« action de former le concept d'un objet (..) de former dans son esprit, d'imaginer, d'inventer »*. Cette vision globale de la conception s'applique à tous les domaines créatifs, qu'ils soient artistiques (arts plastiques, architecture, littérature, musique, cinéma, mode, ...), industriels (conception industrielle, ou *design*, architecture, ...) ou scientifiques (recherche, pédagogie, ...). D'ailleurs, en voulant recenser les domaines où intervient la notion de conception, on constate qu'il est difficile de « cataloguer » ces activités dans des domaines précis tant la notion de créativité est transversale. L'architecture, par exemple, est une discipline créative, science de la construction intégrant à la fois des connaissances scientifiques et techniques, mais aussi des notions esthétiques, artistiques et sociales.

Fort de cette définition et des aspects qu'elle soulève, nous proposons d'aller un peu plus avant dans l'étude du *processus de conception* : quels chemins et méthodes, d'un point de vue cognitif, vont être mis en œuvre pour concevoir et donc exprimer la créativité ?

Le domaine de l'industrie et de l'étude des procédés industriels a fortement décomposé les phases nécessaires à l'aboutissement d'un produit (dans des domaines variés tels que la manufacture, l'architecture, la mécanique, etc.). Ainsi, d'une manière très simplifiée, le développement d'un produit se décompose en trois étapes : la conception dite *industrielle*, le *développement* et la *fabrication*. Dans cette organisation, la notion de créativité est « cloisonnée » à la première de ces étapes, elle est l'exclusivité du *concepteur*.

Pourtant, le domaine de la psychologie ergonomique a remis en question cette vision en étudiant et analysant les processus cognitifs mis en œuvre lors du déroulement d'un projet. Cette remise en question montre que la conception n'est pas une simple phase en amont d'un processus plus global de production, mais représente justement les activités tout au long de ce processus, « c'est désigner un ensemble de caractéristiques de certaines situations professionnelles, de la tâche dont on sait qu'elles vont engendrer certaines spécificités du raisonnement et donc de l'activité » [Béguin et Darses, 1998]. Ainsi, de tels travaux, confortés par des études de terrain, ont montré que les acteurs du processus de conception sont nombreux, impliqués en tous points du projet.

Il est alors une ambiguïté sur le terme de conception qui représente pour les ergonomes toute tâche créative lors des différentes étapes de production, et pour les procédés industriels seulement la première phase (le terme anglais « design » lève cette ambiguïté, ne désignant que cette première phase). Nous utiliserons le terme de conception (et donc de concepteur) dans le sens ou l'utilisent les ergonomes, c'est à dire comme représentant une activité créative intervenant à n'importe quelle étape. Dès lors, nous pouvons préciser que dans nos travaux, nous ne intéresserons qu'à la *conception dans les premières phases d'un projet*.

Nous préciserons dans cette section les aspects cognitifs majeurs du processus de conception *dans un cadre individuel*, en considérant principalement la notion de *créativité*.
Nous terminerons par un bilan sur ce que sont et ce que pourraient être les logiciels de support à cette créativité. En effet, le dernier point de la définition vue plus haut définit aussi la conception dans un contexte informatique : *« Conception assistée par ordinateur ou C.A.O., ensemble des techniques informatiques qui permettent l'élaboration d'un produit nouveau. La conception assistée par ordinateur a de nombreuses applications dans l'industrie automobile et l'aéronautique »*. Cette définition

est la seule qui ne cite pas d'aspects *conceptuels* et *créatifs*. C'est pourquoi nous irons au delà de cette vision des systèmes informatiques comme de simples calculateurs en dressant un bref état de l'art des travaux qui tendent à en faire de véritables supports à la créativité.

1.2.1 La conception créative

Depuis les années 80, le domaine de la psychologie ergonomique a produit beaucoup d'analyses sur l'impact qu'ont les caractéristiques externes des tâches de conception sur la construction des représentations mentales que forment les concepteurs dans leurs activités (design, mécanique, architecture, programmation informatique, etc.). L'objectif en est essentiellement, par la compréhension de ces mécanismes, de produire des outils d'aide aux concepteurs. C'est pourquoi de nombreux travaux se sont d'abord axés sur la question de savoir « Comment les concepteurs réalisent leurs activités ? ».

D'un point de vue global, les problèmes de conception sont par essence « mal définis » : la solution est un système qui doit satisfaire un ensemble de contraintes très peu définies au départ et spécifiées tout au long du processus. Les études et analyses de Vinod GOËL [Goël et Pirolli, 1989] ont permis de caractériser les démarches invariantes mises en jeux par des concepteurs lors de la résolution de ces problèmes larges et complexes :

1. Ils montrent une activité intense de structuration et de restructuration du problème.
2. Ils développent plusieurs modèles du système (traduit par des diagrammes, prototypes).
3. Les problèmes peuvent être résolus par un ensemble de solutions plus ou moins acceptables que le concepteur doit alors évaluer. Cela implique la mise en œuvre de *fonctions d'évaluation*.
4. Cette évaluation est cyclique et s'effectue par approximations successives.
5. Les concepteurs ont tendance à préciser progressivement les contours du système, laissant ainsi la place à des alternatives.
6. Ils décomposent le problème en modules perméables ayant des intersections et des liens plus ou moins forts.
7. Leur démarche évolue de buts abstraits aux spécifications concrètes en passant par des abstractions de plus en plus spécifiques.
8. Ils utilisent des systèmes symboliques ou graphiques pour décrire les résultats intermédiaires.

Cette analyse du processus de résolution de problèmes montre bien que la démarche du concepteur consiste à conserver une vision globale, tout en proposant plusieurs solutions possibles au problème afin de le cerner (il est clair que la séparation entre l'analyse du problème et sa solution reste floue : un problème totalement défini est résolu). D'autres études sur des concepteurs et ingénieurs de domaines et savoirs différents [Maccoby, 1991] ont conforté cette vision par le constat que l'approche la plus fréquemment employée est de rechercher une vision globale et détachée avant d'avancer vers des détails plus précis.

Dès lors, la proposition de solutions et leur évaluation, associées à un certain recul par rapport au problème, permettent aux concepteurs d'introduire de nouvelles contraintes. Ces contraintes réduisent l'espace des solutions possibles et permettent l'introduction de nouveaux concepts, changent les buts et induisent encore de nouvelles contraintes. Un tel constat ne confère toutefois pas à ce processus un chemin de résolution prédéterminé, mais souligne surtout son aspect incrémental et itératif. Les méthodologies, expériences et l'existant vont appuyer la démarche, mais il est nécessaire de recombiner et réinventer à chaque fois : « Le rôle du concepteur est de produire l'inattendu » [Grange, 1983].

Un point intéressant et prévalent dans ces études est qu'elles font émerger un aspect fondamental de la conception : la créativité. En effet, cette génération de solutions, que l'on peut en partie associer à l'introduction de nouvelles contraintes au problème, dénote de l'aspect créatif de la conception. Khaldoun ZREIK identifie d'ailleurs la *conception créative* comme un acte de conception sans contraintes à priori [Zreik, 1990]. Selon [De Paoli, 2004], c'est la spécification des contraintes qui n'existaient pas au début du processus qui confère alors un haut niveau de créativité.

Toutefois, on ne peut pas se limiter à une vision trop réductrice de la créativité dans la conception. Il est en effet évident que la frontière est floue entre la résolution de problèmes peu ou mal définis et la conception créative telle qu'elle a été définie précédemment (aucune contraintes de départ). Cette résolution ne se limite pas à la simple maîtrise d'une connaissance et de techniques, mais puise dans l'exploration, l'assemblage d'idées et l'utilisation de connaissances alternatives (extérieures au domaine du problème). Ce sont ces activités créatives qui, selon [Candy et Edmonds, 1999], font appel à une combinaison de rationalité, d'intuition et de créativité. D'une manière plus générale, la notion de *conception créative* relève donc de la capacité à proposer des solutions novatrices à des problèmes mal ou peu définis (en opposition à la résolution de problèmes analytiques précis). Et, plus que la qualification du problème et du nombre de ses contraintes de départ ou de leur nature, les concepteurs eux-mêmes soulignent la part importante de *l'intuition*. Ainsi, Jack Howe cité dans [Cross, 2002], « (je) crois à l'intuition. À mon avis, il s'agit là de la différence entre le concepteur et le simple ingénieur... (Je) fais une distinction entre les ingénieurs et les concepteurs de produits mécaniques... Un concepteur de produits mécaniques est aussi créatif que tout autre concepteur ».

1.2.2 La créativité dans la conception

Les nombreuses études sur le processus créatif ont toutes caractérisé trois activités primordiales : l'*exploration*, la *génération des solutions* et l'*évaluation*.

Les activités de la créativité

Ces activités, qui synthétisent aussi les points vus précédemment, sont essentiellement basées sur la *connaissance*. Connaissance que beaucoup s'accordent à qualifier de prépondérante dans le processus créatif, tant au sens du savoir et des acquis du concepteur que des données qu'il va devoir extraire de l'analyse du problème ou d'autres sources.

L'exploration
L'exploration est l'activité que l'on peut qualifier de *recherche*. Elle met en jeu les savoirs et l'expérience, mais aussi toutes les données externes que le concepteur va pouvoir récolter et assimiler autour de lui. Ainsi, elle permet de cerner le problème, de le situer afin de pouvoir constituer un espace des solutions possibles, de délimiter le problème par des contraintes.

Ernest EDMONDS et Linda CANDY synthétisent de leurs travaux que la connaissance est un élément clef dans l'exploration des idées, du savoir et des alternatives [Edmonds et Candy, 2002]. Les designers, par exemple, vont se servir des usages et de retours d'expériences pour concevoir ou améliorer un objet. Dès lors, l'exploration ne peut qu'être favorisée par un accès à des sources de données, contenant différents types de connaissances qui sont examinées, évaluées et interprétées dans le sens des buts du créateur (savoirs et acquis, travaux connexes, désirs du client, spécifications du problème, dossiers de conception, ...).

Il est évident que l'exploration est un *processus ouvert*, qui ne peut pas être formellement modélisé. Elle doit évidemment se situer dans le domaine particulier lié au problème, se focaliser sur un ensemble complet et accessible de connaissances : savoir où regarder et comment choisir la connaissance en sont les éléments prépondérants. Mais il est toutefois indispensable de ne pas négliger les transversalités.
[Edmonds et Candy, 2002] souligne ainsi des aspects importants de l'exploration :

- *Rupture avec les conventions*. S'affranchir des attentes conventionnelles, qu'elles soient visuelles, structurelles ou conceptuelles, est une caractéristique majeure de l'esprit créatif.
- *Immersion*. La complexité du processus créatif est servie par une immersion totale dans l'activité. Les distractions doivent êtres proscrites.
- *Vue holistique*. Un problème de conception doit être pris en compte par une vue générale. Le concepteur doit ainsi être capable de se *reprojeter* dans n'importe quel point de vue et, en particulier, de trouver ceux d'où la conception émerge.
- *Chemins parallèles*. Garder un nombre d'approches et de points de vue simultanés est un élément nécessaire à l'apparition de nouvelles idées.

La génération des solutions
Le processus créatif se fonde aussi sur l'expérimentation et l'exploration de solutions multiples. Il s'inscrit dans une démarche itérative pour développer un résultat original et nouveau [Csikszentmihalyi, 1999; Schön, 1983]. La génération des solutions possibles implique alors l'aspect critique de la formation du problème (se poser les bonnes questions, comme pour l'exploration).

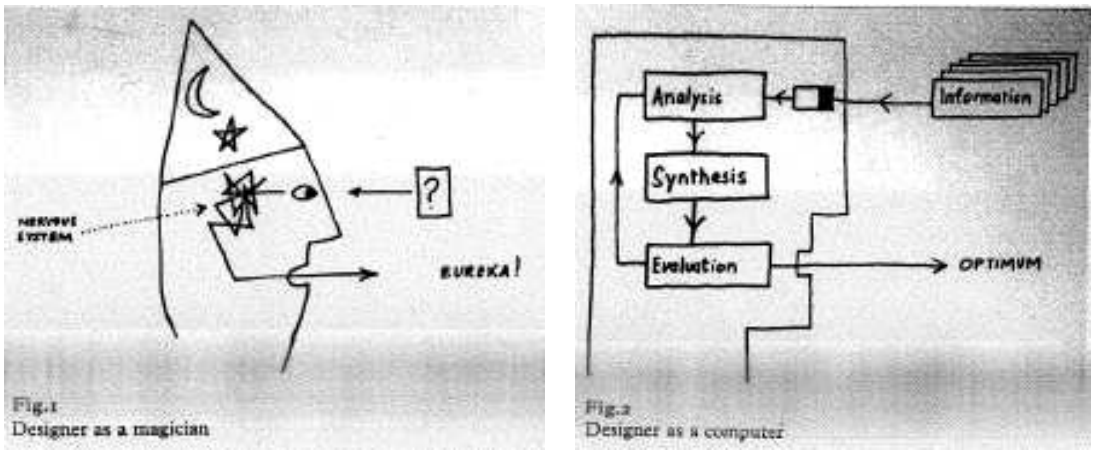
Cette démarche est à la fois un processus de synthèse des connaissances, mais aussi de proposition et d'essai à partir de l'espace initialement délimité par l'exploration et la formulation du problème (d'où son importance). Ces solutions font largement appel à des ensembles de cas analogues, pouvant même surgir de domaines totalement différents à celui dans lequel s'inscrivent les préoccupations : il faut une certaine habilité à créer des associations[(1)].

Dès lors, un point important de la démarche créative est la génération de beaucoup de solutions potentielles. La focalisation sur une seule solution, et donc un chemin unique de résolution qui est souvent le plus évident dans ce cas, offre de fait moins de possibilités pour atteindre l'une des meilleures solutions. Cette habilité à considérer des chemins parallèles dans la conception est un élément important de la génération de solutions, qualifié de *pensée latérale* [de Bono, 1973].

L'évaluation
Ces solutions potentielles doivent alors être évaluées par rapport à l'ensemble de contraintes qui ont été posées au départ, ou introduites a posteriori : il s'agit alors de modifier, reformuler ou écarter des solutions. Mais l'application de ces contraintes n'est pas une fin en soi, une validation finale du processus. Elle doit être considérée comme favorisant l'émergence de nouvelles solutions créatives et de fait, l'évaluation est un point primordial [Boden, 1997]. Elle est souvent constante, depuis la phase d'exploration. Ainsi, le processus est itératif, l'évaluation permanente modifiant l'espace des solutions (réduction ou plus rarement extension) afin de relancer de nouvelles explorations et solutions.

[(1)]Une anecdote intéressante à ce propos concerne Marc DAVIES, animateur de dessins animés pour DISNEY sur le film *Bambi*, qui a étudié et appliqué son savoir sur les expressions du visage des nourrissons pour décrire les expressions du faon. C'est un exemple typique de conception créative où, pour résoudre le problème d'animaux qui apparaissaient trop réalistes et donc pas assez proche du public visé, le concepteur a exploré, utilisé, confronté et combiné ses connaissances dans plusieurs domaines.

Les limites du modèle



(a) Le concepteur « magicien ». (b) Le concepteur « ordinateur humain ».

FIGURE 1.1 – Le concepteur irrationnel (figure a) caractérise la boîte noire, la partie cachée de la conception. Le concepteur rationnel (figure b) caractérise la boîte de verre. Ces images sont tirées d'illustrations de John Christopher JONES.

Il est toutefois important de tempérer ce discours en ne l'enfermant pas dans une vision ou un modèle systématique de la démarche de conception. À la fin des années soixante, John Christopher JONES a d'ailleurs remis en question la formalisation de plus en plus profonde de ces processus. Par deux illustrations, associées au concepts de « boîte noire » et « boîte de verre », il a soulevé la part d'inconnu que recèlent les mécanismes créatifs. La figure 1.1(a) illustre le principe de la boîte noire, le concepteur humain qui va produire un résultat satisfaisant sans pour autant comprendre pourquoi. La figure 1.1(b) illustre le principe de la boîte de verre, le concepteur rationnel qui à partir d'entrées va produire la meilleure sortie possible.

Ces illustrations eurent un impact certain dans le domaine[(2)] : considérer le principe de la boîte noire revenait à abandonner toutes recherches sur le sujet alors que celui de la boîte de verre tendait à simplifier dramatiquement les processus de conception. Cela permis une certaine modération, « gardant la possibilité de qualifier d'artistique ce qui ne pouvait pas encore être qualifié de scientifique » [Chupin, 2002].

Pour l'heure actuelle, ce processus itératif d'exploration, de génération de solutions et surtout d'évaluation constante est admis, même si tous les mécanismes cognitifs qui le régissent et le mettent en œuvre restent non formalisables ou difficilement identifiables (est-ce que l'intuition, dont nous parlions précédemment, est une qualité qui ne s'exprime qu'à un moment précis ?). Par contre, une fois admis, ce processus soulève le problème des capacités cognitives humaines, d'autant plus que toutes ces activités sont entremêlées. Nombre d'études et de travaux dans les sciences cognitives ont montré nos limites sur ce point, que ce soit au niveau de la mémoire ou des capacités de calcul, Do-

[(2)]Quelques années plus tard, certains chercheurs, dont John Christopher JONES, abandonnèrent même toute volonté de rationaliser les processus cognitifs de la conception.

nald A. NORMAN affirmant même que les capacités cognitives humaines sont largement surestimées [Norman, 1993]. Il est alors indispensable pour un concepteur d'utiliser des supports externes, une *extension cognitive* tout au long des activités qui sollicitent lourdement ses capacités. Ces représentations externes des images mentales vont solliciter le système visuel pour soulager la charge cognitive, et c'est là le grand intérêt de l'écriture mais surtout du dessin. Quelle place lui est alors donnée dans le processus créatif ?

Le dessin/croquis dans la conception

Le dessin, et plus précisément le croquis, est considéré comme partie intégrante des activités de conception créative. Il est défini comme l'outil prépondérant de la pensée, « the most important Thinking tool » [Seitamaa-Hakkarainen et Hakkarainen, 2000; Goël et Pirolli, 1989; Goël, 1995]. Il est d'ailleurs d'usage que, dans leur formation, les concepteurs apprennent à maîtriser les techniques de dessin à main levée afin de pouvoir esquisser le plus rapidement et instinctivement possible. Cet apprentissage, et la maîtrise qui en découle, n'est pas dénué de sens. Le dessin est en effet vu par les spécialistes de la psychologie cognitive comme une représentation de l'activité mentale, fixant les idées dans les premières phases de la conception : la concrétisation de concepts. Mais plus que cela, ces représentations visuelles dessinées, qui prennent plusieurs formes suivant les phases de la conception, sont recombinées, modifiées et adaptées.

Eugene FERGUSON propose trois catégories de dessins de conception [Ferguson, 1992] :

1. le *dessin/croquis de la pensée* (« thinking sketch ») supporte l'activité mentale du concepteur.
2. le *dessin parlant* (« talking sketch ») est un vecteur de communication et de discussion pour les acteurs du processus de conception.
3. le *dessin prescriptif* (« prescriptive sketch ») spécifie l'objet conçu pour les observateurs extérieurs au processus de conception.

C'est au premier, le dessin de la pensée, que nous allons essentiellement nous intéresser.

Dans la figure 1.2 page ci-contre, un croquis de conception de l'architecte Harry SEIDLER reflète l'aspect esquissé, et donc imprécis, du dessin de conception, ainsi que la palette des représentations visuelles utilisées. Il n'y a ni format, ni restriction : celui-ci est complètement libre, le concepteur n'est pas tenu de respecter des conventions, des règles géométriques. Tout est bon pour supporter l'activité créative, et une même feuille peut alors contenir des dessins, notes écrites, symboles, calculs, etc...

Mais dans quel sens un tel dessin sert alors le processus de conception ?
Le concepteur ne se contente pas de dessiner, il observe aussi son dessin. Cette *visualisation instantanée de sa pensée* contient un grand nombre d'informations volontairement ou involontairement décrites. C'est justement de l'observation du dessin et de la découverte de ces idées et informations sous-jacentes que va apparaître une réaction, entraînant la naissance de nouvelles idées [Goldsmith, 2002; Suwa *et al.*, 1999]. On rejoint ce que nous avions précédemment exposé sur les liens et itérations entre exploration-évaluation-révision. Le dessin va être la trace de ces opérations, le support listant les contraintes du projet, la mémoire des idées instantanées, même si celles-ci semblent n'avoir aucun rapport avec le problème (exploration et génération de solutions).

Dans ce sens, il s'établit une « conversation visuelle » entre le concepteur et ses croquis, permettant l'évaluation et l'exploration de nouvelles idées [Schön, 1983; Goël, 1995]. La réflexion se fonde sur la relation établie entre la pensée et sa représentation sous forme de croquis. Nigel CROSS qualifie

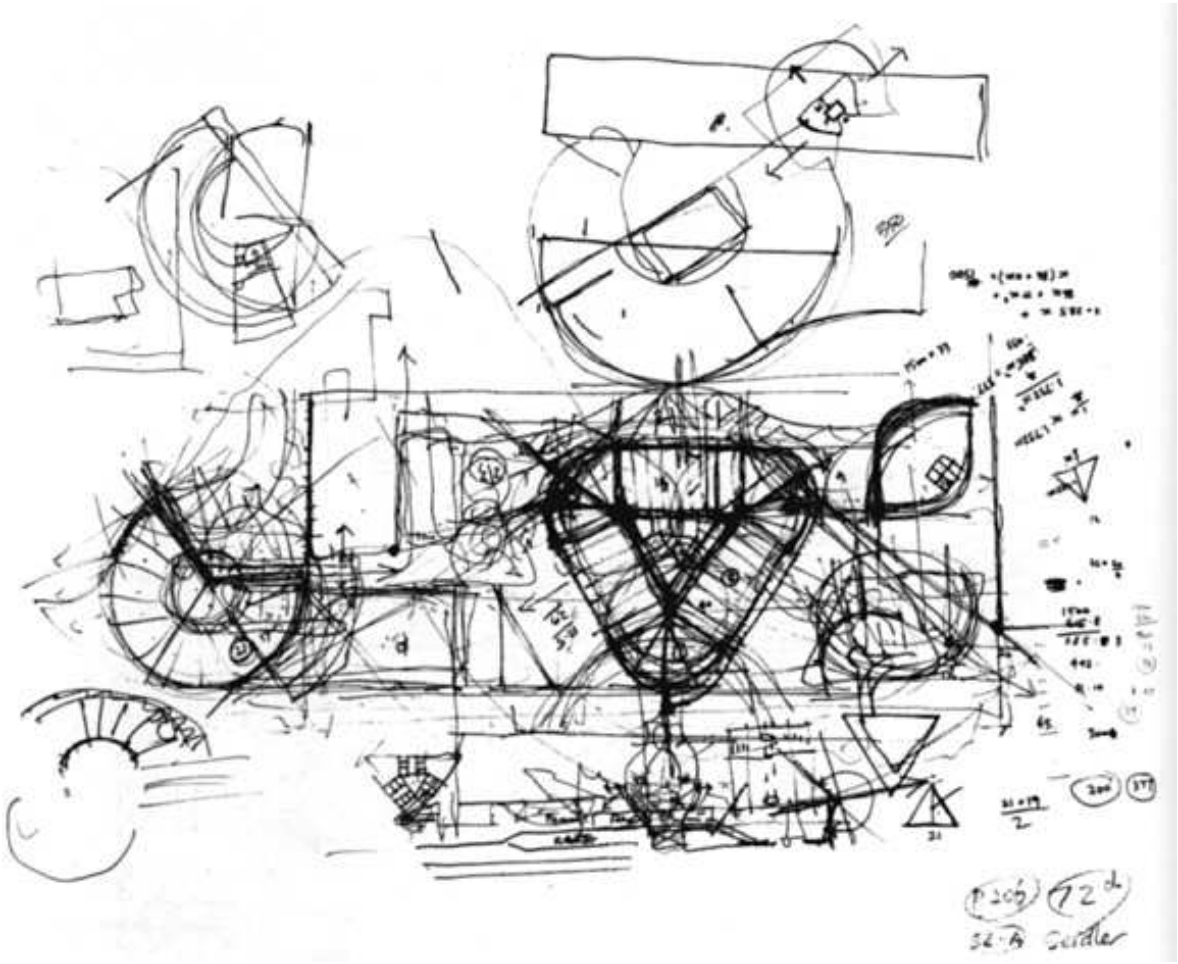


FIGURE 1.2 – Un croquis de conception de l'architecte Harry SEIDLER pour « The Riverside Center », Brisbane, Australie, 1985 (tiré de [Lacy, New York]).

alors la conception de *réflexive* : « le concepteur doit maîtriser un genre de médium – le croquis – permettant aux idées partiellement formées d'être exprimées et réfléchies : d'être prises en considération, développées, rejetées, et reconsidérées. » [Cross, 2002]. Dès lors, le dessin ne peut que favoriser les décisions importantes lors des phases préliminaires de conception.

Le dessin va aussi être le support physique de révisions, raffinements et modifications. Dans le dessin de la figure 1.2 page précédente, des éléments ont étés repassés et redessinés plusieurs fois (parties plus foncées, constituées de traits superposés). Cela reflète les révisions, retouches, affinages et combinaisons des idées et solutions tout au long de la création.
Une technique souvent utilisée par les concepteurs, et spécialement les architectes, est d'améliorer l'efficacité de ce principe de *retouches* en utilisant des calques. Cela facilite la création de séries de dessins. Mais ce ne sont pas seulement des versions ou évolutions différentes, des branches. Leur combinaison va aussi produire des solutions encore plus raffinées [Goldsmith, 2002].

Finalement, plus qu'un support, le dessin devient un instrument de création, par son aspect « flou ». Le concepteur trace des traits qui ne sont pas forcément des descriptions précises de ses idées. Il peut alors les interpréter de différentes manières, mais ils restent *ses* traits et dans ce sens ils sont liés à ses idées, abstraites ou concrètes. Le concepteur expérimenté utilise le dessin pour générer des configurations qui peuvent alors « réveiller » des concepts ou objets qu'il n'avait pas encore portés au premier plan de sa pensée et qui peuvent être des solutions au problème posé. Gabriela GOLDSMITH en déduit d'ailleurs que ces « idées cachées » (« not-initially-accessible ideas ») sont la clef de la compréhension du rôle du dessin dans la production de solutions : il faut chercher, car les idées, mêmes connues, n'émergent pas spontanément lorsque le processus créatif commence.

Les traits, le dessin et le dialogue critique du concepteur avec son dessin vont lui permettre de découvrir des conséquences inattendues. Tout type de visualisation peut aider à l'émergence de ces concepts, mais le dessin l'emporte de par ses avantages d'économie cognitive. Il est rapide et facile à produire, dynamique, flexible et il n'engage que peu d'efforts (nous reconnaissons et interprétons aisément nos propres dessins). Enfin, il focalise l'attention sur le problème, évitant de se disperser dans des recherches inappropriées.

1.2.3 Systèmes informatiques de support à la créativité

La *créativité* est une notion en partie inexpliquée et sa formalisation est impossible, bien que simulable sur certains points (nous en revenons aux illustrations de John Christopher JONES, figure 1.1 page 9). L'icône de l'ordinateur auquel est posé un problème mal défini et qui en tire une solution novatrice est définitivement laissée à la créativité des auteurs de science-fiction et d'anticipation. Nous pensons d'ailleurs que cette vision est l'une des causes principale du refus, voire du dénigrement de l'informatique par beaucoup d'esprits *créatifs* ; d'abord par la frustration, ou même la peur : « la machine ne remplacera jamais l'esprit humain » ; ensuite par l'impression d'inutilité ou d'inadéquation de l'outil : « pourquoi passer du temps à maîtriser et utiliser un ordinateur alors qu'un simple papier et un crayon suffisent ».

Certains exemples prouvent pourtant que les outils informatiques peuvent *supporter* la créativité, voire même faire apparaître de nouvelles formes d'art (voir par exemple le festival SCOPITONE à NANTES : `http://www.scopitone.org`). Seulement, on constate souvent que ces *artistes numériques* ont du s'approprier et même souvent détourner des techniques. Un tel travail nécessite beaucoup d'efforts et de... créativité.

La question qui se pose, dès lors que le domaine de la psychologie cognitive a réussi à lever quelques zones d'ombres du processus créatif, est quels outils informatiques peuvent permettre de fournir un support à la créativité ? Mais avant cela, que peut apporter un outil informatique dans un processus créatif ? Pour répondre à ces questions, nous nous plaçons dans deux contextes différents : les *outils détournés* et les *outils intégrés*.

Les outils détournés

Nous avons déjà rapidement évoqué le fait que l'aspect *nouvelle technique* de l'informatique a introduit de nouvelles formes de création. Il est évident, et l'histoire de l'art le montre, que certains créateurs-artistes ont toujours été à l'affût et à la pointe des nouvelles techniques pour exercer leur créativité (souvent au grand dam de leurs contemporains y voyant un manque évident de sens artistique et de bon goût). Nous ne nous poserons pas la question de savoir si c'est cette créativité qui a engendré une rupture avec les techniques *standards* ou si c'est justement ses techniques qui ne stimulaient pas convenablement la créativité. Nous constaterons seulement que la fin du vingtième siècle a vu apparaître de nouvelles formes de création, directement liées à l'emploi d'outils numériques et informatiques (musiques électroniques, peinture numérique et modélisation 3D, mondes virtuels, etc.). Cet aspect des outils informatiques pour la création n'est pas le propos de notre travail, mais permet toutefois un constat intéressant : beaucoup d'outils informatiques initialement conçus pour s'intégrer dans des processus créatifs dits « traditionnels » ont initié de nouvelles formes de création, sans pour autant remplir toutes les tâches auxquelles ils étaient destinés.
Il reste alors à savoir où réside l'apport créatif à ce niveau : avoir eu l'inspiration d'utiliser ces outils pour ouvrir de nouveaux « buts » (en l'occurrence, de nouvelles formes d'art), ou bien continuer à les utiliser pour atteindre ces buts.

Les outils intégrés

Plus proche des méthodes traditionnelles, et dans l'optique de nos travaux, il est une tendance visant à promouvoir les systèmes informatiques au rang de compagnon privilégié du processus créatif, au même titre que le dessin. Mais plus qu'un simple support, la large palette des techniques informatiques existantes peut apporter des valeurs ajoutées indéniables. Il est toutefois indispensable de considérer leur *intégration* aux étapes du processus créatif, dans le but de le favoriser (et non de s'y substituer) :

1. **Lors de l'exploration**. Il est évident que les systèmes d'information sont devenus des media prépondérants dans l'accès à la connaissance sous toutes ses formes et son partage, offrant des méthodes en constante évolution pour son étude, son analyse, sa fusion (bibliothèques numériques, banques de sons, images et vidéos, bases de connaissances, extraction de données, etc...). L'association de ses techniques peut sans aucun doute favoriser l'analyse de problèmes.
2. **Pour la génération de solutions**. Il n'est pas impensable d'utiliser un ordinateur comme « bloc-notes » lors de l'*apparition* ou de la *proposition* d'idées. Associé aux méthodes d'interprétation (dessin, texte, parole), capacités d'archivage et systèmes récents de manipulation de connaissances, il est probable que ce serait un facteur d'émergence de nouvelles possibilités pour la découverte d'analogies (et la génération automatique de solutions, dans une moindre mesure).
3. **Pour l'évaluation**. Au niveau des évaluations numériques, les capacités des ordinateurs nous dépassent largement. Pour des cas précis, l'utilisation de modèles et de simulations sont des

atouts indéniables. Mais aussi, les techniques de visualisation d'une manière générale peuvent donner une autre vision de ses idées au concepteur, lui permettant alors de les évaluer autrement.

Cette liste d'apports possibles ne se veut évidement pas exhaustive. Il est toutefois un point commun que partagent toutes ses valeurs ajoutées : la nécessité de *les intégrer au processus de conception*. Et cette intégration passe évidemment par les méthodes d'interaction et de visualisation que va proposer le système, afin d'établir un dialogue similaire à celui que crée le concepteur avec ses dessins.

Une approche système

C'est pourtant de tout autres chemins qui ont été historiquement suivis en informatique. La première préoccupation a été de fournir les méthodes de calculs, les modèles analytiques permettant de résoudre des problèmes bien définis. Dès lors, apparaissent des *noyaux fonctionnels* ultra-performants, dans des domaines variés tels que le traitement de texte et la PAO (publication assistée par ordinateur), la CAO et modélisation 3D, la MAO (musique assistée par ordinateur) et tous les X-AO que nous connaissons. Mais, bien que les fonctionnalités et les possibilités de ces logiciels soient très étendues, les interactions et l'approche de construction qu'ils proposent ne prennent pas en compte les aspects du processus créatif que nous avons évoqués. Leurs démarches sont intimement liées aux techniques analytiques de leurs noyaux fonctionnels, nécessitant un maximum de données et de contraintes pour produire un résultat, écartant ainsi beaucoup des aspects incrémentaux et itératifs de la conception. De fait, les interfaces et les interactions se résument à des points d'accès aux fonctions du logiciel, ne laissant que peu de place à la retouche, la comparaison, le façonnage d'un concept.

Plusieurs études viennent appuyer ce discours. L'une d'elles porte sur les processus cognitifs mis en jeux lors de la création de graphismes avec un logiciel de PAO (Adobe Illustrator, en l'occurrence) par des professionnels habitués à cet outil [Bonnardel et Piolat, 2003].
Les auteurs constatent que l'utilisation de ce système, basé sur les mêmes paradigmes d'interaction que ses concurrents, pousse les concepteurs à prendre au plus vite des décisions majeures à propos de caractéristiques locales de l'objet à concevoir, et ce dans le but d'en obtenir au plus vite une représentation graphique (comme lorsqu'ils dessinent). Nous en revenons donc à notre propos précédent, ces outils nécessitent des données précises sur l'objet créé, qui ne sont pas forcément encore définies dans les premières étapes de conception. Il est évident que faire de tels choix dans les phases préliminaires réduit *ipso facto* l'espace des solutions possibles au problème. Dès lors, la créativité en est forcément *limitée* : moins d'ouverture à d'autres solutions, modifications ultérieures limitées, peu de possibilités de transversalité. Cela implique un changement complet de démarche pour l'utilisateur.
Mais cette même étude soulève aussi le problème de l'effort cognitif important que génère le traitement de ces mêmes caractéristiques locales. Les auteurs supposent sur ce point que de telles préoccupations dans les toutes premières phases de la conception induisent des choix conflictuels nécessitant alors l'utilisation de fonctionnalités avancées du logiciel. Seulement, accéder à ces fonctions avancées n'est pas trivial et occupe des « *ressources* » cognitives qui ne sont alors plus disponibles pour l'activité créative.
Il est dommage que cette étude n'ait pas effectué une comparaison de la même tâche sans l'outil informatique afin de conforter ses résultats (comme dans [Whitefield, 1986] où une telle comparaison a permis d'identifier des différences dans la tâche, sans toutefois se focaliser sur les aspects cognitifs engagés). Comment doivent alors êtres conçus les outils informatiques de support à la création ?

Une approche utilisateur

Il est très peu envisageable de remettre en question la structure interne des logiciels pour « modéliser » les processus humains. Comme nous l'avons déjà souligné, il reste une part d'inconnu dans

les aspects cognitifs que nous n'éclaircirons probablement pas. Comment alors les modéliser, ou au mieux les simuler. Nous pensons que la remise en question ne doit pas se poser à ce niveau et qu'il est nécessaire de conserver le fonctionnement analytique des logiciels : laissons l'ordinateur faire ce qu'il peut faire et bien, à savoir résoudre des problèmes analytiques qu'un concepteur ne peut résoudre simplement. Toutefois, ces fonctionnalités doivent évoluer pour s'inscrire dans des démarches moins strictes, plus adaptées à la conception (ainsi, pourquoi le noyau d'un modeleur 3D n'intégrerait pas aussi des outils de recherche d'analogies, ou de suggestions sur des objets incomplets).

Mais nous pensons surtout qu'une meilleure intégration du logiciel passe avant tout par une adaptation de ses interactions à l'utilisateur, une intégration basée sur la tâche à réaliser et sur la méthode (consciente ou non) qu'emploie l'utilisateur : une *conception centrée utilisateur* [Norman et Draper, 1986; Norman, 1998; Mackay *et al.*, 1998] plutôt que système. Là encore, il ne s'agit pas de produire un reflet du processus cognitif qu'engage l'utilisateur dans la tâche de conception [Flach et Bennett, 1996]. Nous retomberions dans les travers qui imposent une seule vision, une seule méthode et un seul chemin possible à l'utilisateur. Il s'agit de s'inscrire dans un principe d'interface *écologique*, ou chaque action réalisable et chaque but possible sont clairement accessibles et lisibles, sans pour autant être imposés. Le libre arbitre est laissé au concepteur, qui conserve la maîtrise de la méthode, des intentions et des décisions en s'appuyant sur des outils visibles et adaptés.

Considérons alors le dessin dans la conception créative. Son utilisation est simple, intuitive et naturelle. Aucun système de CAO actuel ne peut rivaliser en terme d'adéquation à la tâche et d'aisance d'utilisation. Ils ne permettent ni de reproduire, ni d'utiliser une technique aussi limpide que le dessin à main levé, essentiellement à cause de leur approche système, centrée autour des fonctionnalités de l'outil plutôt que son utilisation dans un contexte créatif [Eisentraut et Günther, 1997; Suwa et Tversky, 1997]. Dans ce sens, nous apprécions la qualification de Jean-Pierre CHUPIN qui considère que « la plupart des logiciels de CAO se comportent encore comme des assistants de dessin suréquipés » [Chupin, 2002]. Car, contrairement au dessin qui laisse libre champ au concepteur, l'utilisation de ces outils présuppose de la maturité du projet, ne supportant donc pas la conception créative. Il est alors nécessaire d'adapter aussi la structure même de ces logiciels afin qu'ils s'inscrivent dans un processus flou, moins descriptif et plus conceptuel.

Nous pouvons donc synthétiser ces remarques par le fait que les outils informatiques peuvent supporter la créativité à au moins deux niveaux distincts :

1. en aidant à collecter le savoir et la connaissance, à les partager, les intégrer et voire même à générer les idées : *intégration des savoirs et des savoirs-faire*.
2. en permettant la création d'artefacts créatifs dans un domaine particulier en fournissant les fonctionnalités critiques de manière claire, directe et utile : *une interface écologique, avec des paradigmes d'interaction et des métaphores adaptés*.

Des exemples significatifs
Des travaux comme ceux de Sharon GREENE, [Greene, 2002], ou Michael TERRY, [Terry et Mynatt, 2002], incluent déjà des notions d'exploration indispensables à la créativité. Ces outils ont été conçus pour faciliter l'exploration et l'expérimentation en permettant à l'utilisateur de ne pas faire de choix définitifs (retour arrière, itérations), mais aussi d'avoir des vues des différents choix possibles.

Le principe des *Side-Views* [Terry et Mynatt, 2002; Terry *et al.*, 2004], par exemple, est une technique très adaptée à l'exploration de solutions. Les *Side-Views* sont des vues instantanées de l'application partielle d'une commande à l'objet d'intérêt. Ces vues peuvent être rendues persistantes si l'utilisateur le souhaite. Ainsi, de par leur aspect dynamique, elles peuvent être appliquées, ou même

chaînées et composées entre elles. La figure 1.3 montre l'application simple de ce principe à un outil d'édition d'images.
D'une part, ces vues rendent plus explicites les possibilités et fonctionnalités de l'outil. Mais elles sont surtout une trace de la construction de chemins alternatifs dans l'exploration de solutions.

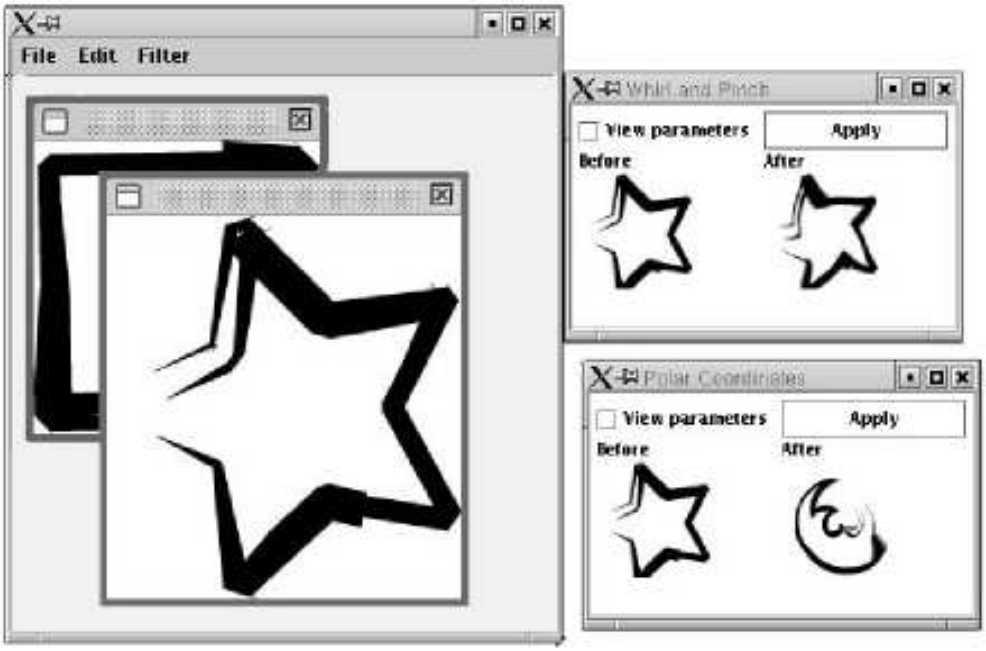


FIGURE 1.3 – Le principe des Side-Views, [Terry et Mynatt, 2002; Terry *et al.*, 2004], appliqué à un éditeur d'images. Ces vues fournissent au concepteur les résultats potentiels de l'application d'une commande.

Cette prévisualisation et manipulation des solutions possibles est une réalisation des « What-if tools », principe plus général proposé par Ben SHNEIDERMAN [Shneiderman, 2000]. Il souligne en effet l'aspect fondamental de pouvoir, dans un processus créatif, essayer, expérimenter et retracer la démarche sans pour autant la figer. Mais plus que cela, il propose aussi un cadre complet pour l'intégration des outils informatiques dans la démarche de conception créative [Shneiderman, 2000; Shneiderman, 2002]. Cette proposition met en relation les quatre activités principales du processus créatif, « Collect – Relate – Create – Donate », avec huit tâches pouvant supporter la créativité, voire l'améliorer. La figure 1.4 page ci-contre représente ces associations.

Il est intéressant de constater que beaucoup d'outils informatiques existent déjà pour accomplir beaucoup de ces tâches (recherche d'informations, visualisation, collaboration, simulations, etc.). L'enjeu majeur est alors leur combinaison et leur intégration homogène et transparente dans un même environnement, un *contexte créatif*.
Toutefois, bien que les activités d'exploration, collaboration et dissémination (collect, relate et donate) soient couvertes par beaucoup d'outils informatiques, il reste à notre avis un long chemin à parcourir pour atteindre un support efficace de l'activité de création (create). Nous en revenons donc toujours au problème de la démarche inadaptée qu'imposent les outils actuels, de par leur réalisation orientée système ne supportant pas ou très peu les aspects *flous* de la démarche créative (ce que Ben SHNEIDERMAN appelle en particulier « Thinking by free associations, Exploring solutions, etc. »).

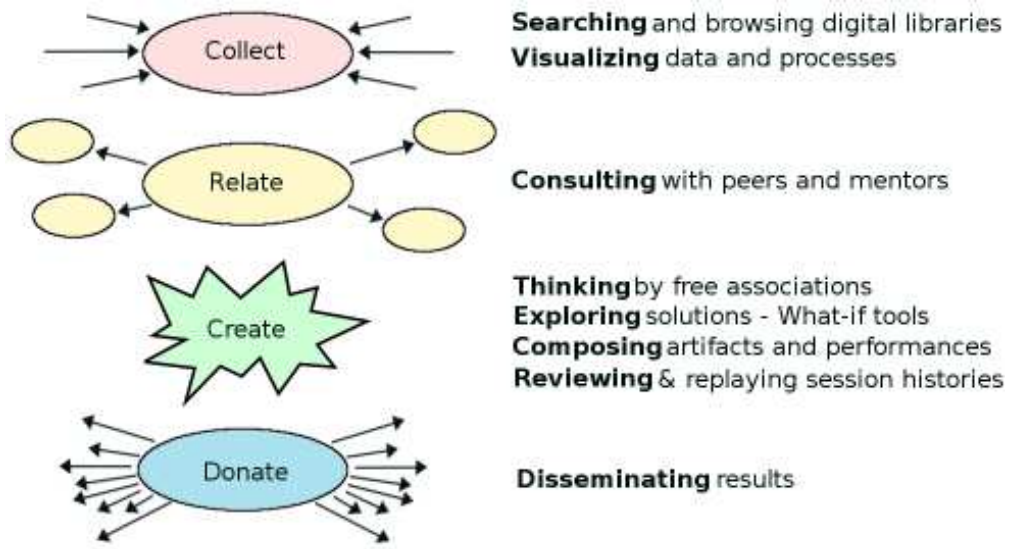


FIGURE 1.4 – Un cadre créatif pour les outils informatiques. Proposé par Ben SHNEIDERMAN [Shneiderman, 2002], ce *framework* créatif met en relation les quatre activités principales du processus créatif avec huit tâches pouvant supporter la créativité, voire l'améliorer.

Cadre de nos travaux

C'est sur cet aspect précis de la conception créative que nous positionnons nos travaux : comment concevoir un outil informatique de support, et même d'amélioration, de la créativité dans les premières phases de la conception ? Nous nous restreindrons pour cela à un domaine précis, celui de la conception architecturale que nous décrivons dans la section suivante. Ensuite, nous montrerons dans le chapitre 3 page 55 les conditions que doit remplir un tel système, à partir des travaux que nous avons exposés dans ce chapitre et de nos propres études. Dans le chapitre 4 page 83, nous proposerons une première réalisation d'un tel outil : SVALABARD.

Mais plus loin que cela, nous constaterons l'inadéquation même des outils informatiques de conception d'Interaction Homme-Machine qui limitent à notre avis le développement des systèmes interactifs : souvent conçus pour un unique paradigme d'interaction et particulièrement figés, ils conduisent au développement d'interfaces stéréotypées, inadaptées à la réalisation de tâches avancées. L'utilisation de tels environnements pour le prototypage, la conception et la combinaison d'interactions avancées nécessite toujours de lourds efforts de programmation.
De plus, et bien qu'ils ne leur facilitent pas forcément la tâche dans le contexte des interfaces non-standard, ils s'adressent principalement à des utilisateurs programmeurs. Pourtant, il est maintenant acquis par la communauté que les informaticiens ne détiennent que rarement tous les savoirs et les connaissances pour proposer des outils comme ceux que nous avons évoqués précédemment [Chatty *et al.*, 2004]. Le travail conjoint d'ergonomes, graphistes et développeurs est devenu un élément clef de la conception des IHM.
Mais les outils dont ils disposent sont-ils adaptés ? Comment, dès lors, proposer des outils créatifs si la créativité des concepteurs de ces outils est elle-même limitée par ceux qu'ils utilisent ou qu'ils détournent ? Nous aborderons ces problèmes dans la seconde partie de ce mémoire et nous proposerons alors une solution basée sur un nouveau modèle d'architecture logicielle (chapitre 7 page 157) réalisé dans une boîte à outils prototype : MAGGLITE.

Mais pour lors, revenons à nos préoccupations sur l'étude des processus de conception créative, et tout spécialement dans le cadre de l'architecture. Dans la section suivante, nous cernerons les particularités du processus de conception architecturale, cadre particulier des notions que nous venons d'aborder. Nous verrons la place qu'y tient le dessin et l'apport que pourraient fournir des outils informatiques, spécialement la modélisation 3D, dans ses premières phases. Associée au chapitre suivant sur les outils actuel de modélisation 3D (outils commerciaux ou expérimentaux), cette section sera, plus qu'un état de l'art, la base des savoirs ayant inspirés nos travaux.

1.3 L'architecture, un exemple de conception créative

La conception architecturale est un exemple concret et complet des notions générales que nous avons évoquées. Devant répondre à des contraintes de départ plus ou moins précises, les problèmes architecturaux sont des problèmes mal définis. Dès lors, l'architecte s'inscrit dans un processus de conception créative, manipulant de larges connaissances issues de domaines transversaux, aussi bien dans des domaines techniques (nombreuses et diverses techniques du bâtiment) qu'artistiques, historiques ou socio-culturels.
En nous appuyant sur les ouvrages référence de Jean-Charles LEBAHAR [Lebahar, 1983] et de Daniel ESTEVEZ [Estevez, 2001], ainsi que sur nos propres expériences et discussions avec des spécialistes du domaine[(3)], nous présentons succinctement dans cette section les spécificités du processus de conception architecturale, en regard des notions plus générales que nous avons exposées précédemment. Nous nous attacherons surtout à y cerner la place du dessin et des systèmes informatiques.

1.3.1 La conception architecturale

Selon Jean-Charles LEBAHAR, [Lebahar, 1983], la conception architecturale fait apparaître trois grandes phases :

1. **Le diagnostic architectural**. C'est dans cette phase de la résolution du problème que l'architecte va le cerner et le définir en respect des contraintes de base. Il va alors prendre en compte les contraintes financières du client, la surface et topologie du terrain, les contraintes écologiques, juridique, les règles de conformité. Il va pour cela visiter les lieux, discuter avec son client mais aussi utiliser des documents liés à ces données : des photographies, des plans de géomètre, etc. Combinant le tout avec ses connaissances et savoirs propres, il est alors en phase *d'exploration* dont le résultat sera une première « base graphique de simulation », mélange de notes et de premiers dessins.
2. **La recherche de l'objet par simulation graphique**. Dès lors, le concepteur va entamer ce que nous avions appelé la *génération des solutions* et leur *évaluation*, dans un processus incrémental et itératif. Et c'est le dessin qui va être le vecteur privilégié de cette démarche. Il va supporter la *simulation*, basée sur les transformations successives que va développer le raisonnement de l'architecte, jusqu'à une définition précise de solutions acceptables au problème. Dans cette situation, comme nous l'avions déjà évoqué en parlant de conversation entre le concepteur et son dessin, le dessin est plus qu'un support. Il représente, comme le souligne Jean-Charles LEBAHAR, « l'objet en création et la pensée qui le crée ».

[(3)]Que Daniel SIRET, Louis-Paul UNTERSTELLER, François GUÉNA et Pierre LECLERCQ voient dans cette section de notre thèse toute notre gratitude pour l'aide qu'ils nous ont apportée.

3. **L'établissement du modèle de construction**. Cette phase est l'établissement des représentations graphiques précises, destinées à rendre claire la solution pour les constructeurs. C'est la « décision définitive concernant l'ensemble du projet » (plans, dessins précis et métrés, avec une échelle spécifiée, etc.).

Nul besoin d'aller plus loin dans les détails de la conception architecturale pour constater que le dessin tient une place prédominante dans ce processus, et ce à toutes ses étapes. Bien entendu, l'architecte a aussi recours à d'autres représentations visuelles. Il utilise divers documents dans les premières phases d'analyse, mais ce ne sont que des apports à sa construction du problème, non des vecteurs de sa résolution. De la même manière, il peut construire des maquettes physiques ou virtuelles de son projet, mais cela présuppose déjà d'une solution aboutie au problème. Elle vont servir à la présentation du projet, à sa communication. Dans une moindre mesure, elles permettront une évaluation supplémentaire de la solution pouvant entraîner des raffinements, mais elles ne constitueront pas un support créatif comme l'est le dessin.

1.3.2 Le dessin d'architecture

FIGURE 1.5 – Dessin sur un coin de table. Photographie : © Thomas Mayer.

Afin de cerner les types de dessin d'architecture et leurs fonctions dans la durée d'un projet architectural, nous reprenons dans cette section les trois fonctions adoptées par Daniel ESTEVEZ : le *dessin spéculatif*, *dessin descriptif* et le *dessin prescriptif* [Estevez, 2001], qui spécialisent celles de Eugene FERGUSON que nous avions déjà évoquées dans le contexte général page 10.

Le dessin descriptif est le vecteur de communication de l'architecte. Il permet de montrer, mais aussi de voir « ce que ça donne ». Le dessin prescriptif est celui qui va permettre la construction, la réalisation de l'édifice. Enfin, le dessin spéculatif est celui que nous avons le plus évoqué jusqu'à présent, le support principal de l'activité créative et conceptuelle. Il ne résulte toutefois pas de cette décomposition une séparation imperméable des fonctions du dessin dans le processus de conception architecturale. Ces fonctions, isolées par Daniel ESTEVEZ pour organiser son discours, s'imbriquent et s'influencent tout au long du processus. Il est d'ailleurs impossible de cloisonner chacune d'elles dans les trois phases décrites précédemment, même et surtout si le « bon sens » suggère une certaine

distribution de ces fonctions du dessin au cours du temps[4].

Toutefois, il s'avère qu'elles offrent une vision claire des différents objectifs du dessin d'architecture. Dans le contexte de nos travaux (offrir de nouveaux outils informatiques supportant la créativité dans la conception), elles vont permettre d'identifier les besoins en la matière mais surtout les apports possibles, en particulier dans les phases préliminaires de conception. Dans ce contexte, les fonctions descriptives et prescriptives du dessin ainsi que les techniques qui leurs sont associées, ne paraissent pas essentielles au premier abord. C'est assez trivial dans le cadre du dessin prescriptif, dont la nature précise et *finalisée* l'écarte à priori des phases préliminaires de conception. Il est par contre plus aisé de voir le dessin descriptif s'inscrire dans les premiers moments de la démarche créative, permettant ainsi une évaluation individuelle ou collective du résultat. Nous verrons pourtant l'importance relative que peuvent prendre ces deux fonctions dès le début de la conception, ainsi que l'importance que pourraient (ou plutôt devraient) leur donner des outils informatiques à ce niveau.

Le dessin descriptif

Le dessin descriptif va permettre à l'architecte de « rendre visible » l'objet qu'il crée. Il va par ce biais chercher à communiquer ses idées et leurs concepts aux autres acteurs du projet, mais tout particulièrement à son client. Dès lors, il doit garder une part d'indéfini et s'attacher à représenter une idée générale, dans l'optique de ne pas figer la vision de son commanditaire. Cela aurait l'effet de brider ses démarches exploratoires ultérieures si la conception n'est pas terminée, ou de provoquer un désaccord si les premières descriptions ont été considérées comme finales.

Ce dessin descriptif utilise principalement la technique de la perspective afin de rapprocher le plus possible l'objet en création de sa future réalisation. Un point intéressant sur le dessin en perspective est que la technique proprement dite, basée sur la géométrie projective, ne concorde pas avec les propriétés du système perceptif humain. C'en est une simple réduction géométrique, écartant certaines notions essentielles de la perception : flou des contours lointains, incertitude des limites du champ visuel, appréciation des fuyantes.
Dès lors, le dessinateur va devoir adapter son dessin afin de le rendre cohérent avec la vision réelle qu'il veut en donner. Cela passe par un choix précis de la disposition des objets et de la distance d'observation, mais surtout par de légères violations des règles géométriques de la perspective. Un exemple en est la correction quasi systématique des déformations latérales et des fuyantes verticales (l'on devine de telles corrections sur la figure 1.6 page ci-contre).

Le dessin prescriptif

Le dessin prescriptif va servir à la construction du bâtiment. Il va établir la communication finale entre les idées de l'architecte et les acteurs de la réalisation physique du projet : les dessins de construction, plans et nomenclatures. Plus qu'une simple transformation *technique* de ces précédents croquis, l'architecte va, à partir des solutions qu'il a établies et qu'il considère comme valables, poser un nouveau regard permettant la détection d'inconsistances [Lebahar, 1983]. Ce passage du conceptuel au constructible implique une grande clarté, supprimant les ambiguïtés. Les représentations figu-

[4]Nous ne nous risquerons pas à une telle classification méthodique et formelle. Il est en effet d'aussi bon sens de supposer qu'un architecte A utilisera beaucoup de dessins descriptifs lors du diagnostic qu'un architecte B des dessins spéculatifs dans la phase d'établissement du modèle de construction.

FIGURE 1.6 – Représentation perspective.

ratives obéissent alors à un code graphique précis et définis, associé à des nomenclatures, détails des matériaux, etc... C'en est fini de « l'espace des solutions » et de la « réversibilité des choix », l'une des solutions est choisie.

Le dessin spéculatif

Nous avons souligné que la conception est une manipulation d'abstractions par leur schématisation, axée sur la globalité, une vue détachée mais unifiée de l'objet. Dès lors, en architecture comme dans un cadre plus général, le dessin lors des phases préliminaires de conception est plus qu'une simple représentation graphique, mêlant conceptualisation et évaluation. L'architecte va utiliser le dessin pour sélectionner et façonner les traits significatifs et pertinents de ce qu'il crée, en fonction des concepts qui guident son analyse.
Un point très important est la propension qu'à le dessin à dialoguer avec son auteur. Outre le fait qu'il favorise l'émergence d'idées spontanées, support incontestable de l'expérimentation, sa disposition spatiale entraîne la perception des relations et inconsistances entre les concepts qu'il représente, point indispensable de la conception architecturale. En effet, la mise en relation successive, rapide et simultanée de contraintes et d'objectifs de sources différentes (structure, fonction, usage, ...) facilite la recherche de cohérence, menant à ce que Jean-Charles LEBAHAR appelle la « réduction d'incertitude » [Lebahar, 1983].

Finalement, le dessin dit *spéculatif* en architecture ne présente pas vraiment de caractéristiques spécifiques par rapport à celles que nous avions déjà évoquées dans le cadre général (voir la sec-

tion 1.2.2 page 10). Il est par contre important d'observer plus avant les aspects techniques propres à ce domaine, du *croquis* au dessin en plans, coupes et élévations (le dessin *géométral*). Nous évoquerons aussi l'utilisation d'un support cher aux architectes : le *calque*.

Le croquis
Au vu de nos lectures, observations et discussions avec des architectes, si l'on devait n'utiliser qu'un seul mot pour caractériser le croquis d'architecture ce serait *liberté*. Le croquis à main levée, bien que souvent dans une vue perspective, n'obéit en effet à aucune règle graphique ou technique, visant avant tout la concision et la rapidité d'exécution. Il illustre une intention plus qu'une réalité ou qu'un but, afin de produire l'effet spéculatif sur son observateur (le dessinateur, dans la plupart des cas). Dès lors, toute considération de précision est écartée (mesures, échelle, ...) pour susciter l'exploration, la projection mentale (voir figure 1.7).

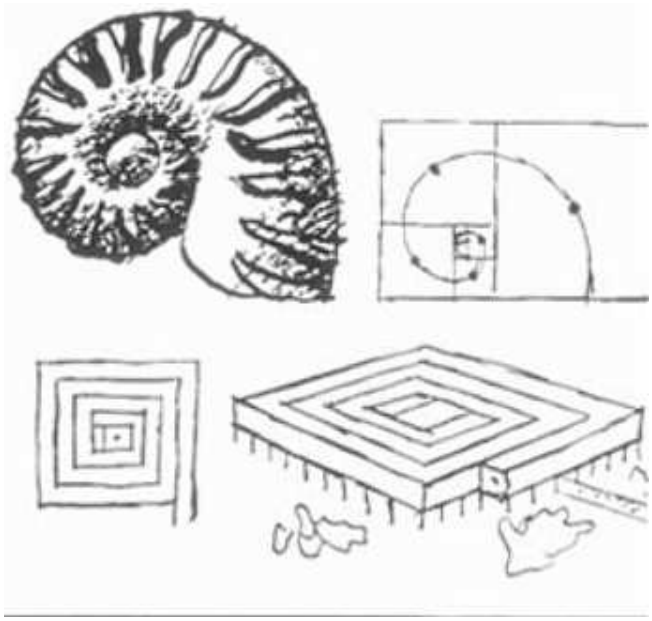


FIGURE 1.7 – Le croquis. Ce croquis de LE CORBUSIER, *le musée à croissance illimitée* – 1939, illustre parfaitement l'aspect conceptuel du croquis. Il illustre un concept, se basant ici sur la métaphore de la spirale, dont la déclinaison va progressivement construire l'objet en faisant appel à toute représentation utile.

Le croquis est donc pour l'architecte « un moyen de simplifier la réalité pour illustrer une intention en allant à l'essentiel » [Estevez, 2001]. Il n'est pas juste de le qualifier de « photographie de la pensée », le terme de photographie impliquant trop de précision et d'exhaustivité. Le croquis suit la pensée de l'architecte, lui permet de sélectionner les traits qu'il va considérer comme importants et de les remettre en cause par leur observation. Nous employons d'ailleurs le mot traits dans deux sens : le trait de crayon, et le trait au sens de caractéristique. Les choix des caractéristiques importantes, les itérations dans l'espace des solutions se font par la sélection et la superposition des traits du dessin. Comme nous l'avons dit dans le cadre plus général du processus de conception, les traits accentués par repassage (avec ou sans calque) sont plus que de nouvelles versions, ce sont des révisions et combinaisons d'idées. Dès lors, l'on peut qualifier le croquis de prolongement de la pensée de l'architecte, sa concision étant liée à la nécessité de suivre la rapidité du raisonnement, son imprécision étant liée aux aspect flous de ce même raisonnement (représenter une intention, et ne fermer aucune porte).

Le croquis d'architecture se décompose en deux catégories :
- *Le croquis d'analyse et d'observation*, essentiellement voué à extraire le concept architectural qui a sous-tendu la conception d'un bâtiment construit ou au moins conçu.
- *Le croquis d'étude*, qui est l'esquisse de conception, celui dont nous discutons ici. Il diffère du croquis d'observation par sa démarche de simulation graphique, contrairement à la vision à posteriori du premier.

Bien que pouvant être distinguées par leur finalité, ces deux catégories de dessins demeurent proches de par leur caractère conceptuel.

Le dessin géométral
Le dessin géométral est un mode de dessin que l'on peut qualifier de plus technique, en deux dimensions, et qui consiste à fragmenter la représentation en coupes, plans et élévations. Le qualificatif de « technique » n'implique pas pour autant, dans les premières phases de conception, une notion de précision dans les mesures et l'échelle. Car, c'est un fait constaté par tous les observateurs de la tâche de conception architecturale, ce type de dessin est celui qui est utilisé majoritairement par les architectes. La perspective, ou autres représentations que l'on pourrait qualifier de réalistes (maquettes), n'interviennent que plus tard dans le projet.

L'utilisation de cette décomposition des dimensions soulève, selon Daniel ESTEVEZ, le « paradoxe du géométral ». En effet, ces dessin fragmentent le projet, ce qui est à priori contraire à la démarche globale de la résolution du problème, et de plus ne semblent pas prendre en compte les aspects volumiques (deux dimensions). Il ressort surtout des études que présente l'auteur que c'est une décomposition de l'horizontal et du vertical, privilégiant le plan (là ou l'Homme évolue), et focalisant l'architecte sur les points d'intérêt, sans pour autant les déconnecter. Ces représentations ne négligent pas l'aspect tridimensionnel, mais rendent la figuration plus proche des images mentales du concepteur, habitué à les manipuler.

Le calque
Bien plus qu'un papier « semi-transparent » comme nous le connaissons pour recopier des dessins, le calque fut longtemps pour les architectes le papier à dessiner. Cette tendance à toutefois diminuée, mais les habitudes sont tenaces (sûrement parce-qu'elles ont fait leurs preuves) et les architectes utilisent toujours du papier calque, appelé *calque d'étude*, sous forme de rouleaux de petite dimension. Ce papier est surtout utilisé comme support pour réaliser des croquis rapides, à tous moments et endroits. Mais le calque sous cette forme est aussi utilisé pour effectuer des retouches ou différentes variantes sur un dessin, plan ou croquis, en déroulant une pièce du rouleau sur la première feuille.

Dès lors, il est à prendre en compte que l'architecte peut, en plus de la technique de dessin en elle-même, faire intervenir le support physique dans sa démarche. La propriété de transparence du calque permet de l'utiliser comme un support à des alternatives issues d'une même base de départ. Mais cette utilisation est rare en architecture. En effet, le calque est principalement utilisé pour assurer la conservation de solutions satisfaisantes tout au long du projet. Il fait alors office de mémoire, permettant de constituer des *couches* dans l'élaboration des solutions au problème. Lorsqu'une solution semble convenable, l'architecte la conserve et rajoute un calque par dessus pour continuer à travailler sur des parties encore insatisfaisantes ou tout simplement pour élaborer de nouvelles solutions. Le projet évolue, sans que les nouvelles solutions ne remplacent jamais vraiment les anciennes, ou toutefois en gardant une trace de leur évolution, leur mise en relation étant assurée par la transparence.

L'architecte Louis I. KAHN, avait associé le calque au fusain. Outre les apports « transparents » du calque, l'utilisation du fusain lui permettait d'estomper, quasiment effacer des traits de la main lorsque

ceux-ci ne le satisfaisait pas. La conjonction fusain-calque fait alors qu'il reste une trace du trait sur le calque, mémoire du dessin (la figure 1.8, croquis de phase préliminaire de conception de Louis I. KAHN, montre à plusieurs endroit des corrections faites par le concepteur).

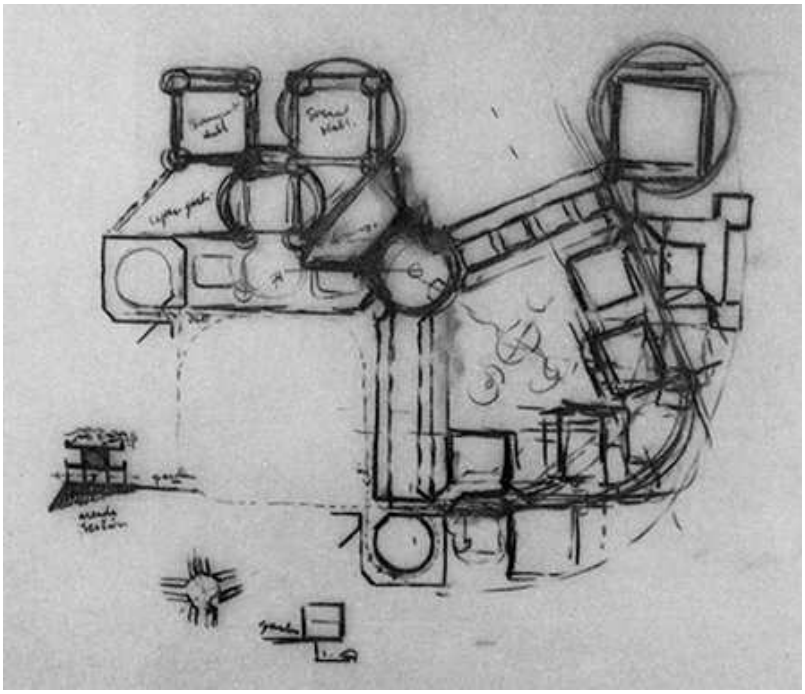


FIGURE 1.8 – Calque et fusain. Un dessin de conception de l'architecte Louis I. KAHN, réalisé au fusain, sur du calque. Ce croquis porte la mémoire des étapes de la conception, par les traits, leur superposition et leur persistance. Image tirée de *The Louis I. Kahn Collection*, Univeristy of Pennsylvannia, `http://www.design.upenn.edu/archives/majorcollections/kahn.html` .

Finalement, il nous paraît clair que le dessin spéculatif, le dessin de la pensée (de l'analyse aux solutions), se caractérise avant tout par la liberté que prend l'architecte pour arriver à ses fin. Liberté qu'offre le croquis, de par sa simplicité d'emploi. Il est en effet courant d'observer un (ou plusieurs) concepteur(s) en train de griffonner sur un coin de table ou sur un sous-bock à bière[(5)], comme le montre la figure 1.5 au début de cette section. Le croquis est accessible à tous moments et en tous lieux. Mais la notion de liberté se retrouve aussi dans les aspects techniques du dessin de conception de l'architecte. Il va utiliser toutes les représentations figuratives dont il dispose (croquis, en perspective ou plans, coupes, élévations, calques) pour supporter son processus de création mental, dégagé de beaucoup d'aspects techniques (Michel-Ange disait qu'« On ne dessine pas avec sa main ; mais avec son cerveau. »). La figure 1.9 page ci-contre est une esquisse d'étude tirée de [Lebahar, 1983] qui illustre bien ce propos de mélange des techniques sur un même dessin.

Dès lors que nous avons vu les trois activités principales de la conception architecturale (diagnostique, simulation graphique et modèle de construction), ainsi que les différents types de représentations graphiques qui s'y entrecroisent (dessins spéculatif, descriptif et prescriptif), nous pouvons nous poser la question de la place que tient l'informatique dans ce processus.

[(5)]Nous devons cette vision empreinte de convivialité (le coin de table) et de « Belgitude » (le sous-bock) à nos fréquentes et fructueuses discussions sur le sujet avec Louis-Paul UNTERSTELLER et Pierre LECLERCQ.

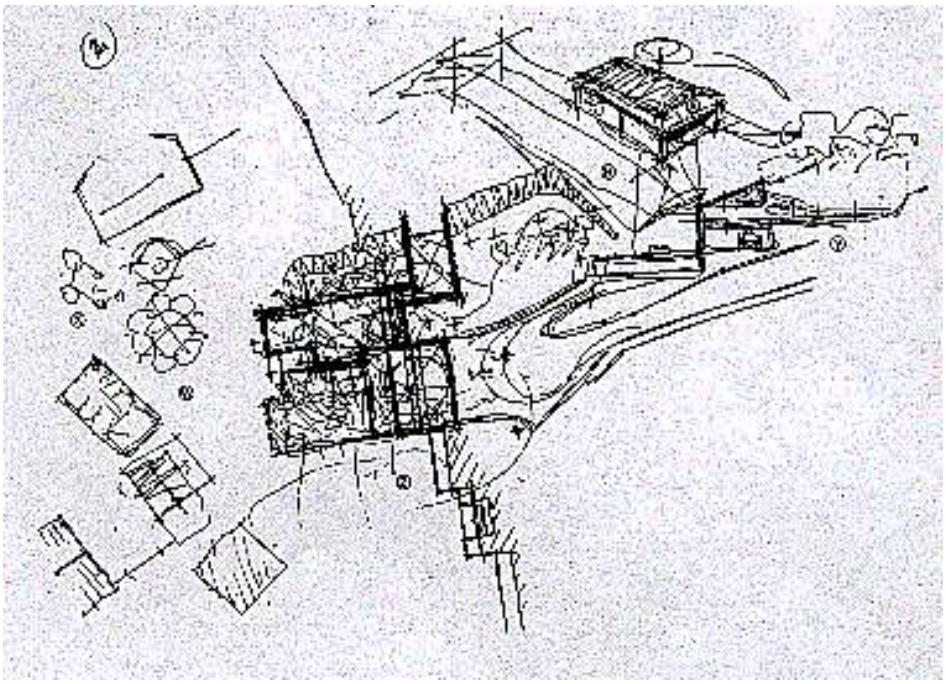


FIGURE 1.9 – Esquisse d'étude. Cette esquisse tirée de [Lebahar, 1983] illustre l'emploi de différentes pratiques figuratives pour supporter la conception architecturale (organigramme, croquis perspective, plan, coupe).

1.3.3 La CAO et l'informatique en architecture

De par les capacités de calcul et de représentation qu'ils offrent, les outils numériques ont rapidement été perçus comme pouvant aider à la conception, en particulier en architecture : simulations, rendus graphiques, archivage, etc. Historiquement (nous en reparlerons au début du prochain chapitre), l'ordinateur a d'abord permis de produire des images, des *sorties* graphiques. Ces premiers systèmes datant des années 70 ne proposaient pas d'interfaces graphiques et de fonctionnalités *temps réel* comme nous les connaissons aujourd'hui. L'objectif principal était alors la spécification et la modélisation des outils mathématiques nécessaires à cette production de représentations graphiques. Puis progressivement, ces systèmes ont évolués vers l'interactivité en conservant l'existant, proposant une démarche centrée sur la représentation graphique sans vraiment présupposer des méthodes de travail des concepteurs (en particulier en architecture). Ces outils, dit de CAO (Conception Assistée par Ordinateur), sont à notre avis « passés à côté » en faisant de cette représentation graphique un *but*, alors que comme nous l'avons vu tout au long de ce chapitre, c'est un *moyen*, un *outil*.

La place actuelle

La CAO tiens de nos jours une place importante dans la majorité des bureaux de conception architecturale. Place ouvrant même des possibilités inédites, selon l'architecte de renom Franck O. GEHRY : « The Guggenheim museum in Bilbao and the Walt Disney Concert Hall (in Los Angeles) could not exist today if we hadn't met Dassault, because there was no way to explore these kinds of shapes and make them economically feasible » [Day, 2004].

Si l'on écarte toute considération de partenariat et de publicité pour la firme produisant le logiciel CATIA [Dassault Systemes, 2002 2004] utilisé par l'équipe de l'architecte, cette phrase situe tout à fait la situation actuelle de l'informatique dans le processus architectural : la vérification du matériellement possible. La modélisation 3D extrêmement précise qui a été produite à partir des concepts de l'architecte a permis de *valider* leur faisabilité (matérielle et économique), mais elle n'en n'a pas été le support. Nous ne renions en aucun cas cet apport. Mais force est de constater qu'il n'a pas aidé le concepteur dans sa démarche première, dans les premières étapes de la conception[(6)]. Il est toutefois évident que cette démarche d'utilisation de l'informatique fait partie du processus de conception, car il est probable que les premières simulations ont engendrées des retours en arrière, des modifications du projet. Mais pour le concepteur, nous sommes d'avis que « tout était déjà joué » au niveau conceptuel.

Le constat est donc assez aisé à faire. Si l'on met en relation la CAO et le dessin de l'architecte, celle-ci n'intervient qu'à des niveaux *descriptif* et *prescriptif*, dans les phases avancées du projet. La CAO permet de produire des maquettes virtuelles, à des fins de présentation.

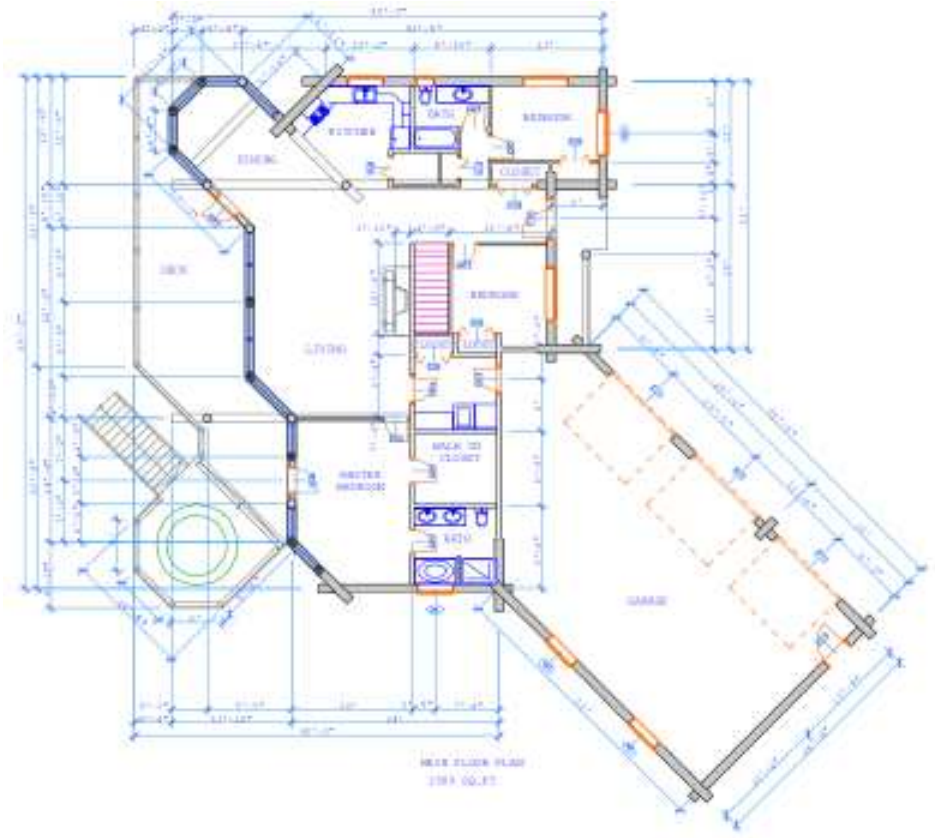
La CAO offre aussi la possibilité, par des modèles construits au fur et à mesure du projet, d'orienter des choix de conception ou de vérifier des contraintes qui n'ont pas encore été prises en compte dans les premières phases. Cela s'étend au niveau prescriptif par l'utilisation de logiciels spécialisés à l'architecture, permettant la correction automatique de plans, ainsi qu'une production aisée des dessins de construction, des prévisions de coûts, etc. (en fournissant des ensembles de symboles normalisés, en intégrant des bases de données de matériaux ou de fournitures spécifiques, par exemple). La figure 1.10 page suivante illustre ce propos.

La CAO est donc largement utilisée dans la conception architecturale (nous en voulons pour preuve la place croissante qu'elle prend dans les cursus des écoles d'architecture), mais n'est pas encore beaucoup plus qu'un support à l'ingénierie. Il est d'ailleurs amusant d'observer dans des magazines spécialisés en architecture, ou dans des discussions sur internet, les craintes qu'émettent nombre d'architectes ou d'étudiants de tous âges en regard de la place que prend la CAO dans leur domaine. Beaucoup semblent s'inquiéter d'une possible disparition du dessin classique au profit des technologies informatiques. Même si nous ne partageons pas ces craintes, étant convaincus au moins autant qu'eux de l'importance du dessin dans la conception créative, nous comprenons toutefois leurs inquiétudes, probablement la conséquence de leurs références en matière de logiciels. Car il serait en effet dramatique, même si c'est improbable, que les outils actuels de CAO deviennent les papiers/crayons de demain. Il est alors important de bien définir les apports envisageables par les futurs systèmes informatiques, en se focalisant sur leur insertion dans le processus de conception et leur adaptation au concepteur (plutôt que le contraire).

Les apports envisageables

Il est admis par la communauté de chercheurs travaillant sur ce sujet que la CAO, et plus généralement les systèmes informatiques, peuvent être un atout évident dans les premières phases d'un projet architectural. L'ordinateur pourrait devenir un partenaire performant dans l'analyse d'un problème, et en particulier pour l'accès à la connaissance. Mais il ne doit toutefois pas devenir un « filtre », fermant l'espace des possibilités du concepteur. Cette idée d'accompagnement s'étend aussi à la génération de solutions. Mais encore une fois, il ne doit pas non plus devenir le guide de la génération de ces solutions. C'est dans cette optique d'assistance que se sont orientés les travaux dans le domaine des

[(6)] Franck O. GEHRY dit d'ailleurs dans le même article qu'il est incapable d'allumer un ordinateur...



(a) Plan de construction.

(b) Rendu 3D.

FIGURE 1.10 – La CAO à l'heure actuelle. Les logiciels actuels de CAO permettent d'établir des plans de construction (figure (a)), ainsi que des rendus de présentation (figure (b)) dans un même environnement.

Systèmes à Base de Connaissances, dont nous reparlerons plus en détail dans le chapitre suivant [Gero et Maher, 1993; Flemming et Woodbury, 1995; Guena, 1997; Boucard, 2004].

Dans une même optique, les capacités de calcul des ordinateurs, associées à des modèles numériques et des simulations sur les facteurs importants de la conception architecturale (éclairement, acoustique, par exemple), peuvent être un atout supplémentaire dans les premières phases de la conception architecturale. Bien que ces notions ne soient évidemment pas écartées par les architectes, elles sont toutefois difficile à envisager précisément dans les toutes premières phases du projet, nécessitant une représentation numérique déjà aboutie du projet. Pourtant, leur prise en compte au plus tôt dans la conception peut être une source supplémentaire de créativité [Leclercq *et al.*, 2004], ouvrant l'espace des solutions possibles et ajoutant des paramètres à son évaluation.

Du point de vue des représentations graphiques, l'informatique peut aussi former une liaison entre le *spéculatif* et le *descriptif*, en permettant d'obtenir des visualisations avancées (en trois dimensions) dès les premiers croquis. L'architecte ne considère pas réellement l'aspect *volumique* dans les premières phases de conception, et de toute façon il fait preuve d'une maîtrise de cette dimension même à partir de ses croquis en deux dimensions [Lebahar, 1983; Estevez, 2001]. Toutefois, cette association permettrait peut-être d'intégrer *l'enveloppe* aux autres préoccupations plus fonctionnelles et conceptuelles du début de projet et d'intégrer les simulations dont nous parlions précédemment. Dans cette optique, notre préoccupation première n'est pas forcément de savoir si les architectes utiliseraient forcément et systématiquement de tels outils, et si ceux-ci seraient un apport indispensable pour tous. L'architecture n'en a pas eu besoin jusqu'alors. Notre démarche ne consiste pas non plus à imposer de nouvelles méthodes de conception, ce que l'informatique actuelle a plutôt tendance à faire. Notre propos est plutôt axé sur le fait que si l'on ne propose pas de telles possibilités, leurs éventuels apports ne pourront être explorés. Nous en revenons donc à l'insertion transparente de nouveaux outils dans la démarche de conception architecturale, de telle manière que l'architecte soit le seul à décider si il va les utiliser ou non.

De ces apports possibles de l'informatique dans les premières phases de la conception architecturale, il ressort surtout le besoin préalable de composer un modèle numérique, en deux ou trois dimensions, de l'objet en cours de création et d'établir un dialogue avec le concepteur (comme le dessin, d'ailleurs). En effet, bien qu'encore en constante évolution, les systèmes de représentation de la connaissance ou de simulation que nous avons évoqués, deviennent utilisables de nos jours. Le problème est leur insertion au plus tôt dans la résolution du problème, au moment où tous les paramètres des solutions envisageables n'ont pas été définis, lorsque l'architecte utilise encore des procédés figuratifs flous. La solution est à notre avis d'orienter le dialogue entre le concepteur et la machine dans la même voie que celui qui s'installe avec son dessin. Cela implique alors la prise en compte de cette activité dessin comme interaction avec le système, avec tout ce que cela suppose en terme d'interprétation (pour en tirer les informations utiles), mais aussi en terme d'adaptation à la tâche (techniques d'interaction, retours graphiques, visibilité du système) : la saisie du modèle numérique doit s'effectuer de manière transparente lors des activités figuratives de l'architecte.

Au delà de la vision du système informatique de bureau d'étude, cet *interfaçage* par le dessin entre le concepteur et l'ordinateur revêt un aspect primordial dans le cadre de l'*informatique nomade*. La conception architecturale, comme tout autre activité créative d'ailleurs, est une activité de tous lieux et de tout moment, pouvant s'inspirer de situations ou d'endroits particuliers. Il est à notre avis encore plus difficile d'utiliser les outils de CAO actuels sur un banc ou dans le métro que dans un bureau. C'est pourquoi un tel paradigme d'interaction (le dessin), associé à un TabletPC, composerait un outil similaire au classique carnet de croquis de l'architecte, avec en plus la dimension ajoutée par

la numérisation, l'analyse et l'interprétation des dessins. Cette notion peut même être poussée encore plus loin avec la notion de *papier augmenté* [Guimbretière, 2003].

En plus de ces considérations sur la démarche *individuelle* de conception qui font l'objet de nos travaux, il est aussi important de souligner les atouts indéniables que l'informatique peut fournir dans le cadre du *travail collaboratif*. La conception mets en jeu la participation de nombreux acteurs et faciliter leur collaboration est aussi un enjeu majeur de la recherche sur les processus de conception [Béguin et Darses, 1998] et en informatique [Candy et Edmonds, 2002; Mamykina *et al.*, 2002].

Enfin, des apports à ne pas écarter, même si ils ne relèvent pas non plus spécifiquement de notre domaine de recherche, concernent les aspects émergeants de la *réalité virtuelle* et de l'informatique enfouie (*ubiquitous computing*). L'architecture connaissait déjà le domaine de *l'architecture éphémère* [Monin, 2001], elle découvre depuis quelques années le domaine de *l'architecture virtuelle* où des projets architecturaux entièrement virtuels ne sont plus voués à la construction. Ainsi, la compétence des architectes s'applique à la création de lieux virtuels, de par leur maîtrise de l'espace, du fonctionnel, mais aussi de l'esthétique[(7)].
À l'inverse, le projet architectural standard, voué à être construit, commence à intégrer l'informatique (notamment dans le cadre de l'informatique enfouie) dans son rapport à l'espace, aux bâtiments et à l'infrastructure. Ce ne sont pas que des décors ou des rajouts, cela fait partie intégrante du bâtiment et des concepts sous-jacents. Dans ces perspectives, architecture et informatique sont déjà intimement liées pour avancer vers une certaine rupture avec les conventions.

1.4 Conclusion

Nous avons, dans ce chapitre, tenté d'apporter une vision globale sur les processus cognitifs que met en jeu l'activité de conception créative. N'étant pas expert de ce domaine, nous avons essayé d'y porter un regard aussi détaché et modéré que possible, considérant toutefois cette connaissance comme une base indispensable à la conception de systèmes informatiques *créatifs* (nous entendons par la *qui supportent l'activité humaine créative*).

Ainsi, nous avons souligné en quels termes l'outil informatique pouvait favoriser la créativité dans la résolution de problèmes de conception créative en complétant et favorisant les activité d'*exploration*, de *génération de solutions* et d'*évaluation*. Il en ressort alors l'enjeu majeur de *l'intégration*, la *transparence* de l'utilisation de l'informatique dans la démarche créative. Bien que nous accordions aussi un certain crédit à une relative prise en compte des modèles cognitifs pour concevoir des systèmes ouverts à la création, il nous parait primordial que cette intégration passe par l'utilisation des techniques figuratives utilisées par le concepteur : le dessin sous toutes ses formes.

Nous nous sommes placés dans un cadre précis de conception créative, l'architecture. Il y a deux justifications à ce choix. La première est que, bien que l'on puisse dégager des notions génériques et communes à tous les domaines de la conception, les spécificités et détails de chacun d'eux sont à notre avis la clef de leur intégration dans la démarche. Les tentatives de standardisation montrent souvent leurs limites, et en particulier dans le domaine de l'interaction où les techniques actuelles s'articulent autour de la spécialisation à la tâche de l'utilisateur. Dès lors, il était important de choisir un domaine d'application pour nos travaux, ce qui n'est pas incompatible avec l'application de nos

[(7)]Des agences comme *Assymptote* (`www.assymptote.net`) se sont établies dans cette voie, ou l'agence *MVRDV*, dont nombre de projets sont voués à rester virtuels.

solutions à d'autres domaines, dans une démarche identique (voir le chapitre 5 page 125). La seconde justification est notre proximité et notre connaissance de spécialistes de l'architecture qui ont facilités notre appréhension de ce domaine.

La fin de ce chapitre précise donc les particularités de la conception architecturale, et notamment l'importance que tient le dessin dans ce processus, à notre avis encore plus marquée que dans les autres champs de la conception créative (ce que suggère la citation de Franck HAMMOUTHÈNE en début de ce chapitre). Nous avons précisé la place actuelle de l'informatique dans la conception architecturale, mais surtout expliqué les apports envisageables. Ces apports possibles se situent en grande partie au niveau des phases préliminaires, posant alors le problème de leur intégration. Car en effet, comme nous le verrons dans le chapitre suivant, les logiciels actuels souffrent d'une profonde inadéquation avec ces activités, tout spécialement en terme d'interaction et de saisie des données numériques : ce que l'on appelle d'une manière générale la *modélisation 3D*.

Finalement, nous pouvons clore ce chapitre, en reprenant la définition de CAO que nous avions citée au début de cette étude : *« Conception assistée par ordinateur ou C.A.O., ensemble des techniques informatiques qui permettent l'élaboration d'un produit nouveau. La conception assistée par ordinateur a de nombreuses applications dans l'industrie automobile et l'aéronautique »*. Nous avions noté l'aspect paradoxal de cette définition qui, bien que le terme contienne le mot conception, ne réfère guère aux aspect créatifs de la conception que nous avons soulignés dans ce chapitre. Les constats que nous feront dans le chapitre suivant confirment la justesse de cette définition en montrant que les logiciels actuels de CAO ne supportent pas (ou très peu) la créativité, et ne sont après tout que des *techniques informatiques*, des outils de « Construction d'Artefacts sur Ordinateur ».

Chapitre 2

Créativité, modélisation 3D et IHM

« Les ordinateurs sont inutiles. Ils ne savent que donner des réponses »

P. PICASSO.

Sommaire

2.1 Introduction

L'objectif de ce chapitre est double. Dans un premier temps, nous confronterons les conclusions que nous avons tirées de l'analyse du processus de conception architecturale, et donc des apports possibles de l'informatique, avec l'existant en termes de logiciels de CAO. Nous focaliserons cette étude sur les techniques utilisées pour la saisie des données numériques, la *modélisation (2D ou 3D)*. Cette mise en rapport des deux démarches nous permettra d'identifier l'inadéquation de ces systèmes avec l'activité créative dans les premières phases de la conception. Nous verrons en particulier que cette inadéquation se situe à deux niveaux :

1. au niveau *cognitif*, par une démarche trop différente.
2. au niveau *contextuel*, par l'emploi de paradigmes d'interaction inadaptés.

Nous exposerons alors des solutions permettant de réduire cet écart tout en restant dans la démarche et l'environnement des outils actuels.

Dans un second temps, nous dresserons un état de l'art des travaux de recherche qui tendent à rendre la saisie de modèles 3D plus naturelle, ou du moins en relation plus étroite avec les méthodes de conception créative, par une rupture plus ou moins prononcée avec l'existant. Nous distinguerons trois grandes approches :

1. les approches *contraintes et connaissances*, misant plus sur l'intégration dans la démarche de conception par l'aide à l'analyse du problème et au diagnostic, mais moins sur les paradigmes d'interaction.
2. les approches *interactives expérimentales*, misant sur l'introduction de nouvelles techniques de modélisation et paradigmes d'interaction.
3. les approches *interactives scientifiques*, misant sur l'étude de la tâche de l'utilisateur (en l'occurrence l'activité de dessin, support primordial à la tâche de conception) et de ses méthodes de travail pour fournir des interactions adaptées.

2.2 CAO et saisie du modèle

Si les ordinateurs ont d'abord été des calculateurs, leur capacités de calcul associées à l'apparition des tables traçantes et des écrans ont très rapidement été pressenties comme des moyens de productions d'images pour l'étude, la simulation et même l'art. La figure 2.1 page suivante est la première image produite par un ordinateur (1960), réalisée par William FETTER, à qui l'on crédite aussi le terme de *Computer Graphics*. Mais bien au delà de son aspect historique, cette image est intéressante car elle a été produite dans le cadre d'un problème de conception, William FETTER étant alors concepteur de la firme BOEING, et travaillant sur l'optimisation de l'espace dans les cockpits d'avion. Très peu de temps après, la thèse de Ivan SUTHERLAND [Sutherland, 1963] introduisait *Sketchpad*, système considéré comme la première interface graphique, qui permettait de créer des images très précises sur l'écran à l'aide d'outils de dessin (lignes, points, coins) et un crayon optique, le « light pen ». Ces deux faits subliment à notre avis deux points fondamentaux de l'histoire de la CAO : la première image numérique est l'œuvre d'un *concepteur*, le premier système informatique permettant de produire des images de manière interactive est basé sur un paradigme de *dessin*.

Depuis cette révolution jusqu'à nos jours, un grand nombre de systèmes informatiques basés sur la création graphique ont été développés et sont largement utilisés. Proposant toujours plus de fonc-

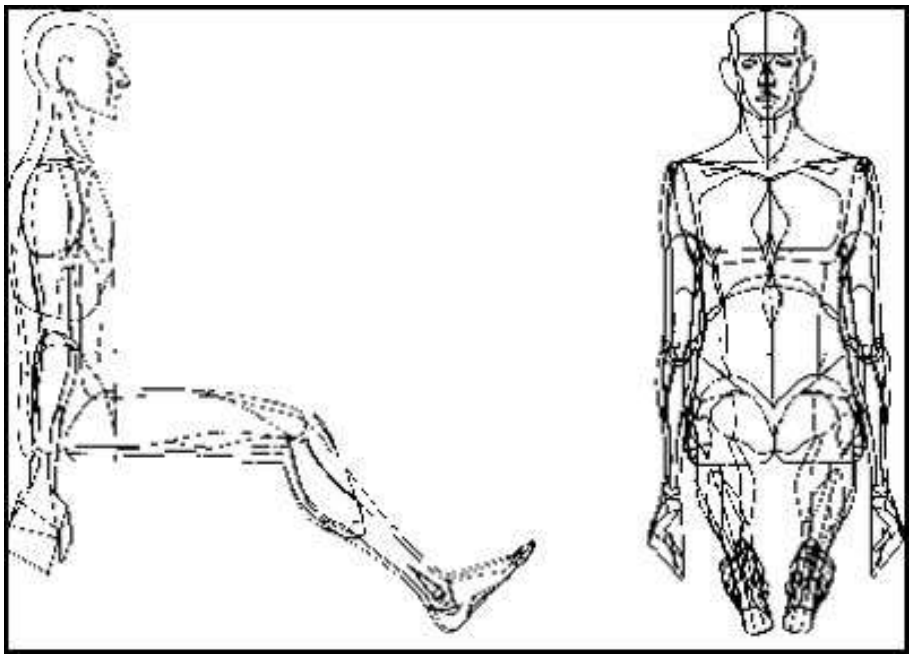


FIGURE 2.1 – The Boeing Man. Cette image générée par ordinateur est considérée comme le premier résultat de *Computer Graphics* (1960). Elle fut produite par le concepteur William FETTER alors qu'il travaillait sur la conception de cockpits d'avions.

tionnalités, ils intègrent et permettent diverses activités, de l'art numérique au cinéma en passant par la conception (design, architecture, mécanique, automobile, ...). Le point commun à tous ces logiciels, qu'ils soient dits de synthèse d'image, de CAO, de DAO ou autre, est la nécessité de saisir une représentation numérique des objets à représenter (une description géométrique dans l'espace). Dans le cadre de la synthèse d'image et de la CAO, ces maquettes numériques sont le plus souvent représentées dans trois dimensions et l'on parle alors de modélisation 3D. Bien que les objectifs de ces logiciels ne soient pas les mêmes, ils n'en restent pas moins tous des modeleurs 3D, proposant ensuite des fonctionnalités étendues à leur contexte d'utilisation.

C'est donc à cette notion de modélisation 3D que nous allons nous intéresser, dans les logiciels que nous qualifierons de *généralistes* et dans les système de CAO.

2.2.1 Tour d'horizon

Logiciels généralistes de modélisation 3D

Il existe un nombre important de logiciels de modélisation 3D dits *généralistes*. Nous avons employé ce qualificatif pour signifier que ces logiciels ne s'inscrivent pas dans un domaine précis, ils sont fondés sur une modélisation de la géométrie de la scène 3D à représenter. Nous ne pouvons les citer tous, tant leur nombre est élevé depuis les logiciels libres jusqu'aux standards commerciaux : 3D Studio Max, Strata 3D, Rhino, Blender, etc.

La méthode de modélisation employée est dite *constructive*, nécessitant souvent un travail fasti-

dieux de saisie des objets géométriques sous leur forme la plus simple (points, lignes, NURBS) ou par des primitives prédéfinies (cubes, sphères, surfaces, etc.). Une foule de transformations, déformations et manipulations est disponible, le but essentiel de ces logiciels restant la production d'images ou d'animations.

Il existe toutefois pour ces systèmes des bases d'objets pré-construits mais leur assemblage et leur mise en relation dans une même scène nécessite toujours la maîtrise de leur géométrie et des transformations qui leurs sont applicables.

CAO : spécialisation et adaptation au domaine

L'évolution notable en terme de modélisation pour la CAO est l'adaptation au domaine de conception en utilisant des objets paramétrés. L'approche reste toujours dans une démarche constructive, mais l'utilisateur a à sa disposition des objets de haut-niveau propres à son domaine, dont l'assemblage est rendu plus aisé par des scenarii d'utilisation délimitant alors les constructions possibles. Un objet *fenêtre* ne peut pas être ajouté si il n'est pas placé sur un objet *mur*. La construction du modèle est alors un processus très *mécanique*, dans le sens répétitif du terme, ou l'opérateur doit, par exemple, commencer par placer un mur. Il précise alors ses paramètres : dimensions, position, couches qui le compose (matériaux de construction, isolants, etc.). Ensuite, il placera sur ce mur une porte. Il choisit alors le type de porte, la place sur le mur et règle ses paramètres, qui sont contrôlés et vérifiés par le système pour être cohérents avec le reste du projet (taille, placement, etc.).

Plus que sur l'amélioration de la démarche de construction et de création de la maquette, du modèle, les logiciels de CAO ont rapidement évolué vers un concept de *gestion du projet de conception*, utilisant alors les capacités de calcul et de précision de l'ordinateur pour faciliter, et améliorer le passage à la fabrication . Ainsi, ont été proposées des fonctionnalités de génération des plans et métrés, d'analyses de coûts de fabrication, et même de pilotage de machines outil dans les domaines de la construction mécanique. Ces logiciels s'intègrent maintenant dans le PLM, « Product Lifecycle Management » (Gestion du cycle de vie du produit).

Apparaissent alors des *suites logicielles*, toutes basées sur un modeleur 2D ou 3D, et déclinées en divers produits selon les spécificités du domaine. On peut citer les exemples de « ténors » dans ce domaine : SOLID EDGE de la société UGS (design industriel et mécanique), CATIA de DASSAULT SYSTÈMES [Dassault Systemes, 2002 2004] avec tous ses greffons dans beaucoup de domaines, MAYA de ALIAS [Alias, 2005b] (modeleur 3D intégré dans des suites adaptées : StudioTools pour le design, AutoStudio pour l'automobile, ...). La société AUTODESK, a construit autour de son produit AUTOCAD toute une série d'applications à des domaines de conception : pour l'architecture, il y a AUTODESK AUTOCAD REVIT7, mais aussi ARCHICAD™, développé comme extension à AUTOCAD par une société indépendante [Abvent, 2005]. La société NEMETSCHEK propose aussi ALLPLAN FT [Nemetschek, 2005], pour la conception architecturale, etc.

Globalement, ces logiciels tendent à s'intégrer sur la durée complète du projet. Les arguments que l'on retrouve dans leurs brochures tournent autour de l'idée « de l'esquisse à la fabrication ». Mais il est important de modérer cette notion d'intégration par le fait que ces logiciels suivent les phases du processus de conception *industriel*. Il est indéniable qu'ils offrent un support de choix pour les phases avancées d'un projet. Mais dans ses premières phases, et bien qu'employant largement les termes de *dessin*, *croquis* et *esquisse*, ils ne prennent pas en compte les aspects *créatifs* du processus de conception, ne deviennent pas des outils de figuration comme l'est le dessin traditionnel. Ils restent,

au niveau des données qu'ils traitent, des systèmes d'ingénierie et non de création.

2.2.2 L'inadéquation avec la démarche du concepteur

Le problème majeur dans la réalisation de ces systèmes a tout de suite été identifié comme « technique », plutôt qu'humain. Ainsi, les travaux se sont avant tout axées sur la façon de représenter les données géométriques, les algorithmes permettant de les manipuler, de les afficher, ainsi que leur optimisation. Puis, dans les environnements complexes de suivi d'un projet, ce sont posées les questions de la transformation des données pour le passage d'une étape à l'autre, de leur persistance (archivage, indexation), de leur conversion (formats de diffusion, passage d'une géométrie à un plan de construction, transmission des données à une machine outil).

Il en ressort alors une explosion des fonctionnalités, voire même une argumentation basée sur leur nombre et une course à leur optimisation, que ce soit pour la création du modèle (la saisie des données et de leurs paramètres) ou pour les étapes ultérieures.
Seulement, l'interfaçage de ces fonctionnalités (le noyau fonctionnel du système) avec l'utilisateur a simplement été réduit à de simples points d'entrée, basant la communication sur la représentation des structures de données du système. Cela soulève à notre avis deux problèmes majeurs, deux inadéquations :

1. une **inadéquation *cognitive***, due à une démarche de conception imposée trop éloignée de celle de l'utilisateur dans les premières phases du projet.
2. une **inadéquation *contextuelle***, due à un support figuratif et des paradigmes d'interaction très différents de ceux des concepteurs.

Inadéquation cognitive

Ainsi, l'opérateur est contraint à manipuler des entités géométriques précises et les transformations qui leurs sont applicables, dans une approche orientée système. Cette représentation des données n'est pas un problème lorsque l'objet conçu est suffisamment avancé pour devenir quasiment « tangible », mais il est évident qu'elle ne correspond pas à la vision plus conceptuelle des premières phases de la résolution d'un problème. Ces systèmes ne savent pas manipuler de telles données *floues*, *imprécises*, caractéristiques de la résolution de problèmes comme nous l'avons vu dans le chapitre 1. Le concepteur doit alors rentrer dans la démarche qu'ont fixé les créateurs du logiciel.
Dans le domaine de la conception en mécanique, où l'aspect « précision et géométrie » est une contrainte présente relativement tôt dans le processus de conception, le problème est moindre. Par contre, si l'on se place dans le contexte de l'architecture, cette démarche se révèle inadaptée aux premières phases du projet.

Tout d'abord, ce mode de construction du modèle rend primordiale l'enveloppe, la forme, par rapport à tous les autres aspects que va prendre en compte l'architecte (fonctionnel, intégration, rapports d'espace). Ensuite, l'aspect itératif et évolutif des solutions, mais surtout leur combinaison pour en produire de nouvelles est incompatible avec la rigueur imposée par la construction géométrique. Ainsi, lorsque l'architecte repasse des traits, les combines, les prolonges, ce n'est pas dans un but esthétique, mais bien dans une démarche de modification et de génération de nouvelles solutions. De telles actions et interactions sont impossibles avec un système de CAO : il est nécessaire d'avoir tout de suite la bonne solution, sous peine de devoir remettre en question toute la construction de l'objet.

Bien qu'il existe des fonctionnalités d'annulation d'action, ou de retour arrière, ce n'est que de la gestion de version.

Dans un même ordre d'idée, nous avons évoqué l'importance que tiennent des supports comme le calque pour l'architecte. Les systèmes de CAO proposent aussi des calques, mais ceux-ci n'ont pas la même sémantique. Ce sont des couches permettant une séparation modulaire, organisationnelle du travail (niveaux d'un bâtiment, séparation sols/murs, etc.), alors que dans le dessin d'architecte, ils servent à l'élaboration graphique et la construction d'un espace de solutions.

Enfin, Daniel ESTEVEZ souligne aussi l'importance du changement d'espace de travail, obligeant le concepteur à prendre en compte des aspects qui n'existent pas avec une feuille de papier et un crayon. Il soulève ainsi les difficultés nouvelles que peuvent engendrer la gestion de propriétés qui ne sont pas visibles dans le modèle, tels que la gestion de la mémoire, la compréhension des notions d'espace image et d'espace objet, ainsi que les possibilités de pannes de la machine [Estevez, 2001].

Ces constatations montrent l'écart entre les principes de modélisation d'un objet numérique et sa création dans un processus de conception. Si l'on reprend les termes de Nigel CROSS lorsqu'il parle du dessin dans la conception [Cross, 2002] : « Le concepteur doit maîtriser un genre de médium – le croquis – permettant aux idées partiellement formées d'être exprimées et réfléchies : d'être prises en considération, développées, et reconsidérées ». Il est clair que la démarche constructive et mécanique de la modélisation 2D/3D, ainsi que le caractère précis des données qu'elle nécessite, ne sont pas adaptés aux premières phases de la conception. De tels systèmes ne peuvent évidemment pas remplacer le dessin, ni même l'utiliser comme support à la communication Homme-Machine.

Mais, en parallèle à cette question de la démarche, demeure un autre point important dans l'inadéquation des logiciels actuels pour les premières phases de la conception : un rapport, un dialogue inadapté entre le concepteur et le système informatique pour l'utiliser comme *outil de figuration* de la conception créative.

Inadéquation contextuelle

Nous avons souligné dans le chapitre précédent l'importance que tiens le dessin dans les premières phases de la conception en sa qualité d'outil figuratif, mais aussi pour la *liberté* qu'il induit dans la génération des solutions à un problème, essentiellement grâce à un rapport intuitif avec le concepteur. Il en va pourtant dans un tout autre sens pour les logiciels de CAO.

Nous avons déjà soulevé le fait que les principales préoccupations et améliorations des systèmes de CAO ont souvent été de fournir des fonctionnalités de plus en plus avancées, mais aussi de plus en plus nombreuses. Or il est clair que, dans le paradigme WIMP[(1)] de ces applications, cela se traduit par une augmentation proportionnelle de sa complexité. Cette notion a été montrée par Michel BEAUDOUIN-LAFON dans [Beaudouin-Lafon, 1997], où il calcule qu'une application *« standard offre 150 commandes dans ses menus, 60 boîtes de dialogue, et 80 outils »*. Si l'on ne considère alors que le temps nécessaire à la prise en main, l'adaptation et la découverte des fonctionnalités, ce fait rend tout de suite un système de CAO excessivement complexe dans un contexte créatif par rapport à l'utilisation d'un simple papier et d'un crayon. À cela, on peut ajouter une réduction de l'espace de travail au profit des widgets permettant l'accès aux outils (voir figure 2.2 page ci-contre).

[(1)] WIMP, pour « Windows, Icons, Mouse, Pull-down menus/Pointer » (fenêtres, icônes, souris, menus déroulants/pointeur) est le terme employé, parfois péjorativement, pour définir les interfaces actuelles.

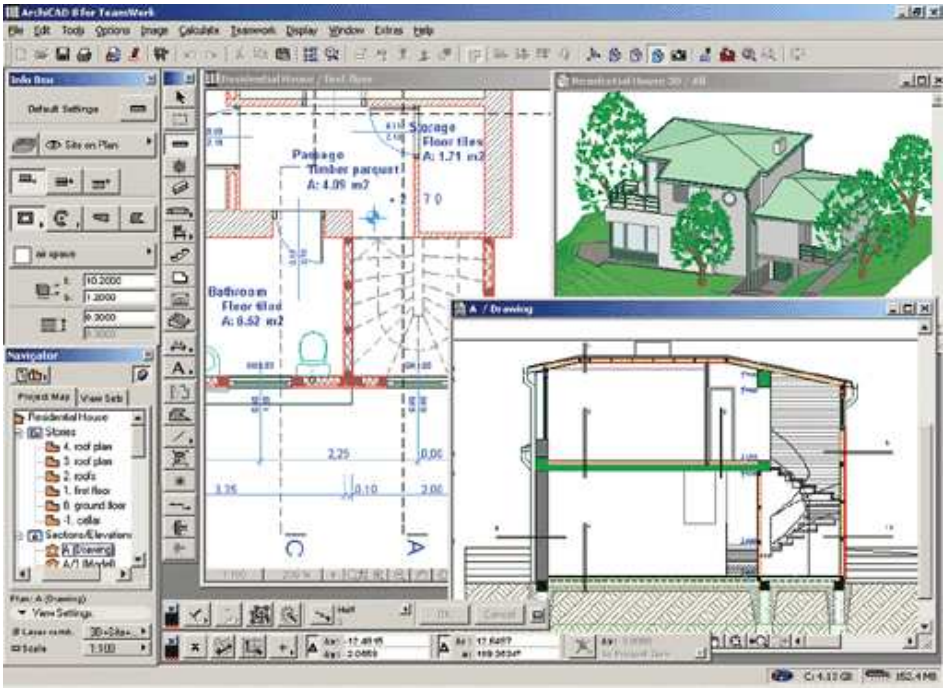


FIGURE 2.2 – ArchiCAD. L'interface d'ArchiCAD présente une partie de ses nombreuses fonctionnalités. Certes, l'espace de travail peut être maximisé en masquant barres d'outils ou fenêtres, mais au détriment de l'accessibilité aux outils.

Ainsi, les interactions sont composées d'un grand nombre de *manipulations indirectes* des objets d'intérêt (primitives de l'objet) par l'intermédiaire de boutons, menus et boîtes de dialogue et de quelques *manipulations directes* sur les objets. De plus, la multiplication des possibilités entraîne à la restriction de ces techniques de manipulation directe. La figure 2.3 page suivante, par exemple, présente la saisie d'un trait par une technique de dessin *avancée*. Le principe est de fournir un affichage permanent des caractéristiques du trait manipulé ou tracé, ainsi que de permettre la saisie au clavier de ces caractéristiques directement sous le curseur. Il est clair que cette technique évite des allers-retours entre une zone de saisie et le dessin tout en offrant à chaque instant une vision des propriétés d'intérêt. Elle est sans nulle doute adaptée au dessin d'ingénierie. Elle n'a par contre aucun intérêt ni avantage dans les premières phases de la conception, voire l'inverse (distraction et focalisation du concepteur sur des détails encore à négliger sous peine de fermer des chemins).

C'est cet aspect que nous appelons *l'inadéquation contextuelle*, le fait que, hormis le biais qui existera toujours dans le cadre de l'utilisation d'une machine, ces systèmes proposent un environnement de conception loin des habitudes de travail des utilisateurs, de leur contexte privilégié. Dès lors, quel dialogue, quelle communication peut s'instaurer entre le système et le concepteur ? Il nous semble que l'aspect *rigide*, *fermé* et purement fonctionnel de ces interfaces est un autre des aspects bloquant leur utilisation dans un contexte créatif.

Nous en voulons pour preuve l'intimité et la proximité qu'entretiennent le concepteur et son dessin dans un environnement traditionnel qui, comme le note justement Gabriela GOLDSMITH, permettent des évasions, gribouillages et autres manipulations physiques du support (pliage, coupures) [Goldsmith, 2002]. Rien ne peut prouver que ce rapport à l'environnement de dessin est un facteur

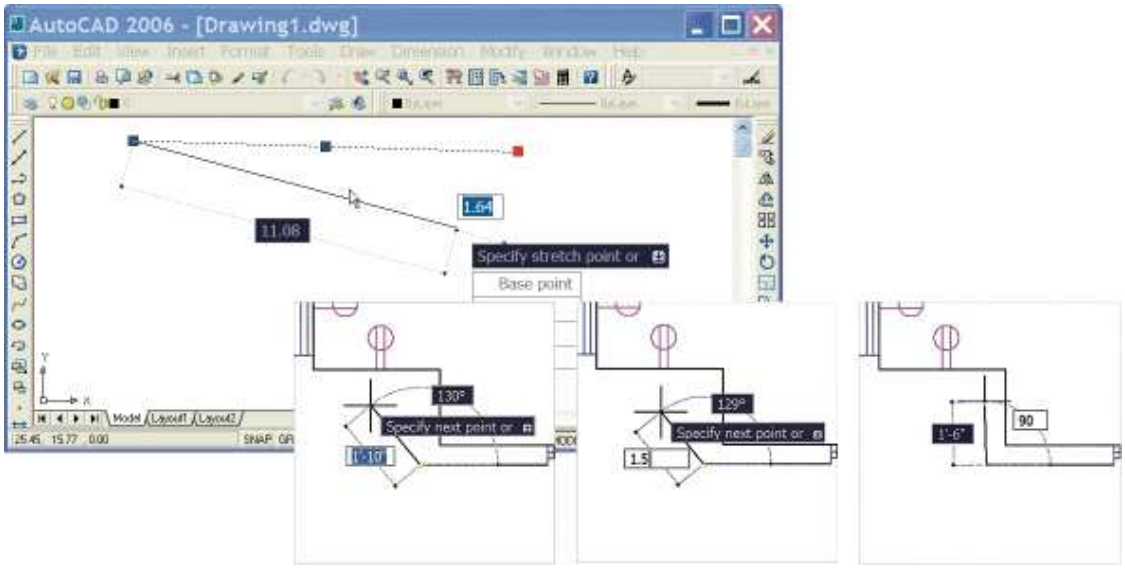


FIGURE 2.3 – Interaction avec AutoCAD® 2006. La fenêtre principale du programme, actuellement dans la vue dite de *dessin*. Les détails montrent les étapes d'une *technique de dessin*.

nécessaire à la créativité, mais le fait est qu'il existe et qu'il n'est pas retranscrit par les systèmes informatique actuels. Nous avons d'ailleurs toujours été surpris d'entendre des étudiants, architectes ou éditeurs de logiciels qualifier les systèmes de CAO d'*outils de dessin*, tant l'appréhension d'un système informatique est à l'heure actuelle différente et éloignée de leurs habitudes et de leurs *outils*.

Cela nous mène alors à nous poser la question de savoir si les logiciels de CAO, et plus particulièrement dans la saisie du modèle, peuvent être considérés comme des *outils* ? Daniel ESTEVEZ aborde cette notion de manière intéressante et balancée dans la fin de son ouvrage [Estevez, 2001]. Sa vision peut être résumée par le fait que ce ne sont pas des outils au sens où les définissent les ergonomes, et ceci en partie à cause du fait que ce sont des *machines* (Il cite d'ailleurs à ce propos Hanna ARENDT : *« L'outil le plus raffiné reste au service de la main qu'il ne peut ni guider ni remplacer. La machine la plus primitive guide le travail corporel et éventuellement le remplace tout à fait. »* , citation que nous reprendront en partie pour illustrer nos travaux dans le chapitre 4 page 83). Par contre, il montre que les logiciels de dessin sur ordinateur *« induisent un mode d'ustensilité nouveau au sein du travail humain (y compris lorsqu'il est d'ordre spéculatif et cognitif), ustensilité qui aboutirait à une forme de partenariat entre l'homme et l'appareil »*. Ce partenariat se traduit par la répartition des tâches entre le concepteur et la machine. Et de conclure : *« C'est qu'un outil de figuration doit augmenter la faculté du concepteur et non la restreindre en le délestant précisément... de ses tâches figuratives »*. Nous partageons cette vision, du moins pour les premières phases de la conception. Même si l'on peut créditer ces systèmes d'outils pour les phases avancées de la conception (qu'il n'ont pas réellement bouleversées, mais améliorées et facilitées), il est clair qu'il ne sont pas des outils de figuration dans le sens où l'est le dessin pour l'architecte. Nous pourrions même arriver plus facilement à cette conclusion en constatant simplement qu'ils ne peuvent pas être qualifiés d'outils lors des premières phases de conception du fait qu'il ne sont pas utilisés à ce moment là.

Nous avons pourtant vu dans la section 1.3.3 page 26 les apports envisageables de l'informatique dans les premières phases de la conception, ainsi que la nécessité d'intégration que cela implique.

Quelles sont alors les pistes pour qu'un tel système devienne **un outil figuratif** des premières phases du processus de conception créative ?

2.2.3 Des débuts de réponse ?

Cette inadéquation relègue actuellement la modélisation d'objets numériques, et donc l'intervention de l'ordinateur, aux étapes avancées du projet de conception. L'architecte, par exemple, va effectuer ses recherches et concevoir une ou plusieurs solutions en employant ses outils traditionnels de figuration : le dessin sous toutes ses formes. L'étape de modélisation est alors souvent réalisée par un spécialiste de l'infographie, devant reprendre, interpréter et modéliser ce que le concepteur a produit.

Pourtant, il est important de noter que les systèmes actuels peuvent, dans une certaine mesure, induire une démarche créative de la part de leurs utilisateurs. C'est ce que nous constaterons dans la section suivante. Mais nous verrons aussi, avec l'exemple du logiciel SketchUp, que ces systèmes peuvent être améliorés tout en restant inscrits dans une démarche et un paradigme d'interaction « standard ».

Association de logiciels et usage transgressif

Bien que n'étant pas une réelle utilisation des systèmes de CAO dans les premières phases d'un projet, il est à noter l'importance que prennent de nos jours l'utilisation combinée de logiciels, ou leur utilisation transgressive dans un but créatif. Ainsi, il est fréquent que des utilisateurs se servent de plusieurs systèmes pour arriver à leurs fins, ou même en détournent certains de leur vocation originale, qu'ils en ignorent des fonctions inutiles pour le but qu'il veulent atteindre [Estevez, 2001].

Association de logiciels
Il est fréquent dans les étapes descriptives d'un projet de conception architecturale, que les architectes aient recours à des outils de retouche d'images (type Photoshop®) pour améliorer, compléter ou modifier le rendu d'une scène 3D produite avec un système de CAO. C'est une tendance aussi visible dans le design industriel, où les designers utilisent ces mêmes logiciels de « peinture numérique » pour concevoir l'aspect de leurs objets, avant de simplement projeter ces images comme textures sur un modèle 3D représentant la forme générale de l'objet. Ainsi, il n'est pas nécessaire de modéliser précisément les détails, ces textures peuvent être modifiées plus facilement par la suite mais aussi combinées plus facilement (il est plus simple de modifier ou retoucher un dessin type « image » qu'une maquette virtuelle en 3D).

Soulignons d'ailleurs, que la société éditrice de logiciels de CAO Alias (ex Alias Wavefront) [Alias, 2005b], souvent précurseur dans le domaine de l'IHM, a introduit de tels outils dans ses modeleurs. Il est ainsi simple de combiner le dessin 2D et la modélisation 3D dans un même environnement. Outre le côté pratique d'un environnement unifié, cette combinaison de modèle 3D pour l'enveloppe et de textures dessinées pour les détails permet plus de souplesse dans la conception avec un ordinateur, l'intégrant alors plus tôt dans le processus.

Usage transgressif
Cette notion d'*usage transgressif* est mise en valeur dans l'épilogue de l'ouvrage de Daniel ESTEVEZ [Estevez, 2001] d'où nous tirons d'ailleurs ce terme. Il est en effet prudent, comme le suggère l'auteur, de ne pas faire tendre ces inadéquations vers un rejet total des systèmes d'infographie actuels. Il

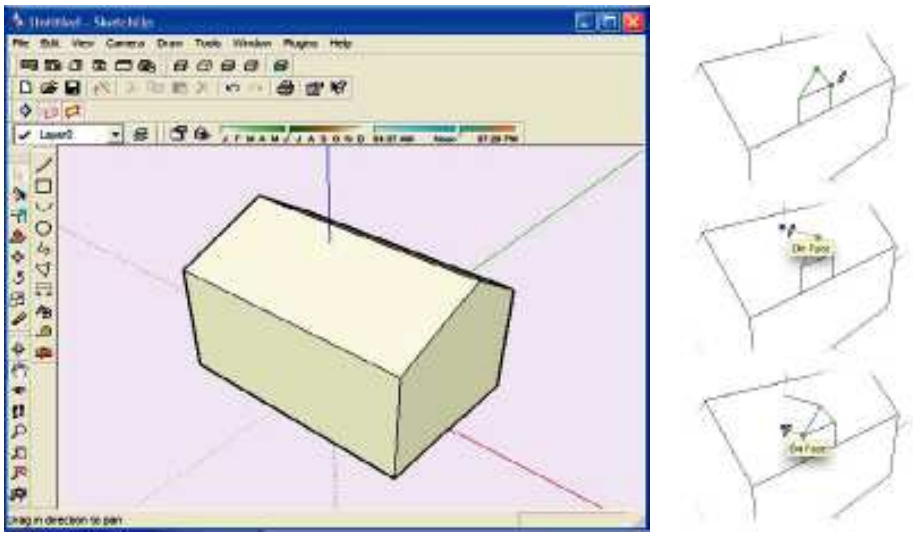
considère pour cela les capacités d'adaptation et d'anticipation de l'Homme lorsqu'il poursuit un but précis, ce qui est le cas dans une tâche de conception. Ainsi, l'usage transgressif est la capacité de ne pas s'enfermer dans la démarche guidée des systèmes actuels, mais de l'utiliser à bon escient selon ses buts et ses moyens : n'utiliser que les fonctions utiles à un moment donné, utiliser des fonctionnalités a priori prévues pour un autre but, alterner les supports (passer du dessin à l'infographie, et repasser au dessin, etc...).

Nous partageons cette vision, mais nous y répondons toutefois que quel que soit l'outil informatique et son domaine d'application, il en sera toujours fait un usage détourné ou transgressif, constatant que c'est surtout la démarche imposée par le logiciel qui induit la transgression, ou sa flexibilité qui en favorise le détournement. Est-ce une raison pour ne pas assimiler ces modes d'utilisation, ou les intégrer dans une démarche plus proche de l'utilisateur, moins guidée et donc plus libre (bien que poursuivant un but précis) ?

Sketchup

Le logiciel **SketchUp** [@Last Software, 2000 2004] marque un pas important vers une modélisation 3D interactive, plus adaptée aux concepteurs et à leurs habitudes figuratives. Ainsi, il propose une large palette d'outils d'interaction essentiellement basés sur la manipulation directe et masquant donc l'aspect « géométrique » de la modélisation 3D traditionnelle (ce qui en a rapidement fait un logiciel très apprécié des architectes). Il est ainsi possible de modeler interactivement les formes à l'aide d'un seul outil interactif permettant de les pousser, les tirer, etc. selon un principe combinant extrusion et opérations booléennes (cet outil n'est d'ailleurs pas sans rappeler les principes de modélisation 3D proposés dans le système **SCULPTOR**, un outil de CAO expérimental basé sur la manipulation des espaces [Kurmann, 1995; Kurmann *et al.*, 1997]). Un autre outil permet de tracer des chemins, selon un principe d'étirement de lignes en 3D proche du dessin vectoriel, avec des aides contextuelles appropriées (comme par exemple l'indication interactive d'appartenance à un plan dans la figure 2.4(b) page ci-contre). Bien entendu, ce logiciel propose aussi des bases d'objets prédéfinis adaptées à divers domaines (architecture et construction, mécanique dans une moindre mesure), ainsi que divers modes de visualisation (coupes et rendus graphiques avancés).

L'utilisation de ce logiciel rend la modélisation 3D plus intuitive et son apprentissage plus rapide. *SketchUp* représente une étape importante, dans un domaine ou l'industrie tarde encore à relâcher un peu les aspects fonctionnels et l'efficacité de calcul au profit de plus d'interactivité. Car bien évidemment, il ne possède pas toutes les fonctionnalités d'une suite logicielle de CAO complète. Toutefois, sans être vraiment du dessin à proprement parler, il peut déjà permettre un premier rapprochement entre les fonctions descriptives et spéculatives du dessin. Il convient toutefois de rester modéré quant à une utilisation spéculative dans le sens de la conception architecturale, les outils interactifs de ce logiciel n'offrant tout de même pas la souplesse des opérations figuratives des traits du dessin à main levée (révision des solutions et itération, émergence d'idées). C'est ce que nous avons pu remarquer dans les témoignages publicitaires des utilisateurs professionnels de ce logiciel qui ne parlent que des avantages descriptifs de celui-ci, en particulier des possibilités de rapidement modifier le modèle en temps réel devant le client. C'est certes un atout, sûrement une introduction plus précoce de l'outil informatique dans la conception, mais nous sommes d'avis que ces architectes ont d'abord conçu les bases de leur projet avec du dessin traditionnel.



(a) Interface globale. (b) Dessin « 3D ».

FIGURE 2.4 – *SketchUp*™ [@Last Software, 2000 2004]. Ce logiciel introduit des fonctionnalités plus « intuitives », inspirées du dessin, pour la création de modèles 3D.

2.2.4 Synthèse

Le bilan que nous venons de dresser peut se résumer simplement : le principal obstacle à ce que les systèmes actuels de CAO deviennent des outils de support à la créativité dans les premières phases de conception se situe principalement dans leurs interactions avec l'utilisateur, et donc dans la manipulation des données et du dialogue. Nous avons situé cette inadéquation à deux niveaux :

- Au niveau de la démarche, tout d'abord, trop axée sur une approche constructive et nécessitant tout de suite des données précises, un modèle établi qui ne peut pas prendre en compte les aspects *flous* et *itératifs* caractéristiques de l'activité créative. C'est ce que nous avons appelé **l'inadéquation cognitive**.
- Au niveau des paradigmes d'interaction et du support figuratif, ensuite, qui trop liés à la démarche sus-citée, n'accordent pas la liberté et le pouvoir d'expression des outils traditionnels (le dessin, le croquis qu'il soit géométral ou perspective). C'est ce que nous avons appelé **l'inadéquation contextuelle**.

Malgré de constantes améliorations les logiciels actuels ne permettent toujours pas la *conception* de modèles numériques. Ces améliorations passent par des primitives de construction plus malléables et spécifiques, par l'adoption de dispositifs d'entrée plus adaptés (tablettes, dispositifs 3D, etc.), mais ne changent ni la démarche, ni le paradigme d'interaction global. Surtout, elles n'améliorent pas les capacités du système à supporter la spéculation.

2.3 Faciliter la création de modèles numériques

Les solutions que propose la recherche pour l'amélioration des techniques de modélisation, 3D en particulier, sont nombreuses. Selon notre problématique, nous avons choisi de les regrouper en trois grandes catégories :

1. les *approches contraintes et connaissances*, qui regroupent les techniques de modélisation par contraintes, déclarative, à base de connaissances, mais aussi les techniques interactives de reconstruction 3D à partir d'images. Ces techniques s'inscrivent le plus souvent dans un paradigme d'interaction standard (WIMP), mais s'orientent par contre vers une démarche plus proche des processus de conception.
2. les *approches interaction-expérimentales*, qui utilisent des paradigmes d'interaction nouveaux et originaux pour établir un dialogue plus adapté entre le concepteur et le système, sans pour autant remettre vraiment en question la démarche constructive.
3. les *approches interaction-scientifiques*, qui proposent une démarche de modélisation ainsi que des interactions fondées sur des observations et études de la conception créative.

2.3.1 Approches contraintes et connaissances

Nous ne rentrerons pas dans les détails de tous les systèmes proposés dans les domaines de la modélisation 3D par contraintes, déclarative ou à base de connaissances. Nous évoquerons tout de même certaines de ces méthodes pour l'aspect innovant de leur démarche de construction de scènes 3D, plus proche de la démarche imprécise et itérative des concepteurs. Elles entrouvrent aussi la possibilité de les combiner entre elles pour améliorer encore leur efficacité, mais aussi de les associer par exemple à des système de reconstruction à partir d'images. Ce sont aussi des voies intéressantes pour une éventuelle association avec des approches mettant plus l'emphase sur l'interaction comme celles que nous présenterons dans la section suivante.

Modélisation déclarative et/ou à base de connaissances

Modélisation déclarative
La technique de *modélisation déclarative* est basée sur le principe de la construction itérative d'une scène 3D à partir de sa description par l'utilisateur. Ainsi, celui-ci décrit la scène, dans un formalisme plus ou moins naturel (du langage de script au langage naturel) et dans des termes plus ou moins précis (très grand cube, plus ou moins prés de). Le système construit alors des propositions de scènes 3D couvrant l'espace des solutions possibles à la description. L'utilisateur peut alors, à partir de ces propositions, raffiner sa description pour diriger le système vers la création de solutions plus précises. Cette technique est aussi qualifiée de *modélisation par contraintes*, reposant d'un point de vue système sur des techniques de résolution de contraintes (les déclarations de l'utilisateur). Nous pouvons citer comme travaux dans ce domaine ceux de Stéphane DONIKIAN [Donikian et Hégron, 1993] et aussi le projet MULTIFORMES [Ruchaud et Plemenos, 2002], encore en cours d'amélioration actuellement. Ce système à même été utilisé comme base d'un modeleur déclaratif à base de connaissance, spécialement dans le domaine de l'architecture [Ravani *et al.*, 2003].

Cette méthode propose une démarche itérative et imprécise mieux adaptée aux processus de conception que les démarches classiques de construction géométrique. Des contraintes de départ

servent à l'élaboration d'un espace de solutions, solutions auxquelles seront appliquées de nouvelles contraintes permettant de raffiner le champ des possibles jusqu'à une solution « acceptable ». Il est d'ailleurs intéressant de noter que les auteurs des travaux que nous avons cités dans le paragraphe précédant emploient l'expression *croquis de scènes 3D* (« 3D scenes sketch modeling ») pour qualifier cette démarche, de par l'évolution en parallèle des propositions du système et de l'image mentale du concepteur. Mais cette analogie avec le croquis est ambiguë et inexacte, voire même erronée. En effet, nous avons constaté dans le premier chapitre l'importance du croquis dans un cadre figuratif et spéculatif. Surtout, nous avons vu que le dialogue établit entre le concepteur et ses techniques figuratives lui permettait d'élaborer des solutions. Or, dans cette démarche déclarative, c'est au système informatique qu'est déléguée la construction de solutions. Mise à part les problèmes techniques dus à la « compréhension » et à la résolution de contraintes imprécises par le système, cette délégation a d'autant plus d'impact sur la démarche créative que le système doit nécessairement faire un choix entre un grand nombre de solutions possibles, soulevant alors dans un même temps le problème de leur présentation à l'utilisateur. Au contraire, le croquis n'impose rien et n'opère pas à un premier tri des solutions, il figure et suggère un espace de solutions, leur génération et leur évaluation étant du domaine du concepteur.

Bases de connaissances

Les approches de modélisation basées sur la connaissance visent à fournir des outils informatiques de représentation des connaissances d'un domaine, mais aussi de raisonnement sur ces connaissances, afin d'appuyer le concepteur dans sa démarche. Ces travaux présentent un intérêt certain dans l'insertion des outils de modélisation dans les premières phases de la conception, notamment pour l'interprétation et la manipulation par le système des données imprécises qui en résulte.

Les travaux de François GUÉNA, par exemple, s'appuient sur le *raisonnement par classification* pour classifier et identifier un objet par rapport à ceux déjà présents dans la base de connaissances [Guena, 1997]. Cette approche permet essentiellement de compléter la description d'un objet par les propriétés de la classe à laquelle il a été rattaché. Bien que ce formalisme n'ait pas été exploitée plus profondément dans un système de conception architecturale[(2)], elle laisse entrevoir des possibilités intéressantes pour l'interprétation des données imprécises que représente le dessin dans les premières phases de la conception architecturale. Cela soulève toutefois divers problèmes, tels que la construction de classes d'objets adaptées au raisonnement de l'architecte ou le mode de dialogue entre le système et son utilisateur (suggestions et propositions). Ce type de raisonnement a d'ailleurs été proposé dans le système **GIDeS** [Fonseca *et al.*, 2004], associé à une interface de saisie de dessins.

Dans le même esprit, le principe de modélisation 3D associant résolution de contraintes et bases de connaissances proposé par Didier BOUCARD dans sa thèse de doctorat [Boucard, 2004] rend possible la reconstruction de scènes 3D à partir de données imprécises ou incomplètes. Il utilise pour cela un corpus d'objets et de règles issus des critères de l'architecture classique, qui associé à un solveur de contraintes va permettre de construire et placer les objets architecturaux que l'utilisateur a spécifié. Il est alors possible de manipuler des entités de plus ou moins haut niveau, le système complétant les données manquantes et assurant le respect des règles du domaine. Bien que spécifique à la reconstruction de scènes architecturales classiques (à partir de documents tels que des plans ou des récits historiques), cette démarche ouvre des pistes vers un modeleur 3D spécialisé à l'architecture [Boucard et Colin, 2002]. Il nous parait envisageable qu'une telle méthode puisse assister la conception architecturale, notamment par ses capacités à compléter l'information manquante dans la description

[(2)]Le raisonnement par classification a été pour l'heure utilisé dans un exemple d'application pédagogique pour la composition de parois et dans un projet plus complexe de reconstruction de voûtes d'ogives.

de la scène.

Dans un cas général, les approches de modélisation basées sur des contraintes ou des connaissances ne nous paraissent pas comme étant une solution à part entière pour l'intégration de l'outil informatique dans les premières phases de la conception architecturale. Malgré leur démarche itérative, utilisant des données imprécises caractéristiques des premières étapes de la conception, le principal problème est qu'elles ne fournissent pas à notre avis le support figuratif nécessaire au concepteur (comme le fait le dessin). De plus, ces systèmes visent souvent à la production d'un ou plusieurs résultats « coûte que coûte », pouvant conduire à une limitation du champ des possibles dans un cadre créatif.
Toutefois, nous pensons que cette capacité à la manipulation de données imprécises peut se révéler un atout majeur pour un système de CAO « créatif », notamment pour l'interprétation et la saisie de dessins. Il serait alors primordial de définir précisemment les connaissances du domaine pour que celles-ci ne deviennent pas une limitation, mais aussi de trouver l'accord juste pour que le système propose à l'utilisateur, mais jamais n'impose.

Reconstruction à partir d'images

Il nous parait important d'évoquer le domaine de la reconstruction 3D à partir d'images dans cette section sur l'amélioration des techniques de modélisation 3D. Si l'on écarte les méthodes « automatiques » issues de la photogrammétrie, les approches interactives proposées dans ce domaine sont une piste intéressante pour rendre la modélisation 3D plus abordable dans les premières phases de la conception.

Nous ne voulons pas ici établir un état de l'art de ces méthodes, le lecteur peut pour cela se référer à nos travaux antérieurs [Huot, 2000; Huot et Colin, 2001]. Il est toutefois important de souligner que des méthodes interactives, comme celle introduite par Paul DEBEVEC dans [Debevec *et al.*, 1996] et reprise dans le logiciel **CANOMA** [Adobe, Inc, 2004], se basent sur la connaissance de l'utilisateur et la description graphique qu'il fait d'une photographie pour en reconstruire une scène 3D. Ainsi, il suffit de préciser les contours et la nature des objets significatifs présents dans une image pour que le système en déduise la scène 3D. Dès lors, une telle approche peut aussi être utilisée afin d'obtenir rapidement une reconstruction à partir d'un croquis de conception réalisé sur papier, comme nous l'avions proposé dans [Huot, 2000] (voir figure 2.5 page suivante). [Boucard *et al.*, 2002] et [Hégron *et al.*, 2000] proposent de plus l'utilisation conjointe de la reconstruction interactive à partir d'images et des bases de connaissances pour améliorer les techniques de reconstruction 3D.

Ces méthodes ne constituent pas un apport direct pour la modélisation 3D dans les premières phases de la conception et ne peuvent être utilisées comme telles. Toutefois, les techniques mathématiques sous-jacentes qu'elles ont introduites pour résoudre le problème de la reconstruction en trois dimensions d'une image en deux dimensions répondent à des préoccupations proches de celles de l'obtention d'un modèle 3D à partir d'un dessin. Il reste tout de même à prendre en compte le problème de l'interprétation en temps réel d'un dessin à main levée et des modes d'interaction avec l'utilisateur. C'est pourquoi nous allons nous intéresser plus précisément à des propositions visant à rapprocher, d'un point de vue interaction, la saisie de modèles numériques (principalement en 3D) de l'activité de dessin.

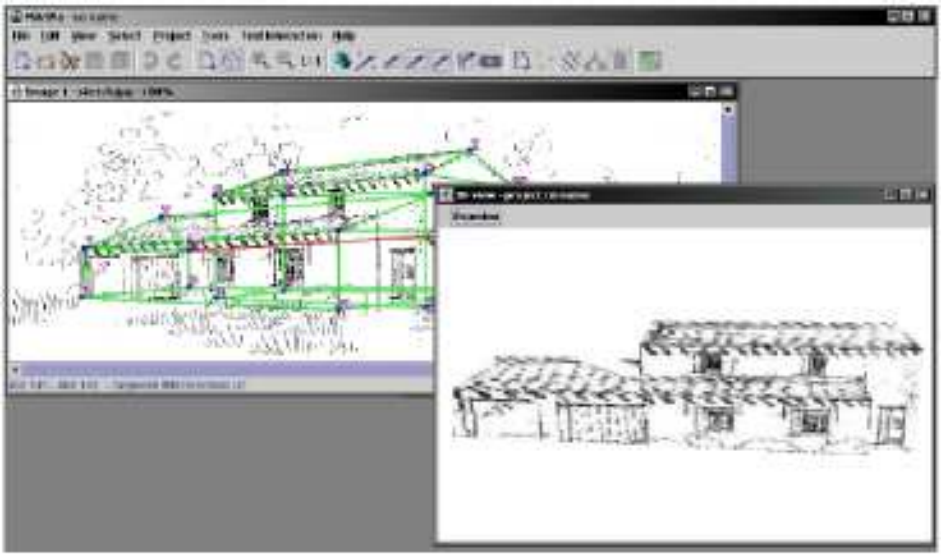


FIGURE 2.5 – MArINa. Le système que nous avons proposé dans [Huot, 2000] permet de reconstruire rapidement une scène 3D à partir d'une image, mais aussi d'un dessin.

2.3.2 Approches par l'interaction

De par leur conception même, les interfaces post-WIMP[(3)] [van Dam, 1997] sont conçues pour être adaptées aux applications et outils spécifiques à un domaine ou à une tâche. Ainsi, plutôt que de viser une standardisation des méthodes d'interaction, leurs concepteurs tentent de répondre au plus juste à un besoin précis, dans un cadre donné. Les interfaces de modélisation 3D novatrices que nous allons présenter ici sont toutes post-WIMP. Elles poursuivent en général deux buts : rendre la modélisation 3D plus abordable aux débutants et tirer parti des systèmes informatiques dans les processus créatifs (pour les experts). Nous distinguons deux orientations dans ces recherches. Une approche **expérimentale**, visant à introduire des paradigmes nouveaux et inhabituels. Une approche que nous qualifions de **scientifique**, proposant des systèmes basés sur des études d'utilisateurs et de leurs méthodes de travail.

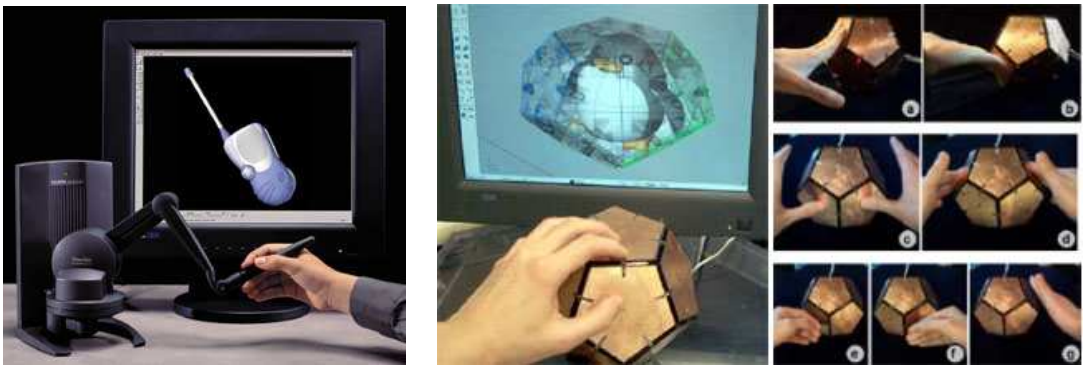
Approches expérimentales

Nous présentons dans cette section des travaux en rupture avec les techniques de modélisation « classiques ». Ces méthodes reposent sur des paradigmes d'interaction ou des dispositifs d'entrée inhabituels par rapport aux standards WIMP, mais aussi par rapport aux techniques des concepteurs. Elles s'avèrent toutefois plus proches de ces dernières et facilitent la construction de modèles 3D dans des domaines précis. Ne pourraient-elles pas, dès lors, être un substitut au dessin comme technique figurative dans les phases créatives de la conception.

Périphériques d'entrée avancés
Une première approche novatrice pour l'amélioration des interactions dans le cadre de la construction

[(3)]Le terme post-WIMP évoque les interfaces employant de nouveaux paradigmes d'interaction, plus ou moins en rupture avec les standards du WIMP.

de modèles 3D consiste à développer des dispositifs spécifiques. Ce problème de la correspondance entre les degrés de liberté des dispositifs standards et des besoins pour la modélisation 3D a donné naissance à des dispositifs plus évolués et adaptés comme par exemple le ***Phantom*** (voir figure 2.6(a)). Longtemps resté un outil de laboratoire, ce périphérique à six degrés de liberté et à retour d'effort est de plus en plus associé à des systèmes de modélisation 3D commerciaux traditionnels, mais aussi à des logiciels dédiés comme ***FreeForm*** [Sensable Technologies Inc, 2005] afin de mieux tirer parti de ses capacités.



(a) Phantom. (b) iSphere.

FIGURE 2.6 – Périphériques d'entrée avancés. Le *Phantom* s'intègre à des applications commerciales de CAO alors que la recherche propose encore de nouvelles alternatives avec par exemple *iSphere*.

Le monde de la recherche est aussi productif dans la conception de nouveaux dispositifs d'entrée et récemment, par exemple, **iSphere** a été présenté comme un nouvel outil spécifique à la modélisation 3D [Lee *et al.*, 2005]. Ce dodécaèdre recouvert de senseurs est dédié à la création de modèles 3D par déformation de surfaces paramétriques en les poussant, les tirant et les tordant directement avec le dispositif (voir figure 2.6(b)).

Reconnaissance de gestes : SKETCH
En 1996, Robert ZELEZNIK introduisit **SKETCH** [Zeleznik *et al.*, 1996], le premier outil de modélisation 3D utilisant la reconnaissance de gestes. Ce système permet de construire des volumes polyédriques, de les éditer et de les manipuler, uniquement par interprétation de gestes tracés directement sur l'espace de travail. C'est une avancée majeure permettant de rendre la modélisation 3D plus intuitive, un premier pont entre le dessin à main levée et la CAO. La figure 2.7 page ci-contre représente une partie de l'ensemble des gestes de *SKETCH*, ainsi qu'une scène 3D exemple.

Les gestes correspondant à la création des objets ont été choisis pour leur relative ressemblance avec la primitive 3D qu'ils permettent de créer. Cela implique une mémorisation plus rapide, basée sur l'analogie, et favorise donc l'apprentissage des fonctionnalités du système. De plus, la position du geste tracé détermine aussi la position de la primitive crée, position qui peut même être relative à un objet existant (voir l'encadré en haut à droite de la figure 2.7 page suivante).

Les objets peuvent aussi, par la même technique de reconnaissance de gestes, être manipulés (translations, rotations), modifiés (étirements selon les axes, sections, extrusions, opérations booléennes), groupés, copiés et collés. Ces opérations de modification semblent toutefois plus difficiles

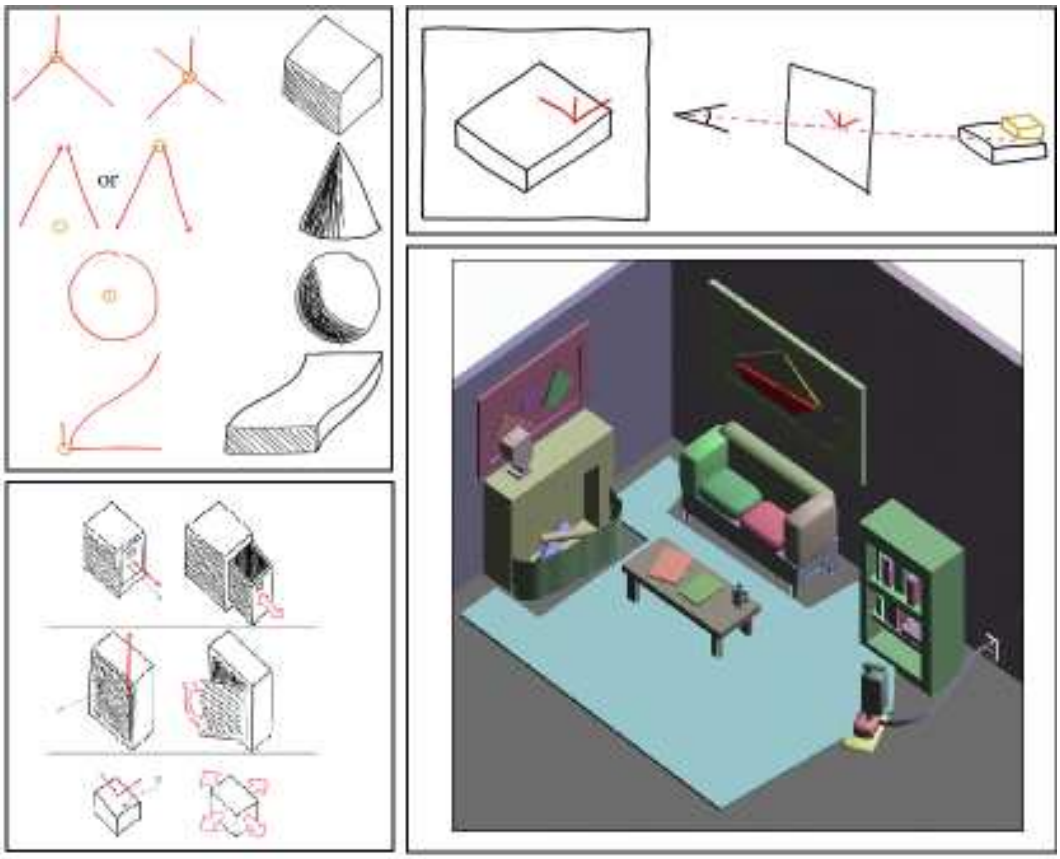


FIGURE 2.7 – *SKETCH*. Ce système permet la création d'objets 3D (en haut à gauche), mais aussi leur placement (en haut à droite) et leur édition (en bas à gauche) par reconnaissance de gestes. Ces images ne présentent qu'une partie des possibilités (notamment au niveau des objets disponibles). En bas à droite, le résultat de la création d'une scène.

à appréhender pour l'utilisateur que les opérations de création. Une dernière opération de manipulation intéressante est le repositionnement d'un objet par le dessin et la manipulation de son ombre. Il est possible de dessiner l'ombre d'un objet ; la manipulation interactive de celle-ci permet alors de déplacer l'objet.

Enfin, le rendu de la scène 3D en cours de création est volontairement non photo-réaliste, de type crayonné, en accord avec la relative imprécision des primitives créées. En effet, étant basée sur des gestes, la création des primitives n'engendre pas de valeurs précises (taille et position). Le rendu crayonné permet à la fois de signifier cette relative imprécision, mais aussi de laisser une certaine liberté d'interprétation à l'utilisateur.

Une approche similaire a été proposée dans [Pereira *et al.*, 2004], se focalisant plus sur les aspects techniques de la reconnaissance de gestes proches de dessins pour effectuer des commandes, créer des objets 3D ou leur appliquer des opérations, afin de proposer une interaction cohérente pour toutes les opérations possibles.

Dessin, gestes et suggestions : TEDDY, Chameleon et Château
Takeo IGARASHI a produit plusieurs systèmes intéressants dans le domaine des interactions pour la modélisation 3D.

Tout d'abord, avec **TEDDY** [Igarashi *et al.*, 1999], il a réalisé un mélange efficace entre le dessin et la modélisation 3D, dans un contexte bien particulier (dessin d'enfant). C'est un outil de modélisation regroupant deux modes d'interaction : dessin de formes et reconnaissance de gestes. Le principe de *TEDDY* est simple : il suffit de tracer le contour d'un volume pour que celui-ci soit interactivement déduit par le système, selon un principe d'expansion de ce contour selon les axes. Il est alors possible de « creuser », couper ou déformer ce volume, ainsi que de lui ajouter des extrusions par des gestes et des tracés simples (voir la figure 2.8).

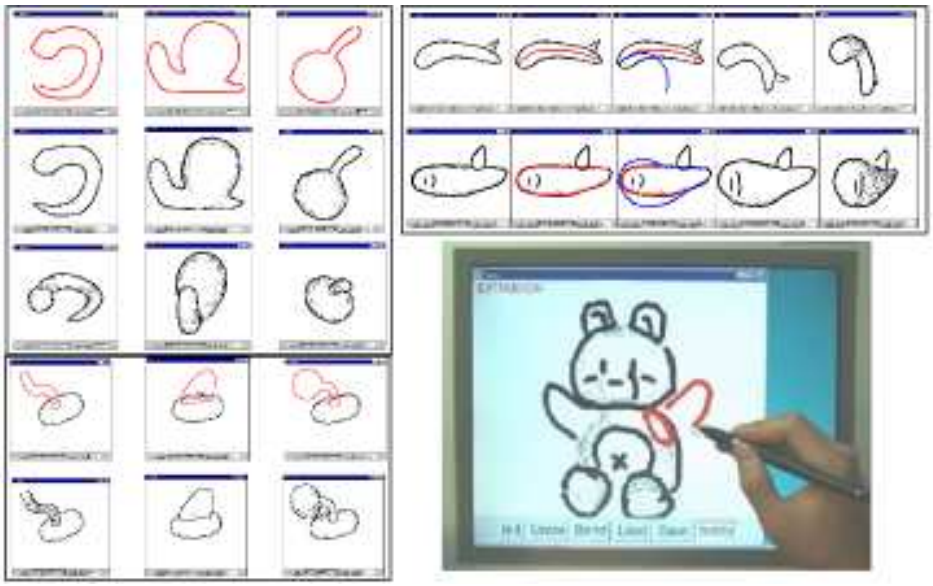


FIGURE 2.8 – *TEDDY*. Essentiellement destiné à la construction de modèles 3D par des enfants, *TEDDY* permet de construire des formes 3D en dessinant leur contour (en haut à gauche), des extrusions (en bas à gauche) et des transformations (en haut à droite).

Dans la continuité de *TEDDY*, le même auteur a proposé **Chameleon**, un système interactif pour peindre directement sur des modèles 3D [Igarashi et Cosgrove, 2001]. Il a ensuite unifié ces deux approches dans un seul système, **SmoothTeddy**, améliorant par la même occasion la qualité des formes 3D construites par le dessin de leur contour [Igarashi, 2003].

Toutefois, même si ils permettent de produire intuitivement des modèles simples, ces travaux sont toujours limités car produisant des modèles particuliers (patatoïdes) et/ou imprécis. La reconnaissance de gestes, lorsqu'elle est utilisée comme seul paradigme d'interaction comme dans *SKETCH*, pose aussi le problème d'un ensemble limité de gestes (donc de commandes et d'objets) et la relative difficulté que peut éprouver un nouvel utilisateur pour les assimiler et donc s'approprier le système.

Takeo IGARASHI a proposé un autre type d'interface, tout aussi intuitif bien que plus « guidé » : une interface suggestive pour la modélisation 3D. Le système, nommé **Château** [Igarashi et Hughes, 2001], est basé sur un paradigme de dessin vectoriel en trois dimensions. L'utilisateur, par simple technique de cliquer-déplacer trace des segments de droites qu'il peut orienter comme il le souhaite dans l'espace de travail. Afin de faciliter la création de ces segments, l'interface propose des techniques classiques de magnétisme et de calage sur une surface. Mais là où ce système innove, c'est par sa capacité à formuler des suggestions à l'utilisateur. Des agents de suggestion analysent les segments tracés et le segment en cours de tracé pour proposer automatiquement la structure qu'ils sont « entraînés » à reconnaître. Ces propositions apparaissent sous forme de vues miniatures que l'utilisateur peut valider ou ignorer afin de continuer à dessiner librement. Ainsi, comme le montre la partie gauche de la figure 2.9, il suffit de tracer deux traits pour que le système propose différentes possibilités de complétion (un plan, un triangle ou un carré).

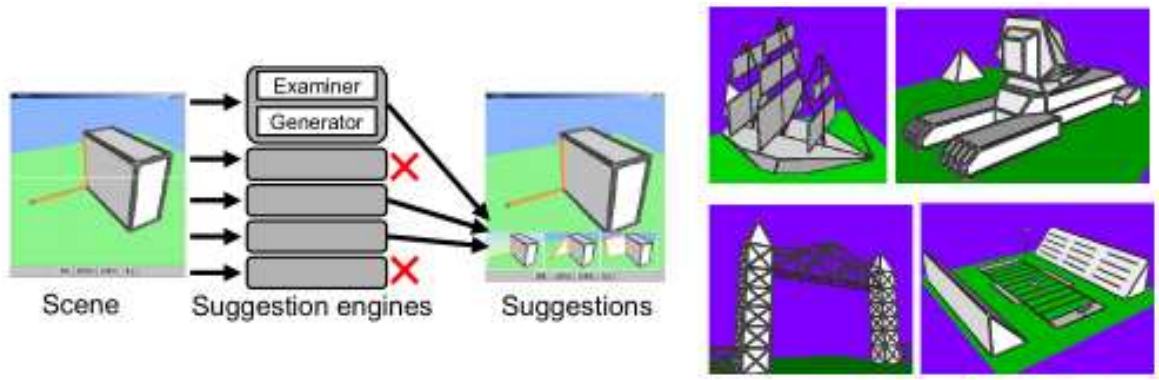


FIGURE 2.9 – *Château*. Selon un paradigme de dessin vectoriel 3D, *Château* fournit une architecture d'agents qui suggèrent de manière non intrusive des possibilités de complétion du dessin.

L'intérêt est que, tout en facilitant le dessin en complétant automatiquement des formes simples et habituelles, le système ne prend pas le pas sur l'utilisateur et se contente de suggérer des possibilités. C'est une approche bien adaptée à la modélisation 3D dans un contexte créatif, permettant de réduire les tâches fastidieuses et répétitives, tout en laissant l'utilisateur mener le dessin (il peut très bien construire un modèle complet en ignorant systématiquement les propositions du système, celle-ci ne gênant en rien la progression). *Château* n'est toutefois pas un système complet de modélisation 3D, et le paradigme de dessin à base de segment limite grandement le type de modèles que l'on peut créer (voir figure 2.9). Toutefois, l'architecture modulaire des agents de suggestion, permettant d'en ajouter simplement et surtout la technique non-intrusive employée pour proposer ces suggestions, sont des

pistes intéressantes pour l'assistance dans les systèmes plus complets de CAO.

Bilan sur ces approches
Les dispositifs d'entrée avancés, tout d'abord, facilitent certainement la manipulation et la saisie de données dans le cadre de la modélisation 3D. Ils permettent une *vraie* manipulation directe des objets d'intérêt qui fait souvent défaut avec des dispositifs standards. Cela facilite la prise en main et libère alors l'utilisateur de certaines contraintes. Ils restent toutefois peu répandus de par leur coût ou leur état de prototype. Mais de plus, dans une démarche créative, leur apport n'est pas si évident car les logiciels qu'ils contrôlent proposent toujours la même démarche constructive, le même paradigme d'interaction global et la même représentation figurative que les systèmes de CAO standards.

Les autres outils présentés sont d'un apport indéniable en facilité d'accès de par leur analogie avec le dessin. Toutefois, ils en sont relativement éloignés, notamment si l'on se place dans un contexte de conception créative. *SKETCH* ne permet que l'utilisation d'un ensemble de primitives prédéfinies et ne supporte que peux la modification et l'itération sur les modèles. *TEDDY* offre des outils plus adaptés à des formes libres et à la modification mais dans un contexte bien défini (« patatoïdes » pour les dessins d'enfants). Enfin, *Château* est contraint par un paradigme de dessin vectoriel, plus proche du dessin technique que du dessin spéculatif. Ces démarches sont toutefois une ouverture de la modélisation 3D vers l'intuitivité et la facilité d'utilisation, permettant d'avoir une idée moins précise de ce que l'on veut modéliser.

Approches scientifiques

Les systèmes que nous présentons maintenant, et que nous qualifions d'approches « scientifiques », sont plus proches des méthodes traditionnelles « sur papier ». Issus d'observations de tâches ou d'études utilisateurs, ils ont permis d'introduire de véritables interfaces de dessin pour la création de maquettes numériques (2D ou 3D).

Une approche « design » : Digital Tape Drawing
Bien que n'entrant pas dans le cadre des interfaces de dessin pour la modélisation 3D, nous avons jugé opportun de présenter ici une technique proposée dans [Balakrishnan *et al.*, 1999] pour sa qualité de transfert d'expertise.

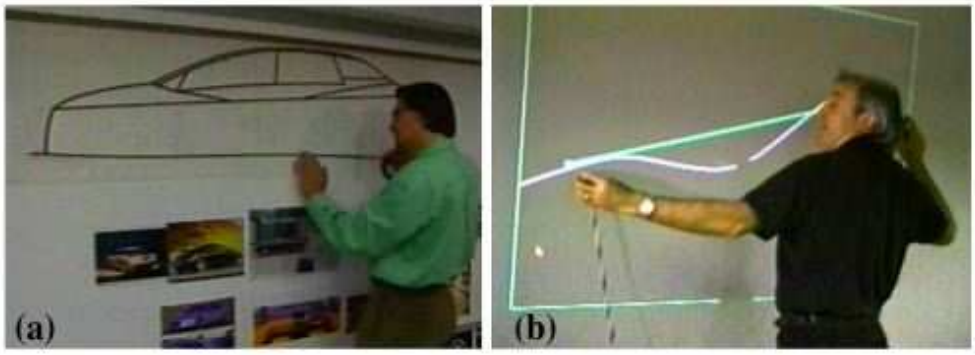


FIGURE 2.10 – À gauche, un concepteur utilise la méthode de *Tape Drawing*. À droite sa réalisation électronique, le *Digital Tape Drawing*.

Une méthode utilisée dans la conception automobile consiste à dessiner à grande échelle les contours de l'automobile avec du ruban adhésif sur une surface verticale (c'est le *Tape Drawing*, voir figure 2.10 page précédente). Par l'étude de ces usages, Ravin BALAKRISHNAN a proposé une méthode électronique, le **Digital Tape Drawing**, reproduisant en grande partie les techniques qu'utilisent ces concepteurs avec des capteurs positionnels 3D (voir figure 2.10). Ces travaux sont un exemple d'adaptation des systèmes informatiques aux capacités et à l'expertise des utilisateurs créatifs.

Une approche « architecte » : ESQUISE

Le logiciel **ESQUISE**, développé par le LUCID Group de l'université de Liège, est un prototype d'outil informatique pour l'architecte basé sur l'interprétation du dessin dans les premières phases de conception. L'idée directrice de ces travaux est d'insérer de manière transparente l'outil informatique dans la conception, par le concept que Pierre LECLERCQ nomme *l'interface absente* [Leclercq, 2004]. Ainsi, l'interface est une zone de dessin, sur laquelle l'architecte dessine à main levée le plan de son bâtiment avec une tablette graphique. Ces traits sont alors interprétés sur la base des concepts de frontières, d'espaces fonctionnels et sémantique. Le système en déduit les relations entre ces espaces pour construire un premier modèle topologique. L'ajout de légendes aux espaces fonctionnels par reconnaissance de caractères permet alors la construction d'un modèle sémantique, le système intégrant des informations standard sur les espaces fonctionnels traditionnels (cuisine, salle de bain, chambre, etc.). Juste en dessinant, et en écrivant, l'architecte a « sans le savoir » construit une représentation logique et sémantique du bâtiment qu'il réalise. Comme le montre la figure 2.11, cette représentation va être utilisée pour produire un modèle géométrique, une extrusion 3D, ou des évaluations (le prototype n'intègre pour le moment que le calcul de consommation annuelle d'énergie).

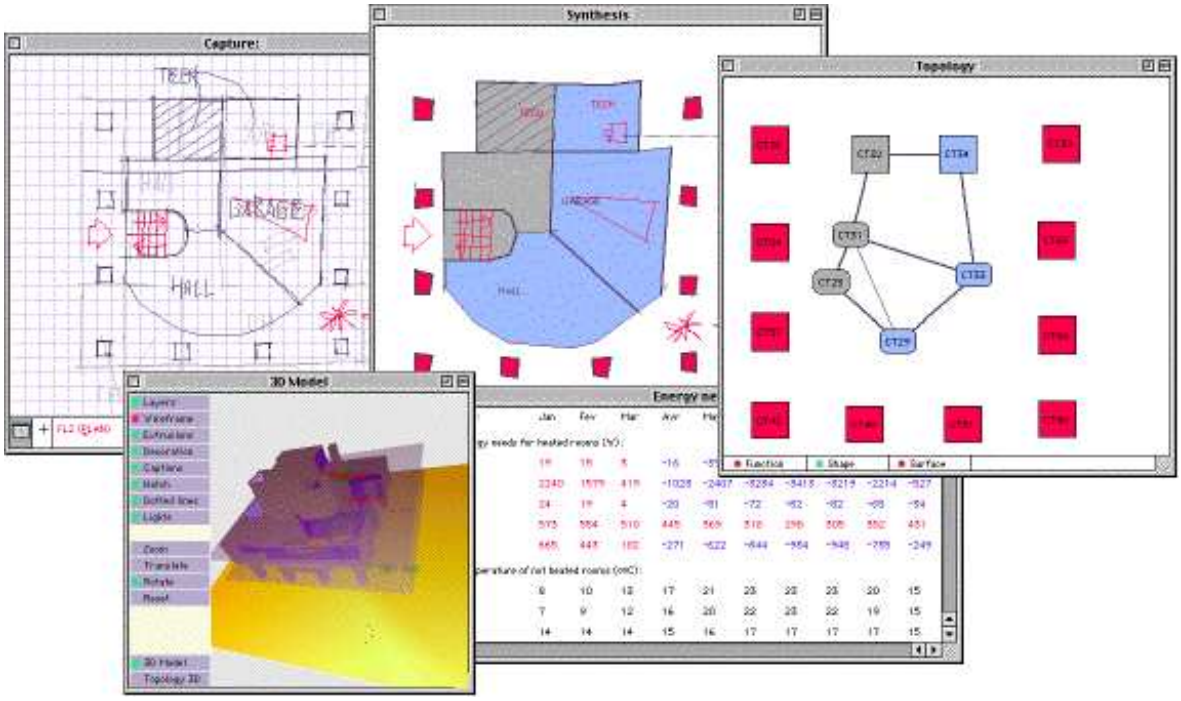


FIGURE 2.11 – L'interface de dessin et les différente vue de ESQUISE. Le dessin tracé librement (en haut à gauche) est interprété afin d'obtenir un modèle sémantique de la construction, utilisé alors pour différentes évaluation (géométrie, topologie, extrusion 3D, consommation d'énergie).

ESQUISE promet une bonne intégration dans les premières phases de la conception architecturale : il est basé sur un paradigme habituel au concepteur (dessin plan), agit de manière transparente et sans être intrusif dans l'activité de l'utilisateur. Un autre atout est la flexibilité de son architecture d'implémentation basée sur une architecture multi-agents facilitant son évolution (évolution d'ailleurs en cours de réalisation afin d'y introduire de nouveaux paradigmes d'interaction multimodale et d'orienter le système vers un bureau virtuel pour la conception collaborative [Leclercq *et al.*, 2004]).

Ces travaux sont proches de ceux de Ellen Yi-Luen DO et Mark D. GROSS qui ont proposé une plateforme et différentes applications informatiques pour supporter la conception. Ils ont notamment étudié l'importance des *diagrammes* et de leurs relations dans les premières phases de la conception afin de développer un outil facilitant leur gestion et leur interprétation [Gross et Do, 1996]. Plusieurs applications en sont issues, notamment celles proposées dans [Do, 1998] et [Do, 2000] très proches d'ESQUISE dans leurs modes d'interaction.

Le dessin pour la modélisation 3D
Construire des modèles en trois dimensions à partir de dessins a d'abord soulevé le problème de l'interprétation du dessin et de sa transformation en un modèle 3D. Ainsi, les premières solutions ont été de contraindre certains aspects du dessin (vue, révisions) afin de rendre possible son interprétation.

Del LAMB, dans [Lamb et Bandopadhay, 1990], a proposé une des premières approches efficaces de dessin 2D pour créer des modèles 3D. Le principe est d'interpréter automatiquement des dessins à main levée en vue axonométrique. Même si l'on trouve des limites évidentes à cause de la vue 2D imposée et d'une interaction limitée, ces travaux marquent les premiers résultats significatifs dans ce domaine.

Le système **Viking** [Pugh, 1992] va plus loin en interprétant un dessin à base de lignes associé à des contraintes descriptives ou structurelles spécifiées par l'utilisateur. David PUGH qualifie alors son système de *« What you draw is what you get user-interface »*. Lynn EGGLI propose dans [Eggli *et al.*, 1997] une méthode très proche de *Viking*. L'évolution majeure est le passage du dessin vectoriel au dessin à main levée. Ces deux systèmes extraient les propriétés structurelles des objets pour construire des modèles 3D. Afin de contourner les ambiguïtés dans la détection de propriétés et les problèmes de projection, la vue de l'objet dessiné doit être une axonométrie ou une coupe plane. Dans le même ordre d'idées, nous pouvons aussi citer **Digital Clay** [Schweikardt et Gross, 2000] ou les travaux menés au *Cornell Computational Synthesis Lab* [Masry *et al.*, 2004] qui, avec des algorithmes différents, présentent un fonctionnement similaire : dessiner à main levée les lignes continues du modèle à reconstruire en vue axonométrique (voir figure 2.12 page suivante).

Un système de dessin à main levée en perspective (et pas uniquement en projection axonométrique) a été proposé dans [Tolba *et al.*, 2001]. Conçu pour être utilisé dans les premières phases de la conception, cet outil de dessin projectif fournit des aides et des guides au créateur, proposant de dessiner selon des lignes de projection. Ce système parait simple à utiliser et efficace pour manipuler des vues en perspective des scènes dessinées. Il est toutefois basé sur une projection du dessin, par un principe sous-jacent de *points projectifs*, qui ne permet ni d'obtenir un modèle 3D, ni la structure spatiale des scènes et objets dessinés. Cela en fait donc plus un système de description et de communication que de conception.

Nous avons, dans cette section, tenté de proposer une vue d'ensemble des travaux les plus significatifs dans le domaine des interfaces de dessin, en particulier pour la modélisation 3D. Ce tour d'horizon non exhaustif évoque les différentes approches que l'on retrouve dans beaucoup d'autres travaux similaires (outre les conférences du domaine telles que SIGGRAPH, EUROGRAPHICS, CHI,

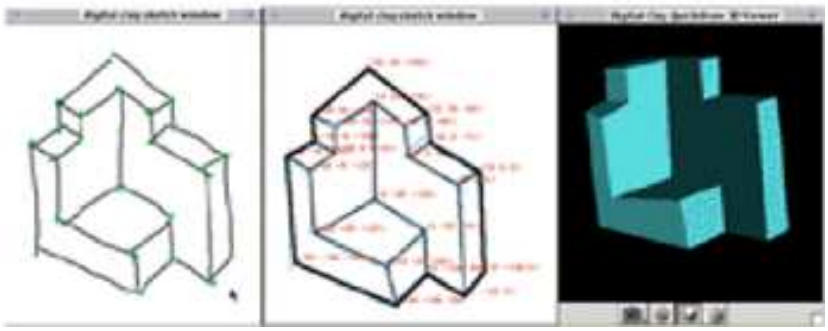


FIGURE 2.12 – *Digital Clay* permet de dessiner les lignes continues qui seront reconstruites en 3D.

UIST, etc., nous invitons le lecteur à consulter les actes du groupe de travail annuel « Eurographics Workshop on Sketch-Based Interfaces and Modeling », tenu pendant la conférence EUROGRAPHICS [sbm,]).

2.4 Conclusion

Dans ce chapitre, nous avons cerné l'inadéquation des systèmes de CAO face à la démarche du concepteur. Nous l'avons située à deux niveaux :

1. au **niveau cognitif**, d'abord, car leurs méthodes de construction d'un modèle numérique tend à imposer des choix à l'utilisateur et ne se fonde pas sur les données imprécises des phases initiales de la conception.
2. au **niveau contextuel**, ensuite, car les modes de représentation et les paradigmes d'interactions qu'ils proposent ne placent pas l'utilisateur dans un contexte optimal pour la création.

Dès lors, nous avons examiné les solutions qui ont été proposées par différents axes de recherche pour remédier à cette inadéquation. Ainsi, nous avons d'abord présenté des méthodes de construction de scènes 3D obéissant à une logique plus conceptuelle et floue (modélisation à base de contraintes ou de connaissances) et donc de fait plus aptes à assister le concepteur.

Ensuite, nous avons présenté des systèmes visant à se fondre dans les premières phases de la conception en utilisant des techniques d'interaction avancées ou tout simplement l'interprétation de dessins.

Trois constats majeurs émergent de ces travaux :

1. Bien que plus intuitifs que les systèmes dits « standards », les systèmes permettant d'obtenir des modèles 3D à partir de nouvelles interactions (périphériques d'entrée, gestes, dessin 3D) sont souvent trop contraignants pour être de véritable supports « créatifs » en comparaison au traditionnel papier/crayon.
2. Les systèmes de dessin à main levée permettent une meilleure expression des idées et des concepts, mais sont souvent contraints par un choix de point de vue, par le fait de devoir tracer directement le bon trait (ne supportant alors pas l'aspect itératif du dessin de conception) ou ne produisant pas une maquette numérique 3D.

3. Bien qu'étant basés sur des paradigmes d'interaction avancés, ces systèmes présupposent de l'utilisation de périphériques d'entrée standard (souris, clavier) ou au mieux d'une tablette graphique.

Nous nous inscrivons toutefois dans le même axe de recherche et nous pensons que le principe de l'interface de dessin est une solution envisageable pour l'utilisation de l'informatique dans les premières phases de conception. Toutefois, nous désirons aller plus loin que les travaux précédents, en proposant non seulement un paradigme d'interaction basé sur le *dessin libre*, sans contraintes de vue (dessin en perspective), ni de limitations à la saisie (révisions, repassages, calques), mais en proposant de plus des outils adaptés aux habitudes de travail des concepteurs et à leurs besoins dans leur démarche créative, supportés par une métaphore facilitant leur prise en main.

Dans le chapitre suivant, nous proposons une étude sur l'activité de dessin d'architecture en perspective que nous avons menée. L'objectif de cette étude était de permettre la construction de techniques d'interprétation du dessin efficaces, mais surtout d'en tirer des implications sur la construction d'un système informatique adapté à cette activité dans le contexte d'une tâche créative. Nous proposerons alors des lignes directrices pour la réalisation d'un système informatique de modélisation 3D de support à la créativité, sa finalité n'étant pas seulement de produire un modèle numérique, mais de s'insérer dans la démarche du concepteur pour que celui-ci exprime sa créativité et arrive à ses buts.

Chapitre 3

Du trait à un système de modélisation 3D créatif

« Ce n'est que quand il ne sait plus ce qu'il fait que le peintre fait de bonnes choses »

E. DEGAS.

Sommaire

3.1 Introduction

Nous avons souligné dans le chapitre 1 l'importance que tenait le dessin dans le processus de conception créatif, en particulier en architecture. Nous avons aussi présenté dans le chapitre 2 plusieurs approches innovantes pour la construction de modèles numériques, visant à intégrer au mieux les systèmes informatiques dans les premières phases de la conception. Toutefois, nous avons aussi soulevé certains problèmes et limites de ces approches, en particulier en termes de créativité (contraintes de vue et de style dessin).

Dans ce chapitre, nous exposons la démarche incrémentale que nous avons suivie pour tirer partie des savoirs-faire et des habitudes des architectes afin de rendre plus naturelles les interfaces de dessin numérique. Nous avons pour cela combiné des hypothèses de conception nous paraissant efficaces (approche expérimentale) avec une étude de la tâche et de son activité principale (approche scientifique) afin de s'inscrire dans une vision globale du problème. Il nous semble que cette approche globale est indispensable pour proposer des solutions à tous les niveaux (intégration de l'outil dans l'environnement, adéquation avec la démarche de conception, aspect techniques de traitement du dessin).

Dans la première section, nous énoncerons les hypothèses qui on conduit notre démarche, notamment dans le choix de certains paradigmes d'interaction et du dessin à main levé en perspective comme médium de communication. Ensuite, nous présenterons la première étape de nos travaux, consistant en une étude sur les pratiques du dessin en perspective.

Les résultats des analyses de cette étude axée sur le *trait* nous ont permis de déduire des implications importantes sur l'interface et les interactions. Ces implications, combinées à nos hypothèses de conception, à l'état de l'art et à des discussions avec des spécialistes, on conduit à la proposition de lignes directrices à l'origine d'une métaphore de *table à dessin virtuelle*.

Certains des travaux exposés dans ce chapitre ont fait l'objet de communications [Huot et Dumas, 2002; Huot *et al.*, 2003].

3.2 Hypothèses pour un outil de modélisation 3D « créatif »

À partir de l'étude du domaine et des besoins que nous avons présentés dans les deux premiers chapitre de ce mémoire, nous formulons des hypothèses de trois ordres quant à la réalisation d'un système informatique de modélisation 3D pour les premières phases de la conception architecturale, un *outil de modélisation 3D « créatif »*. Ces hypothèses sur *l'interface*, les *interactions* et la *démarche d'utilisation* guideront nos travaux depuis une étude de la tâche jusqu'à la réalisation du prototype.

3.2.1 Importance de la métaphore

Il nous parait tout d'abord primordial d'utiliser une métaphore qui suggère au plus proche les caractéristiques de l'environnement de conception habituel de l'architecte. Les métaphores courantes qu'emploient les interfaces WIMP telles que le bureau comme espace de travail, les boutons pour représenter une action ou les onglets pour manipuler différents documents ne nous semblent pas en adéquation avec les gestes et techniques habituels au concepteur.

Notre hypothèse est donc de fournir un environnement de travail *instrumenté*, utilisant des dispositifs et des techniques d'interaction particuliers et appropriés aux actions à réaliser (un stylet pour dessiner, des techniques de manipulation directe pour utiliser des calques, etc.), mais aussi des représentation graphiques et logiques adaptées (feuilles, traits).

Ce principe d'interface *écologique*, où les actions possibles sont clairement visibles et directement accessibles, tient selon nous une place importante dans la conception d'un système qui ne « bride » pas la créativité des utilisateurs.

3.2.2 Prépondérance du dessin

Nous avons souligné l'importance du dessin comme outil figuratif de la conception, et l'intérêt qu'il a alors suscité pour les interfaces de modélisation 3D. Cette activité des tâches créatives « traditionnelles » (conception, architecture, art, etc.) est donc aussi devenu une activité des tâches créatives informatisée. Nous nous inscrivons dans la même démarche, considérant que le dessin doit être le mode de saisie privilégié, mais aussi et surtout un mode de communication entre l'homme et la machine dans le cadre d'outils pour la conception créative.

Dès lors, il est important de proposer des possibilités similaires aux techniques et activités du dessin dans le cadre de tâche traditionnelles de conception créative :

- tracer des traits (de différentes épaisseurs et styles) ;
- repasser des traits ;
- utiliser et manipuler le(s) support(s) (feuilles et calques) ;
- tracer des annotations.

La médiatisation de ces activités de base peut permettre de transposer la notion de *dialogue* qui existe entre le dessinateur et son dessin si tant est que les interactions et les représentations proposées soient habituelles (épaisseur et variation des traits, touché de la pointe, brosse, etc.). Mais il est aussi primordial que d'éventuelles activités complémentaires, induites par l'emploi d'outils informatiques et la tâche qui en découle, ne brisent pas non plus ce dialogue privilégié et n'interfèrent donc pas avec le dessin.

Le système doit donc être fondé sur des interactions cohérentes, essentiellement à base de dessin et de manipulation directe du contexte (selon le même principe que, avec une feuille et un crayon, la main dominante dessine et l'autre manipule la feuille [Guiard, 1987]).

3.2.3 Liberté d'action

La démarche de création doit être laissée au libre choix de l'utilisateur. Ainsi, le système, de son interface à son noyau fonctionnel, doit s'adapter à la démarche du concepteur et non l'inverse. Cette notion de *liberté* est essentielle pour *l'exploration*, indispensable à la création. Ce qui n'exclut toutefois pas d'éventuelles aides dans cette démarche exploratoire, mais sans contraindre ni imposer.

Du point de vue du dessin, nous considérons qu'aucune contrainte de vue ou de style ne doit être imposée. Dès lors, dans le cadre de la construction de modèles 3D, le système doit être capable d'interpréter des dessins en vue perspective (cas général) à main levée, imprécis et révisés de manière itérative : des *croquis* en perspective.
Cette hypothèse peut paraître en contradiction avec ce que nous avions évoqué sur la préférence des architectes pour l'utilisation de croquis en plans et coupes [Estevez, 2001]. Il s'avère que nous retrou-

vons tout de même beaucoup de croquis en perspective sur les dessins d'architectes dans les premières phases de la conception. Dès lors, il doit être possible de faire cohabiter ces deux techniques, dans des buts différents[(1)].

Du point de vue de l'interface et des interactions, cette hypothèse se traduit par la nécessité pour le système de comprendre au mieux les actions de l'utilisateur afin de s'y adapter. Deux approches sont alors possibles : proposer implicitement les outils adaptés à l'activité courante, ou rendre les outils disponibles à tout moment.

Ces hypothèses soulignent l'importance que va tenir le dessin comme vecteur de communication entre l'utilisateur et le système informatique, mais aussi l'importance de comprendre l'activité de dessin afin de s'y adapter. C'est pourquoi, dans la suite de ce chapitre, nous présentons une étude originale du dessin architectural en perspective. Les résultats des analyses de cette étude, associés à la reprise des hypothèses que nous venons d'énoncer, nous permettront d'établir les quatre lignes directrices pour la conception de notre système.

3.3 L'étude du trait

Nous avons, depuis le début de ce mémoire, présenté le dessin comme le point commun, l'invariant à tous les concepteurs, la trace tangible qui permet la construction de leurs idées et supporte en partie le processus de conception créative. Le dessin et les traits qui le composent forment aussi un historique de la démarche qui a conduit à ces idées. Même s'il est certain que la manière de dessiner est propre à chaque concepteur, nous avons voulu déterminer des invariants, hors de toutes considérations stylistiques et culturelles.

Nous avons donc mené une étude sur la façon de dessiner des architectes lors d'un processus de conception créative, une analyse de l'activité de dessin dans ce contexte, à partir du *trait* et de ses caractéristiques. Ainsi, au lieu d'identifier et de relever directement les comportements et intentions du dessinateur pour comprendre ses actions et caractériser la tâche, nous avons choisis d'étudier les résultats de ces actions pour en extraire les caractéristiques : les traits. Ce choix se justifie par :

1. L'art, les modes de création, les impacts culturels, les effets de style, etc. sont des paramètres de la conception difficilement maîtrisables et modélisables par un système informatique. C'est pourquoi nous avons jugé important de se limiter à l'étude du dessin.
2. Les traits produits par l'utilisateur seront les données de base, les entrées que devra traiter notre système de dessin. C'est de ces données que nous voulons extraire le plus d'informations possibles afin de replonger le dessinateur dans un environnement naturel, évitant ainsi les nombreuses perturbations induites par des interactions inadaptées.
3. Comme nous l'avons vu au début de ce mémoire, nombre d'études ont déjà été faites sur le dessin, la conception (en architecture, ingénierie, art, etc.) et les comportements ou modèles mentaux du dessinateur dans ce cadre. Par exemple, Masaki SUWA, par une étude publiée dans [Suwa et Tversky, 1997], cherche à cerner les intentions de l'architecte et à comprendre ce qu'il perçoit lorsqu'il utilise le croquis comme support à la création. Nous avons également présenté dans la section 1.2.2 page 10 des travaux et études sur l'utilité du dessin dans les processus de

[(1)] Bien que nos travaux s'orientent essentiellement sur la modélisation 3D, et donc la création de l'enveloppe, nous verrons dans le chapitre 5 page 125 en quels termes notre proposition peut être étendue pour l'aide à la conception d'espaces fonctionnels dans l'esprit d'ESQUISE [Leclercq, 2004].

conception, ainsi que dans le cadre spécifique de l'architecture (section 1.3.2 page 19). Mais il n'a encore jamais été établi, à notre connaissance, de caractérisation de l'activité de dessin par les traits composant ce dessin. Or, c'est de ce lien entre les traits et la tâche dont nous avons besoin pour proposer une interaction adaptée et c'est dans cette optique que s'inscrit donc notre étude.

De l'étude des traits dessinés, données de plus bas niveau, nous avons extrait des caractéristiques spécifiques, des invariants ou des propriétés générales de la phase de dessin.
Cette section présente tout d'abord le protocole expérimental et ensuite les analyses que nous avons effectuées sur les données collectées.

3.3.1 Protocole expérimental

L'étude a été menée sur 26 sujets volontaires. La taille de cet échantillon est certes insuffisante pour obtenir des résultats généraux sur le dessin dans la conception architecturale (un échantillon plus conséquent serait nécessaire). Toutefois, en considérant le trait comme donnée de base et au vu de nos objectifs, la quantité de traits obtenus et analysée fournit des résultats nouveaux et intéressants sur la composition d'un croquis architectural ainsi que des implications sur l'interaction avec un utilisateur.

21 de nos sujets étaient étudiants en architecture à l'École d'Architecture de NANTES. Leur cursus comporte 6 années d'études, le dessin en perspective leur étant enseigné dès la 1ère année. Notre panel d'étudiants était composé de : 2 étudiants en 1ère année, 9 de 2nde année, 2 de 3ème année, 1 de 4ème année et 7 de 5ème année. Les cinq autres sujets étaient des architectes diplômés, doctorants au laboratoire CERMA de l'École d'Architecture de NANTES. Ceux-ci avaient donc mené à bien divers projets et effectués différents stages professionnels durant leur cursus, l'un deux ayant de plus exercé quelques années dans un studio d'architecte avant de commencer un doctorat.

La consigne suivante a été donnée aux participants :

> *« Vous disposez d'un stylo et d'une feuille de papier A3 fixée à un dispositif de capture numérique. Vous devez dessiner un bâtiment vu de l'extérieur, dans une vue en perspective unique. Vous ne devez utiliser aucun document (images ou photographies) et il est préférable de concevoir un nouveau bâtiment plutôt que d'en dessiner un déjà existant. Enfin, il n'est pas nécessaire de dessiner l'environnement autour du projet (arbres, ciel, oiseaux, etc.) ».*

Lors de la présentation de cette consigne et durant les courts entretiens préliminaires au passage de chaque sujet, nous avons seulement expliqué brièvement que cette étude portait sur la façon de dessiner d'un architecte, sans introduire la notion de modélisation 3D et d'interaction homme-machine. En effet, leur exposer le but précis de notre étude aurait modifié leur comportement et leur façon de dessiner si ils s'étaient placés de fait dans un contexte de modélisation 3D plutôt que de dessin. Nous avons éludé les questions des sujets trop « intéressés » en leur expliquant que tout leur serait dévoilé en fin de session, lors de la remise de leur rémunération. Inspiré des étude expérimentales en sciences économiques, la rémunération des sujets d'une étude s'est aussi imposée dans les sciences sociales et humaines. Cela contribue à créer un environnement d'expérimentation favorable en intéressant le sujet. Il est ainsi incité à faire de son mieux, surtout lorsque la rémunération dépend de ses résultats (si l'étude le permet). Dans notre cas, la rémunération[(2)] remise à la fin du travail est devenue le principal

[(2)]Les étudiants sont considérés comme les *sujets idéaux*, ne représentant pas un rapport coût/temps élevé. Pour information, nous offrions à chaque participant un T-shirt de l'École des Mines de Nantes, ainsi qu'un livret conséquent sur l'école et la région Nantaise.

sujet d'intérêt en début de passage de chaque sujet, ce qui nous a permis d'éluder simplement les questions portant sur la finalité de l'étude que nous ne voulions pas dévoiler. Ainsi, ne furent traitées que les questions portant explicitement sur la consigne et sa compréhension (« je ne fais qu'un bâtiment ? », « j'ai droit qu'à une feuille ? », ...) ainsi que celles issues de « manques » dans la consigne (« combien de temps j'ai ? », « c'est commencé, là ? », ...).

Au niveau de la disposition physique, les sujets étaient seuls dans une pièce pendant tout le temps de leur travail. Comme le présente la figure 3.1, ils ne pouvaient pas voir l'écran de l'ordinateur qui enregistrait leurs dessins et leurs actions étaient filmées.

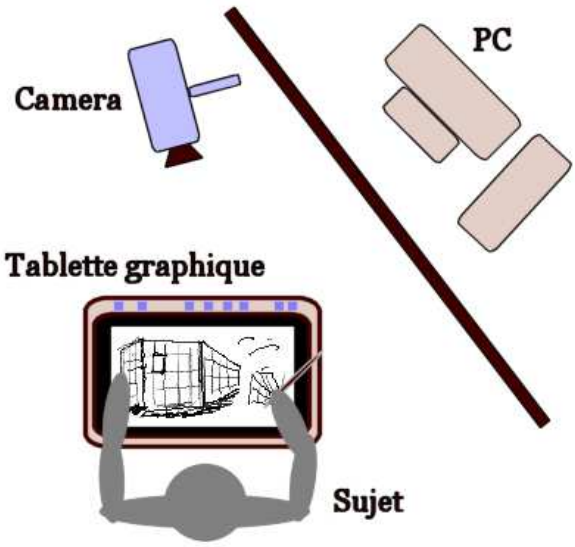


FIGURE 3.1 – Dispositif expérimental.

D'un point de vue technique, nous avons réalisé une application afin de numériser leurs dessins en temps réel avec une tablette graphique connectée à un PC standard. Les données enregistrées étaient structurées de la manière suivante :

- *Les traits*, composés de tous les points compris entre une pression et un relâchement de la pointe du stylet. Pour chaque trait étaient enregistrés ses points, ainsi que les temps de début et de fin du trait.
- *Les points*, composants des traits, pour lesquels étaient enregistrés la position (coordonnées x et y à l'écran), ainsi que la date de création du point dans la chronologie du dessin.

Nous n'avons pas, pour des raisons techniques, enregistré la pression exercée sur le stylet, ni son inclinaison. Les données de la pression auraient sûrement été un apport intéressant à notre étude. Toutefois, les versions papier des dessins permettent d'en avoir un aperçu subjectif. Car en effet, la tablette employée permettait de dessiner avec un stylet à encre sur une feuille de papier normale de manière à ne pas perturber les sujets par l'utilisation de périphériques nouveaux et de ne pas les obliger à regarder l'écran. Dessiner sur une tablette tout en regardant un écran pour avoir un retour visuel demande un certains temps d'adaptation et d'apprentissage. Nos sujets n'étant pas ou peu habitués à un tel environnement, cela aurait introduit un biais conséquent dans nos résultats et nous aurait éloigné de notre but d'analyse de l'activité de dessin au plus proche de son contexte habituel.

Enfin, l'utilisation du stylet à encre ne permettait pas d'utiliser une gomme. Cela ne posait toutefois pas de problème car il n'est pas coutume chez les concepteurs d'effacer des traits (comme nous l'avons vu dans la section 1.3.2 page 19 sur le dessin d'architecte, ceux-ci opèrent par révision et repassent ou modifient des traits, mais n'effacent pas). La figure 3.2 présente le dessin d'un des sujets ayant participé à l'étude.

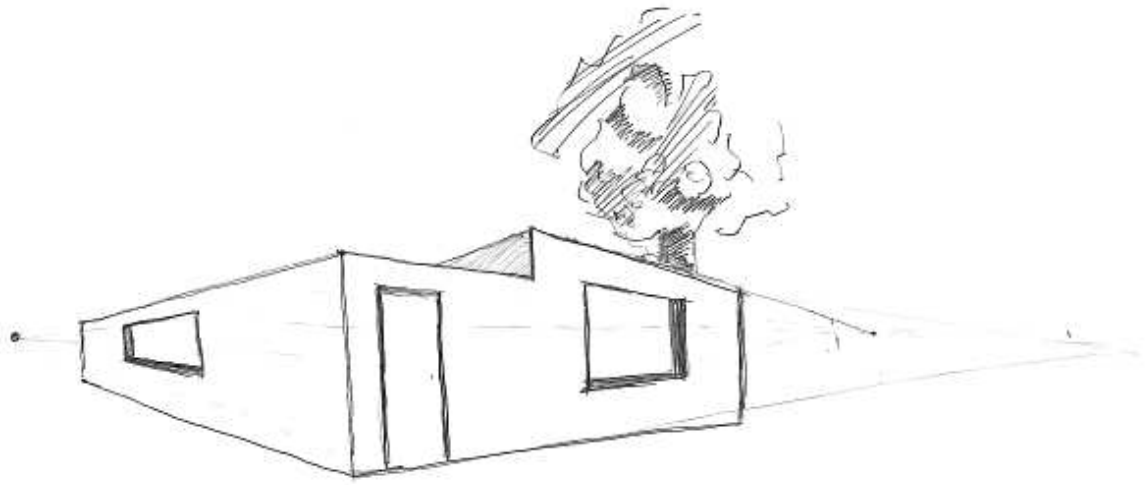


FIGURE 3.2 – Un dessin réalisé au cours de l'étude (d'autres réalisations sont présentées dans l'annexe B page 23).

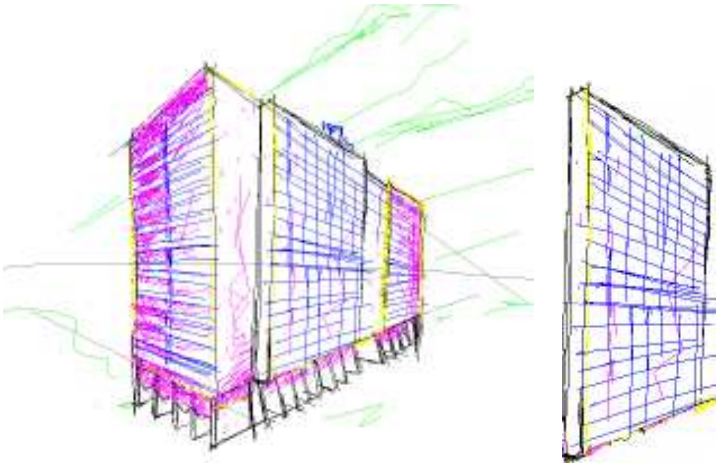
Cette disposition et ce protocole, bien qu'introduisant toujours un biais du fait de la participation à une étude et de l'utilisation d'un outil légèrement différent du traditionnel *papier/crayon*[(3)], a garanti un contexte créatif optimal pour nos sujets. Notons toutefois que ce contexte créatif optimal est celui que nous voulions créer, de manière à pouvoir se détacher au maximum d'éléments extérieurs en plaçant le dessinateur dans un environnement relativement neutre. Ce n'est sûrement pas l'environnement habituel pour un concepteur que de se retrouver à dessiner seul devant une feuille, dans une pièce « blanche », sans bruits, ni repères (dessiner dans son bureau, avec des collègues autour, en écoutant sa musique préférée, dans un parc). Cela nous assurait par contre que nous n'aurions pas à faire intervenir l'environnement dans notre étude, le dessinateur n'ayant que sa feuille pour centre d'attention et son savoir comme source de créativité (pas de « copies » possible d'un bâtiment qu'il voit devant lui, ou d'inspirations diverses). Ces précautions nous on garanti un environnement neutre et identique pour tous les sujets, caution supplémentaire aux résultats de nos analyses.

Une fois les dessins recueillis, il nous a fallu émettre des hypothèses et construire des méthodes et outils d'analyse afin de les infirmer ou les confirmer. Les sections qui suivent présentent ces hypothèses et analyses, en commençant par celles qui nous ont permis de constituer une taxinomie des traits du dessin architectural en perspective.

(3) Les architectes sont toutefois coutumiers de l'utilisation de beaucoup de types de « mines », y compris le premier stylo qui leur tombe sous la main.

3.3.2 Une taxinomie des traits architecturaux

Des classifications et des vocabulaires propres au domaine architectural et à ses dessins ont étés établis. Citons par exemple, l'ouvrage de Jean-Marie PÉROUSE DE MONTCLOS [Pérouse de Montclos, 2000] où sont définis les termes normalisés de l'architecture classique, ou celui de Michel PAULIN [Paulin, 2003] qui présente le vocabulaire de la construction. De la même manière, les dessins architecturaux respectent, avec toutefois certaines variantes, un code graphique pour la représentation des objets usuels ou de certains éléments architecturaux [Lebahar, 1983]. Des groupes de traits d'un dessin peuvent ainsi être reconnus comme des symboles ou des éléments architecturaux. Ces classifications, vocabulaires et conventions donnent donc une sémantique de haut-niveau qu'il est possible d'utiliser pour analyser comment ont été dessinés ces divers éléments, et en extraire des relations.



(a) (b)

FIGURE 3.3 – Sémantique du trait. Dans la figure (a), Les associations de traits forment des éléments à la sémantique bien définie. La figure (b) présente un élément isolé de ce dessin, la saillie de la façade principale. La sémantique des trait est-elle la même pour chaque trait à l'intérieur de ce groupe et/ou dans le dessin global ? (Où, quand et comment.)

Par contre, si l'on souhaite analyser un dessin architectural trait par trait dans le but d'en tirer des conclusions globales sur la phase de dessin, il est nécessaire d'établir une définition et une classification précise de ces traits. Dans la figure 3.3(a), nous présentons un dessin de bâtiment. Le vocabulaire et la sémantique architecturale nous permettent d'en extraire des groupes de traits qui forment les éléments composant ce bâtiment (les piliers du bas du bâtiment, la saillie – un avant-corps – de la façade isolée dans la figure 3.3(b)). Par contre, aucune des classifications actuelles ne nous permet d'analyser ce dessin au niveau du trait, pour étudier par exemple l'ordre chronologique des traits, ou leur regroupement dans des phases temporelles plutôt que dans des éléments physiques, et de fait en extraire des liens.

Nous proposons donc une taxinomie permettant de catégoriser et classifier les traits d'un dessin architectural, dont la construction suit l'hypothèse de départ suivante :

Hypothèse 3.1 *En observant les dessins réalisés par les sujets de notre étude et par des discussions*

informelles avec des architectes ou experts en conception architecturale, nous avons émis l'hypothèse que le dessin architectural en perspective est composé de trois différentes « couches » (voir figure 3.4 page suivante) :

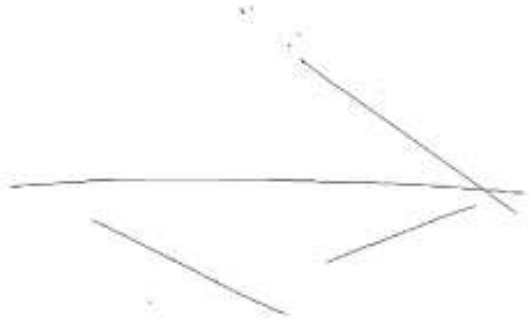
1. *une couche* technique*, composée des traits permettant au dessinateur de se repérer, de construire et de respecter des contraintes techniques et culturelles (voir figure 3.4(a)).*
2. *une couche* d'intérêt*, regroupant les traits du dessin qui composent les objets que veut représenter le dessinateur, le centre d'intérêt du dessin (voir figure 3.4(b)).*
3. *une couche* artistique*, formée par les traits renforçant l'aspect visuel et le style de la couche d'intérêt (voir figure 3.4(c)).*

Il est important de souligner que nous n'affirmons à aucun instant que la démarche de conception créative du dessinateur architecte suive formellement cette structure de couches comme nous la présentons (dans l'ordre et la structure). Cette hypothèse, et donc la notion de couche structurelle, n'est issue que d'une observation *a posteriori* de dessins terminés, sans considération de temps ni d'ordre de construction. Il est toutefois clair que notre intuition première dans cette démarche est l'idée d'une certaine *cohérence* temporelle des traits, cohérence liée à la construction de ces couches au cours de la tâche créative.
À partir de cette décomposition, nous avons donc établi la taxinomie suivante :

Proposition 3.1 *Taxinomie des traits du dessin architectural en perspective :*
- *les traits de* ***construction*** *guident et servent de support au dessin en vue perspective (points et lignes de fuite, horizon, points de repère, etc.).*
- *les traits* ***principaux*** *constituent la forme et le contour de la forme d'intérêt, le bâtiment. Cette classe est divisée en deux sous catégories :*
  - *les traits principaux* ***primaires****, qui sont les premiers dessinés pour créer le contour,*
  - *les traits principaux* ***secondaires****, pour tous les repassages, appuis et « gribouillages »sur les traits principaux primaires.*
- *les traits de* ***détail*** *représentent les caractéristiques du bâtiment (portes, fenêtres, etc.).*
- *les traits de* ***décor****, donnant une apparence réaliste au dessin (matériaux, ombres, etc.).*
- *les traits de* ***style*** *(ou embellissement), pour représenter l'environnement, sans lien avec la forme principale (arbres, nuages, herbes, etc.).*

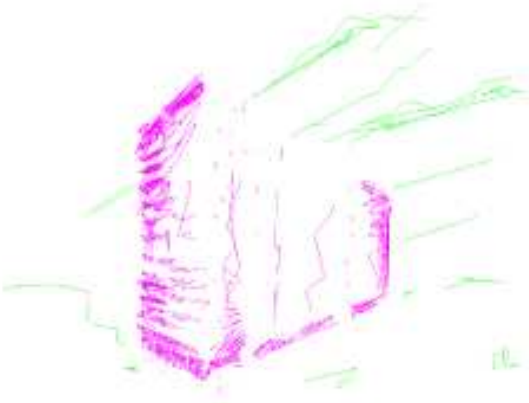
Nous concédons que cette taxinomie est orientée et guidée par la nécessité d'isoler la couche que nous avons précédemment appelée *couche d'intérêt*, pertinente pour l'analyse et le traitement géométrique du dessin. Toutefois, dans un souci d'exhaustivité et pour tenter de couvrir tous les aspects du dessin, nous avons pris en compte dans cette classification tous les traits du dessin. Composée à l'origine des *traits de construction*, des *traits principaux* et des *traits de décor*, les premiers étiquetages de dessins nous ont permis d'observer des différences plus subtiles et donc d'introduire d'autres taxons de granularité plus fine.

Nous avons donc émis l'hypothèse que les dessins récoltés lors de notre étude étaient formés de trois couches structurelles. À partir de ces couches, nous avons définis une taxinomie des traits du dessin architectural en perspective. Dans la suite de cette section, nous allons montrer que des regroupements et des cohérences entre certaines catégories de traits caractérisent ces couches et leur construction dans la chronologie du dessin. Les analyses et résultats qui y sont présentés confirment notre idée première de cohérence structurelle dans le dessin et fournissent les clés pour l'appliquer à un outil de dessin en perspective.



(a) Couche *technique*.

(b) Couche *d'intérêt*.



(c) Couche *artistique*.

FIGURE 3.4 – Couches structurelles d'un dessin d'architecture.

3.3.3 Analyses et résultats

Nous avons développé un outil informatique permettant de « rejouer » chaque dessin chronologiquement, d'en étiqueter les traits selon notre taxinomie et d'appliquer différents outils d'analyse (voir figure 3.5). Des 26 réalisations de nos sujets, seules 24 ont étés retenues pour constituer notre corpus d'étude. En effet, l'un des dessins s'est avéré inexploitable pour des raisons techniques. L'autre dessin a été ignoré car trop éloigné de la consigne : le sujet avait dessiné des parties de bâtiments destructurées et sans rapport les unes avec les autres. Ces 24 dessins forment un corpus de 9858 traits.

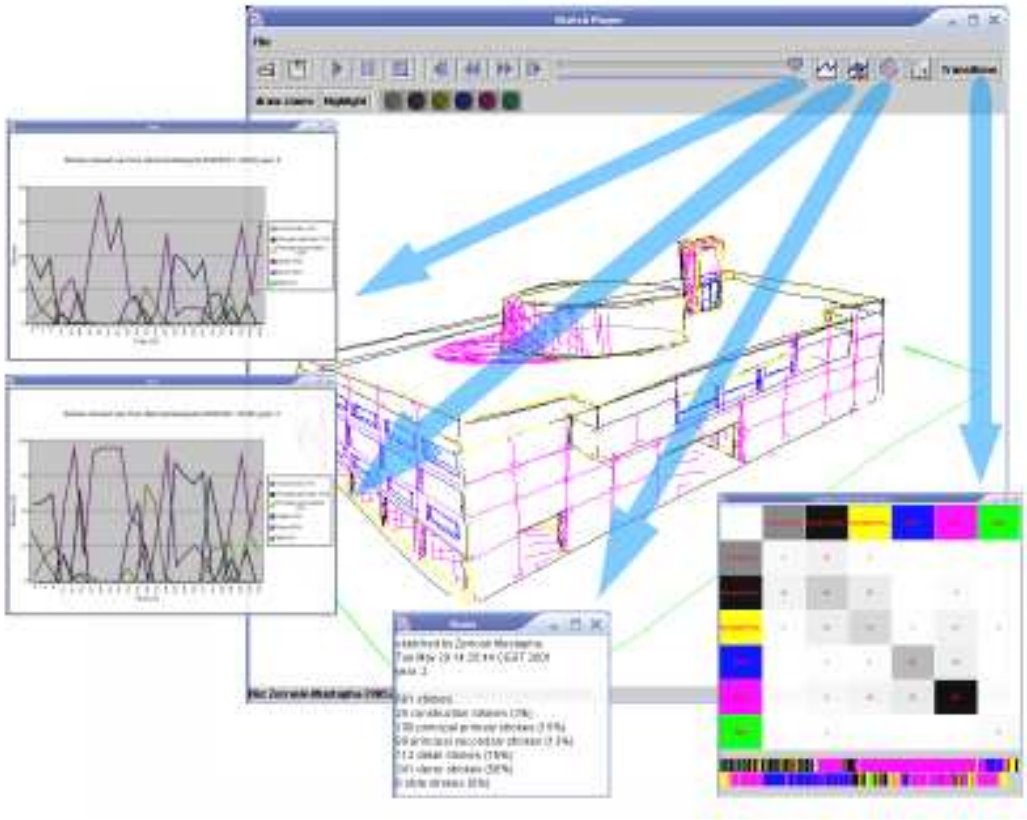


FIGURE 3.5 – Logiciel d'analyse des dessins. Cette application permet de naviguer chronologiquement dans la construction des dessins, d'en étiqueter les traits suivant notre taxinomie, de n'afficher que les traits d'un type et d'appliquer des outils d'analyse.

Dans la suite, nous emploierons souvent le mot *dessin*, sans spécifier *dessin architectural en perspective dans le cadre de notre étude* par souci de simplification. Il est toutefois important de considérer que nos propositions ne visent en aucun cas à généraliser nos résultats à tous types de dessins et aux tâches qui y sont liées.

La première observation est que le temps utilisé pour effectuer le travail demandé varie de 10 à 25 minutes (du moment où le sujet s'installe et reçoit la consigne à celui où il quitte la table), pour un temps de dessin effectif de 2 minutes 30 à 19 minutes 30 (temps passé entre le premier et le dernier trait du dessin). La moyenne de temps de dessin est de 8 minutes, avec un écart type de plus de 4 minutes. Le constat que nous pouvons faire de cette observation est l'irrégularité dans la

durée de l'expérience selon les sujet, sans pouvoir en tirer de conclusions. Il est évident que chacun à besoin de plus ou moins de temps pour accomplir cette tâche, selon la complexité de sa création, ses habitudes et ses aptitudes. Toutefois, nous pouvons tout de même considérer que les sujets étaient (plus ou moins) experts dans le domaine du dessin. Cette expertise leur permet, dans une certaine mesure, de se détacher des difficultés techniques et de ne pas être limités par un apprentissage ou des hésitations imputables à l'outil et son utilisation. Ainsi, quel que soit le sujet, nous considérons que le temps passé à dessiner était entièrement consacré à la tâche et que hésitations ou ralentissements s'inscrivaient dans la démarche créative.

Pour ce qui est de la consigne, celle-ci fut en général respectée. Seul le dernier ordre, celui de ne pas dessiner l'environnement extérieur, fut dans la quasi totalité des cas ignoré. Cela nous a naturellement conduit à prendre en compte ces traits dans notre taxinomie lors de l'étiquetage des traits (traits de style/embellissement). Cette dérogation à la consigne peut paraître anodine mais a pourtant été l'une des bases de notre concept de table à dessin virtuelle et d'interface de dessin libre, comme nous le montrerons dans la section 3.4 page 75.

Pourcentages des différents types de traits

Dans un premier temps, nous avons considéré le pourcentage de chaque type de traits dans tous les dessins du corpus (voir tableau 3.1). Les traits les plus nombreux sont les traits primaires (35%), bien que leur nombre soit de peu supérieur à celui des traits de décor (33%), ce qui va de soi étant donné que la tâche à réaliser était de dessiner un bâtiment. Ces traits (primaires) étant « l'armature » de la réalisation, il semble logique qu'ils soient majoritaires.

Type de trait	*Pourcentage*
Construction	4
Principal $\frac{\textit{Primaire}}{\textit{Secondaire}}$	$\frac{22}{13}$
Détail	18
Décor	33
Style	10

TABLEAU 3.1 – Pourcentage des différents types de traits dans le corpus.

Mais le résultat important porte sur les traits de construction. Nous avions émis l'hypothèse que les dessinateurs, et en particulier les architectes, utilisaient souvent des lignes de fuite pour dessiner en perspective. Cette hypothèse est confortée par le fait que cette technique leur est enseignée dans le cadre de leurs cours de dessin. Un tel usage peut, dans le cadre d'un système informatique de dessin, donner lieu à un outil spécifique. Or, il s'avère qu'il n'y a que 4% de traits de construction sur l'ensemble du corpus. Ce résultat remet en cause notre hypothèse d'une utilisation intensive de traits de construction, dans le cas de notre échantillon[(4)].
Nous avons tout de même jugé important de caractériser ces traits même s'ils sont peu nombreux. Nous avons observé que les traits de construction utilisés sont en majorité de simples points de repère (un seul point ou des traits relativement courts par rapport aux autres), ou parfois des traits plus longs

[(4)]Les sujets de notre étude étant des étudiants pour la plupart, il serait intéressant de réaliser la même étude sur un échantillon d'architectes confirmés afin de voir si les résultats sont identiques.

mais très peu marqués. Nous n'avons pas pu caractériser un niveau de pression sur le stylet caractéristique de ces traits, n'ayant pas enregistré cette donnée. Toutefois, notre observation des dessins papier nous a permis de constater que celui-ci était probablement très significativement inférieur que dans le cas des autres traits. Nous formulons donc la proposition suivante :

Proposition 3.2 *Les traits de construction d'un dessin architectural en perspective peuvent être caractérisés par une longueur relativement courte (points) et/ou une pression relativement faible sur le stylet.*

Distribution temporelle des catégories de traits

Nous avons vu précédemment que la répartition des types de traits sur la totalité du dessin n'était pas d'un grand intérêt, excepté pour les traits de construction. C'est pourquoi nous nous sommes penchés sur leur distribution temporelle pendant le dessin. Le principe est de représenter et d'étudier l'évolution du nombre des traits de chaque type tout au long du temps de création du dessin. Il faut pour cela fixer un temps d'échantillonnage afin de partitionner le temps de dessin en intervalles. Pour chaque intervalle, il suffit alors de comptabiliser le nombre de traits de chaque catégorie qui y débute pour construire une visualisation de l'évolution du nombre de traits (ordonnée) de chaque catégorie (courbes de couleur) en fonction du temps (abscisse), comme le montre la figure 3.6 page suivante.

Plus les intervalles seront petits, plus l'échantillonnage sera précis, mais les *tendances* majeures des courbes restent visibles (des intervalles de 1 à 2 secondes, par exemple). L'observation de ces graphiques individuels (figure 3.6 page suivante), nous a révélé un regroupement des traits principaux en début de dessin. Afin de confirmer et formaliser ce résultat, nous avons appliqué la même analyse sur tout le corpus, dans le but d'établir un invariant dans la distribution temporelle des classes de traits. Pour cela, nous avons *normalisé* les temps relatifs des traits de chaque dessin en pourcentage de temps global du dessin. Ainsi, nous avons pu construire un graphique unique pour tout le corpus de dessins (voir figure 3.7 page 69).

Nous observons sur ce graphique :

1. que même s'il sont peu nombreux sur l'ensemble des traits, les traits de construction sont en nombre conséquent dans les 10 premiers pourcents du temps total de dessin.
2. que les traits principaux primaires sont les plus nombreux dans les 30 premiers pourcents du temps total du dessin.
3. un autre *pic* de traits principaux primaires aux deux tiers du dessin.
4. une augmentation constante du nombre de traits principaux secondaires dans le dernier tiers du dessin.

Les deux premières observations permettent d'identifier une phase *constructive* dans l'activité de dessin en perspective, phase qui situe au début du dessin la création de la plus grosse part des couches précédemment nommées *technique* et *d'intérêt*. La position temporelle de cette phase de traits principaux primaire va dans le sens d'une observation empirique de Pierre LECLERCQ sur une esquisse descriptive [Leclercq, 1996]. Il constatait que la structure fonctionnelle et le partitionnement des espaces d'un appartement dessiné étaient présents après environ 40 pourcents des étapes du dessin. Ce résultat n'a pas été généralisé par d'autres observations, mais nous pouvons tout de même dresser un parallèle avec notre observation du dessin en perspective. La phase constructive que nous situons

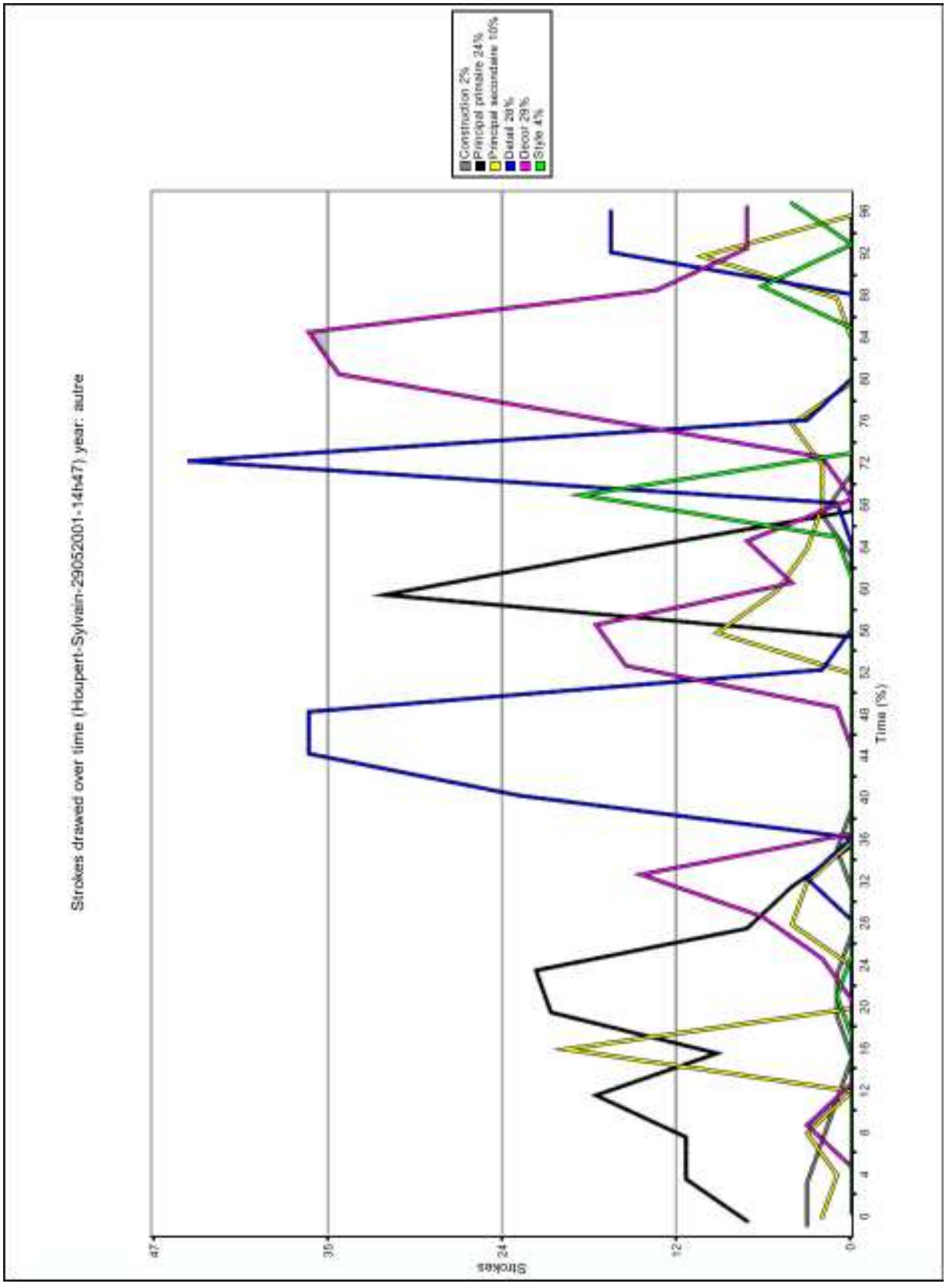


FIGURE 3.6 – Distribution temporelle des classes de traits pour un dessin individuel.

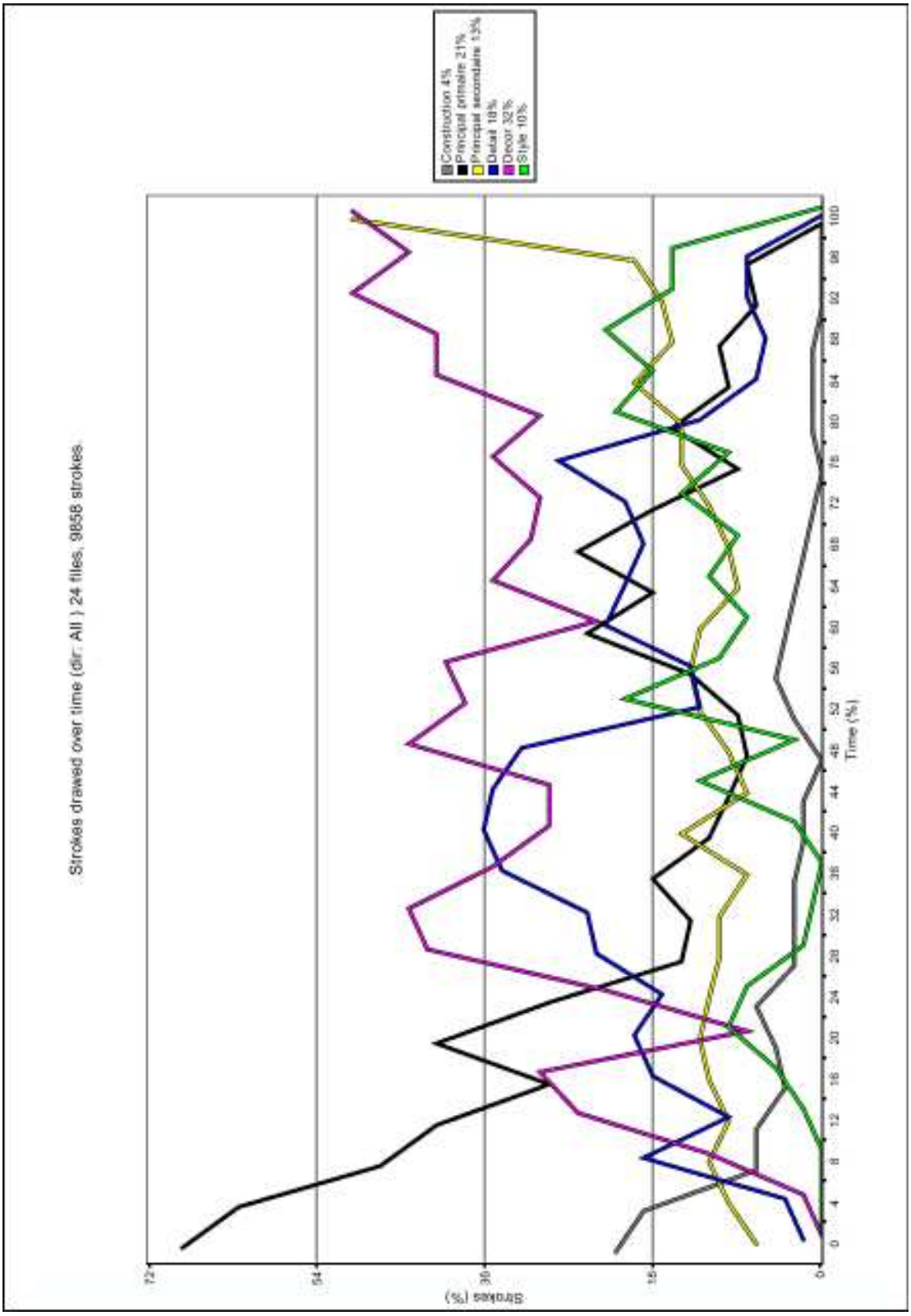


FIGURE 3.7 – Distribution temporelle des classes de traits pour le corpus complet (intervalles d'échantillonnage de 4%). Les courbes des traits de construction et des traits principaux primaires identifient une phase *constructive* dans les 30 premiers pourcents du dessin.

donc dans les 30 premiers pourcents du dessin architectural en perspective, correspond à la création du volume du bâtiment par le dessin de ses contours (les traits primaires de notre taxinomie). Ainsi, comme lors d'une esquisse descriptive (dessin en 2D) un architecte s'attache d'abord à fixer, poser les bases fonctionnelles et spatiales, lors d'un dessin de bâtiment en perspective il va d'abord se concentrer sur la création du volume. Nous pouvons alors parler de *contexte constructif*.

Proposition 3.3 *Une* phase constructive*, correspondant à la création du volume du bâtiment, s'observe dans les 30 premiers pourcents du temps d'un dessin architectural en perspective.*

Nous ne pouvons pas tirer de conclusions aussi franches des deux dernières observations, sur le second pic de traits principaux primaires et l'augmentation du nombre de trait principaux secondaires en fin de dessin. En effet, le second pic de traits principaux primaires est tout de même largement absorbé par la quantité constante des autres types de traits. Toutefois, cela nous permet de considérer le fait que même si la plus grande partie de l'enveloppe du bâtiment dessiné est présente en début de dessin, des *retouches* ou des ajouts sur cette enveloppe sont présents en fin de dessin.

Transitions entre les différentes catégories de traits

Nous avons par la suite étudié les transitions entre les traits du corpus. Pour cela, considérons chaque trait. Nous définissons par *transition* le passage au trait suivant. Tout trait appartenant à une catégorie bien définie, il est intéressant d'étudier le nombre de transitions entre chaque catégorie. Nous avons donc construit différents diagrammes de transitions sous forme de matrices (voir figure 3.8 page suivante). Chaque ligne et colonne de la matrice représente une catégorie de trait, les intersections (cases internes) contiennent le nombre total de transitions entre les deux catégories. Afin d'aider la recherche d'information, plus le nombre de transitions est grand, plus la case est foncée. Nous avons appliqué cette technique de visualisation sur chaque dessin, puis sur le corpus entier. Pour les diagrammes individuels (voir figure 3.8(c)), nous avons aussi examiné l'évolution chronologique des transitions. C'est une ligne de temps sur laquelle chaque trait est représenté par un rectangle de la couleur correspondante à son type dans notre taxinomie (voir bas de la figure 3.8(c)). Cette représentation permet d'évaluer et de comparer la taille des paquets de traits de même type (par la taille des rectangles de même couleur) et d'observer leurs transitions avec les paquets de types différents.

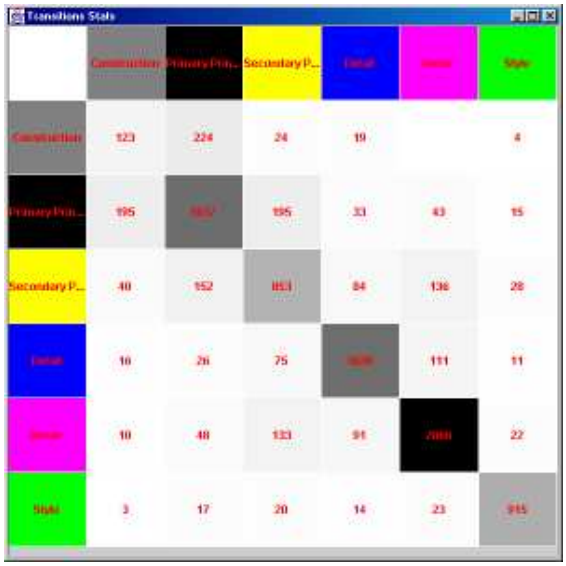
Notre premier constat, sur le corpus et sur les diagrammes individuels, a été le grand nombre de transitions entre traits de même type (sur la diagonale). Nous les appellerons *transitions intra-type*. Nous les avons retirées des diagrammes pour plus de clarté et de contraste sur les transitions entre types différents, les *transitions inter-types* (voir figure 3.8(b)).

Nos observations de ces diagrammes et les déductions que nous en avons tirées sont :

1. La majorité des transitions sont des transitions intra-type.

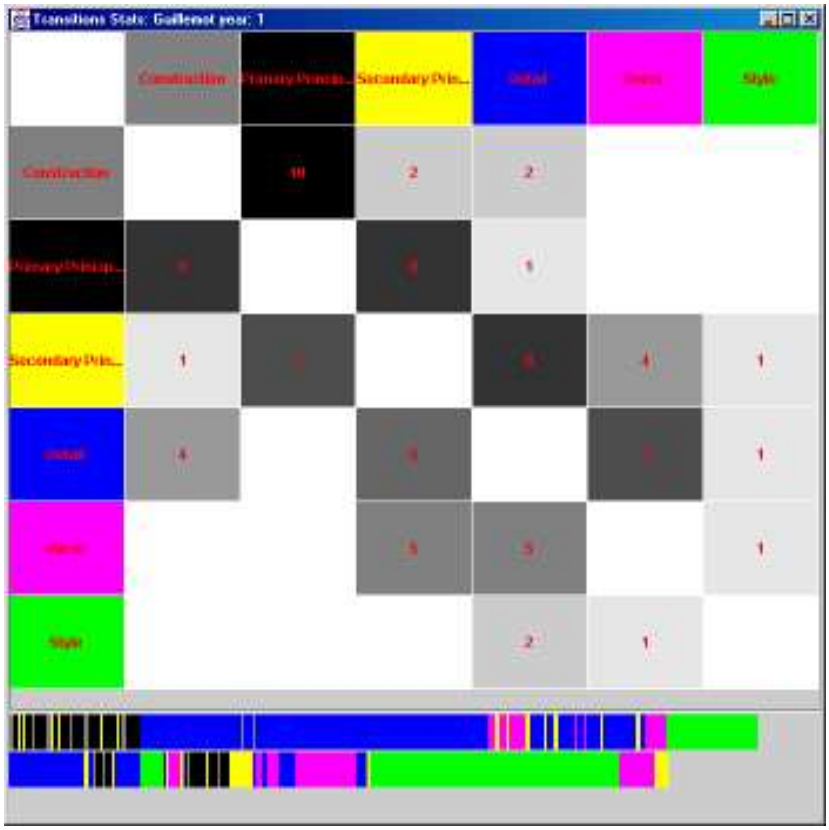
   Proposition 3.4 *Le dessin est formé principalement de* paquets *de traits de même type.*

2. Il y a une symétrie évidente par rapport à la diagonale : les transitions d'un type A à un type B sont donc sensiblement aussi nombreuses que les transitions du type B au type A. Si de plus on observe les évolutions chronologiques individuelles, celles-ci présentent beaucoup de répétitions de paquets de deux ou trois types avant un passage à un type différent des précédents. La figure 3.9 page 72 illustre ces propos. Une alternance de paquets de traits de types A (principaux secondaires en jaune), B (construction en gris) et C (principaux primaires en noir) forme



(a) Corpus complet, avec diagonale.

(b) Corpus complet, sans diagonale.



(c) Cas individuel, avec le diagramme chronologique.

FIGURE 3.8 – Diagrammes de transitions entre types de traits.

un groupe de paquets de traits. Une transition vers un paquet de type D (détail en bleu) marque le début d'un nouveau groupe, formé de paquet de types D et E (décor en magenta).

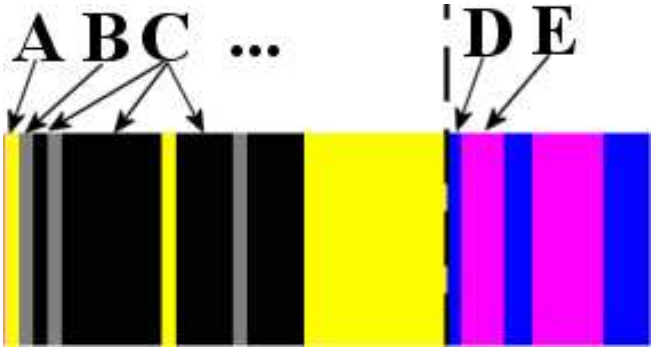


FIGURE 3.9 – Groupes de paquets de traits. Des paquets de types A, B et C se répètent pour former un groupe de paquets de traits. Le passage à un paquet de type D marque le début d'un nouveau groupe formé de paquets de types D et E.

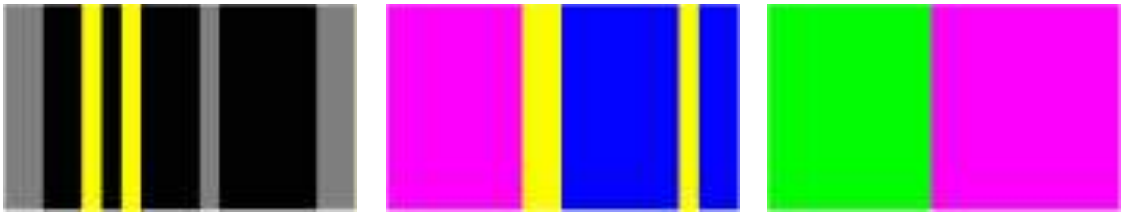
Proposition 3.5 *Le dessin est formé de* groupes *de paquets de traits de deux ou trois types liés qui se répètent dans le groupe.*

3. Les transitions inter-types les plus nombreuses sont :
   (a) Construction ↔ Principal primaire
   (b) Construction ↔ Principal secondaire
   (c) Principal primaire ↔ Principal secondaire
   (d) Décor ↔ Principal secondaire
   (e) Détail ↔ Principal secondaire
   (f) Détail ↔ Décor

   De plus, les diagrammes individuels présentent trois types de groupes, qui se répètent tout au long de la chronologie du dessin.

   Proposition 3.6 *Le dessin est formé de trois types de groupes :*
   (a) ***Groupe constructif*** *: groupe de paquets de traits de construction et de paquets de traits principaux (figure 3.10(a) page suivante).*
   (b) ***Groupe complétion*** *: groupe de paquets de traits principaux secondaires, de paquets de traits de détail et de paquets de traits de décor (figure 3.10(b) page ci-contre).*
   (c) ***Groupe style*** *: groupe de paquets de traits de décor et de paquets de traits de style (figure 3.10(c) page suivante).*

Nous avons donc montré que dans la chronologie du dessin d'architecture en perspective, les traits étaient regroupés en *paquets* de traits d'une même catégorie (proposition 3.4 page 70). Ces paquets sont chronologiquement liés en *groupes* (proposition 3.5). Ces groupes, au nombre de trois, sont formés d'alternances de paquets de deux ou trois catégories différentes (proposition 3.6). Les transitions les plus nombreuses sont des transitions *intra-groupe*, c'est à dire entre traits de même type ou entre traits de paquets du même groupe. Les transitions les moins nombreuses marquent le passage d'un groupe à un autre.
Ces groupes peuvent donc êtres définis comme des *phases* majeures du dessin, dont nous proposons la caractérisation suivante :



(a) Groupe constructif. (b) Groupe complétion. (c) Groupe style.

FIGURE 3.10 – Trois groupes de paquets de traits.

1. ***p**hase constructive*. Cette phase est formée de paquets de traits de constructions et de traits principaux (transitions Construction ↔ Principal primaire et transitions Principal primaire ↔ principal secondaire). La phase constructive située en début de dessin est l'une de ces phases.
2. ***p**hase de complétion et d'amélioration*. Cette phase est composée de traits Principaux secondaires, de détail et de décor, avec les transitions associées.
3. ***p**hase de style*. Cette phase est principalement composée de traits de style et de décor, avec en majorité des transitions Style ↔ Style.

Même si ces phases se trouvent souvent dans cet ordre, nous ne pouvons pas établir un invariant chronologique. Dans la plupart des cas, ces phases se répètent dans un dessin. Et même lorsqu'il n'y a pas de répétition, leur ordre n'est pas garanti. Par contre, nous pouvons affirmer qu'un dessin débute toujours par une phase constructive, suite aux résultats précédents sur la répartition temporelle des catégories de traits (proposition 3.3 page 70).

Temps de pause

Nous n'avons pas pu établir une relation chronologique absolue entre ces phases (un ordre invariant). C'est pourquoi nous nous sommes intéressés à une relation temporelle relative. Pour cela, nous avons considéré *les temps de pause* que prend le dessinateur entre le dessin de chaque trait, le temps ou le sujet ne dessine donc plus. Il y a deux types de pauses :

- les pauses sans changement de phase, dont la moyenne arithmétique est de 0.7 secondes,
- les pauses avec un changement de phase, de moyenne arithmétique 2 secondes.

D'un point de vue statistique, les moyennes arithmétiques seules ne peuvent permettre de déductions sur les durées des temps de pause dans l'un ou l'autre des cas. Toutefois, elles montrent intuitivement que les temps de pauses seraient plus longs lors d'un changement de phase. Nous avons, pour prouver cela, construit et étudié les diagrammes « en boîtes » reproduits sur la figure 3.11 page suivante.

Ces diagrammes nous montrent trois résultats importants :

1. La médiane (notée « med » sur les graphiques) est plus élevée lors d'un changement de phase.
2. L'espace interquartile (longueur de la boîte) est plus grand lors d'un changement de phase, et inclut pratiquement toute la boîte de l'autre cas (pause sans changement).
3. Les valeurs extrêmes des temps de pause sont très proches (non visibles sur la figure).

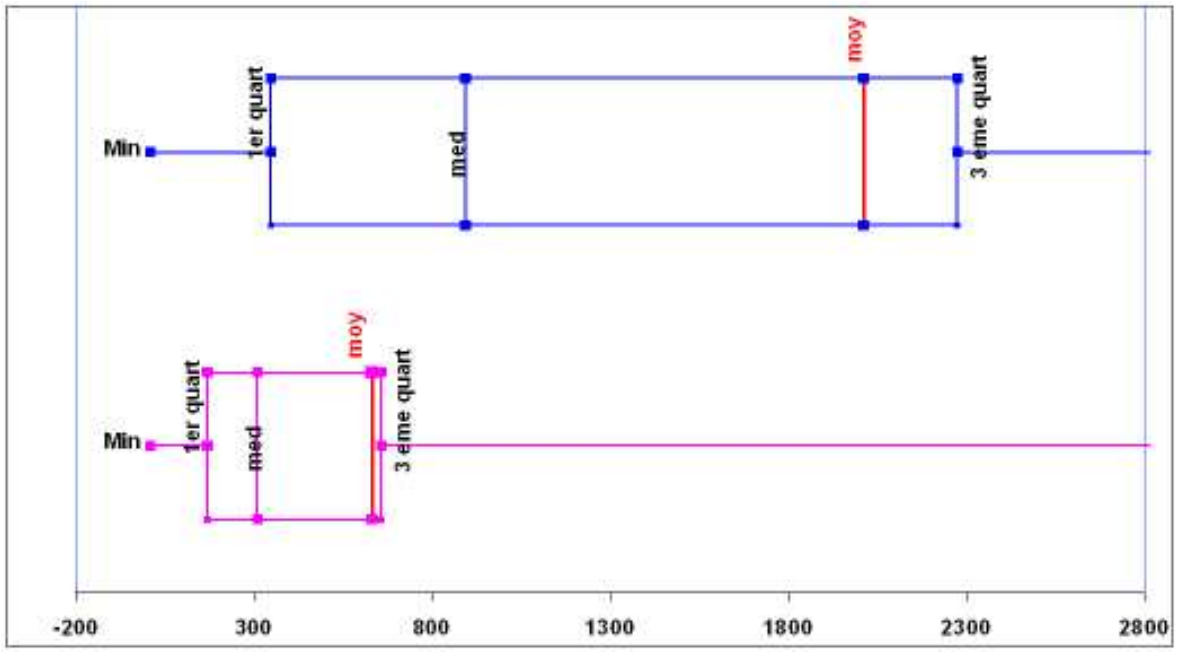


FIGURE 3.11 – Diagrammes *« en boîtes »* des temps de pauses entre traits. En haut : avec changement de phase. En bas : sans changement de phase.

Ces trois propriétés confirment notre intuition première qui est que les temps de pause sont plus longs lors d'un changement de phase.

Proposition 3.7 *Dans l'activité de dessin architectural en perspective, le dessinateur marque des temps de pause plus longs lorsqu'il change de phase de dessin.*

Récapitulatif des résultats

Nous pouvons donc récapituler les résultats important de nos analyses de cette étude :

1. Les dessinateurs de notre étude utilisent peu de « traits de construction » pour guider le dessin en perspective (quelques points de repères ou traits légers, proposition 3.2).
2. La conception de l'enveloppe du bâtiment – traits principaux primaires – s'opère principalement au cours des trente premiers pourcents du temps total (proposition 3.3).
3. Le dessin est composé en plusieurs phases (*constructive*, *complétion* et *style*) qui se répètent le long du dessin, sans ordre particulier excepté qu'il débute par une phase constructive. (proposition 3.6).
4. Les temps de pause sont plus longs lors d'un changement de phase (proposition 3.7).

Dans la section suivante, nous allons, à partir des hypothèses de conception énoncées au début de ce chapitre, proposer les lignes directrices à suivre pour la conception d'un outil de modélisation 3D adapté aux premières phases de conception. Les résultats de cette étude ont étés primordiaux pour conforter la faisabilité des hypothèses de départ, notamment pour ce qui est de l'adaptation de l'interface et surtout de l'interprétation du dessin.

3.4 Lignes directrices pour un outil de « création » 3D

Au vu des constats des chapitres précédents et des hypothèses formulées au début de ce chapitre, nous proposons donc les lignes directrices suivantes :

Proposition 3.8 *Un outil de dessin numérique pour la modélisation 3D se voulant* créatif *doit être fondé sur un paradigme de* dessin libre *:*

1. *Avec des périphériques d'entrée adaptés au dessin à main levée et un environnement et des retours graphiques familiers, permettant de placer naturellement le dessinateur dans un contexte habituel (ou du moins semblable, par une métaphore adaptée). Cet aspect d'intégration de l'outil informatique dans la conception par des dispositifs et un environnement (physique et logiciel) spécifique est à notre avis trop souvent négligé par les travaux actuels qui se focalisent plus sur l'interprétation du dessin.*
2. *Sans contraintes de projection, que l'on retrouve dans les systèmes imposant une vue plane ou une axonométrie, etc.*
3. *Sans contraintes de sémantique du dessin, comme les systèmes qui imposent de ne dessiner que la partie du dessin qui est exploitable par le noyau fonctionnel de l'outil.*
4. *Sans contraintes de qualité de traits, comme le font certains systèmes qui imposent de produire des traits continus et ordonnés (sans retouches, révisions ou gribouillages) afin de conserver l'aspect spéculatif du dessin dans un cadre créatif.*

Cette proposition s'inscrit dans une vision globale du problème de l'intégration de l'outil informatique dans des processus créatifs, depuis l'environnement physique jusqu'à l'interprétation des entrées de l'utilisateur (le dessin). Dès lors, relever toutes les contraintes qu'imposent les systèmes actuels paraît la parade « idéale » et évidente à leurs limitations. Mais chaque contrainte éliminée soulève un problème, soit au niveau du noyau fonctionnel de modélisation 3D (contrainte de vue), soit au niveau de l'interaction et des traitements du dessin (contraintes de sémantique et de qualité du dessin) :

1. Permettre des interactions avec des périphériques d'entrée adaptés conjointement à un environnement graphique avancé implique de rendre leur utilisation la plus transparente possible pour l'utilisateur, mais aussi d'avoir les moyens logiciels de les incorporer dans un tel système.

2. Ne pas imposer de vue particulière, de projection, pour le dessin implique de savoir reconstruire un modèle en trois dimensions à partir de n'importe quel dessin en deux dimensions. Nous verrons que la solution choisie a aussi des répercutions importantes sur l'interaction.

3. Ne pas imposer ce que le dessinateur doit dessiner implique que le système détecte quelles sont les données (les traits) qu'il doit prendre en compte pour effectuer la reconstruction.

4. Ne pas imposer de faire des traits continus ou ne pas utiliser d'outils de dessin spéciaux (dessin vectoriel, dessin guidé, etc.) implique de pouvoir transformer les données du dessin spéculatif à main levé, imprécises ou incomplètes, en données structurées.

À ces problèmes, nous proposons les solutions suivantes, issues de travaux antérieurs ou implications directes des résultats de notre étude.

3.4.1 Périphériques d'entrée et environnements graphiques avancés

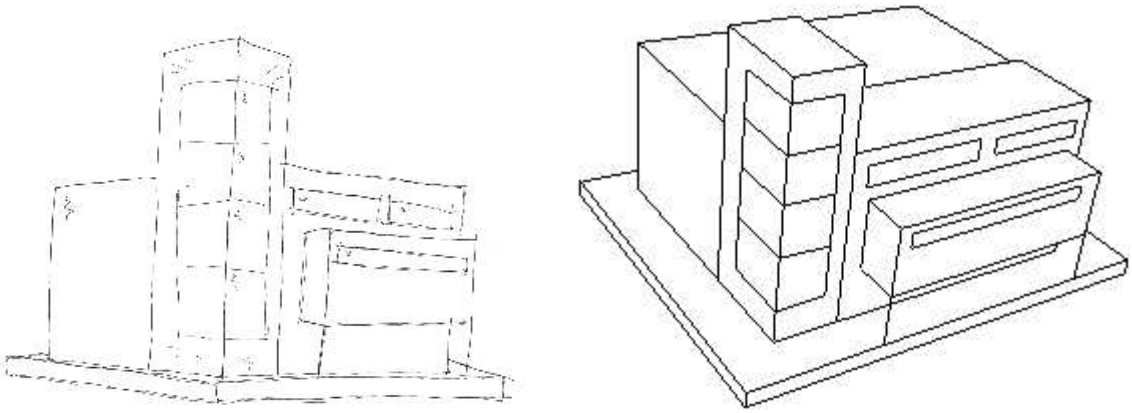
Le premier point de notre proposition implique de permettre l'utilisation pour un tel système de matériels avancés, tels que des tablettes graphiques ou autres périphériques « non standard », afin de servir une métaphore d'instruments de conception. Bien que leur utilisation tende à se démocratiser, il est dommage de constater que de tels dispositifs sont trop souvent sous-exploités par rapport à leur capacités. En effet, si l'on prend l'exemple des tablettes graphiques, elles ne sont supportées par les systèmes informatiques qu'en mode de « compatibilité », ce qui les limite à une *souris avancée*. L'apport est indéniable du point de vue ergonomique, mais encore trop mal exploité au niveau des interfaces. Il faut donc que les logiciels intègrent dans leur conception les fonctionnalités telles que la pression sur le stylet ou son inclinaison. Trop souvent, les logiciels de CAO ou de dessin numérique ne le permettent pas, ou bien au prix d'efforts de configuration hors de portée de l'utilisateur novice.
Il est donc important d'exploiter au maximum les capacités de tels outils, tout en permettant aux utilisateurs de configurer et d'adapter eux-mêmes le système à leurs besoins et habitudes, comme ils utiliseraient un crayon ou un type de papier plutôt qu'un autre. Comme solution à ce problème, et afin de spécifier les interactions avancées que nous décriront dans le chapitre 4 page 83, nous avons utilisé et raffiné la boîte à outils ICON, proposée par Pierre DRAGICEVIC dans [Dragicevic, 2004b].

D'un point de vue visuel il nous semble important de replacer l'utilisateur dans un environnement proche de celui auquel il est habitué, pour ne pas retomber dans les écueils des interfaces dites « standard ». Il faut pour cela reproduire cet environnement (espace de travail optimisé, traits de crayon, grain de papier, etc.) en y introduisant des interactions adaptées aux activités de la tâche, mais sans omettre d'y incorporer les *valeurs ajoutées* que peut proposer un système informatique. Or, c'est encore une fois un problème bien connu des boîtes à outils graphiques « standard » qui par un souci de généralisation ne permettent que très peu d'innovations dans la conception d'interfaces, d'interactions avancées et de composants graphiques adaptés à une tâche précise.
Nous proposons dans la partie II de cette thèse une solution unifiée à ce problème dual de l'association de périphériques d'entrée et d'interactions non-standard avec des graphismes enrichis pour la conception et le prototypage d'interfaces avancées. Nous introduirons pour cela une nouvelle architecture logicielle pour les interfaces post-WIMP, les *graphes combinés*, implémentée dans la boîte à outils MAGGLITE.

3.4.2 Interprétation et élévation de dessins en projection quelconque

Dans le deuxième point, notre proposition énonce qu'une méthode de modélisation 3D plus créative implique la prise en compte du dessin dans une projection quelconque. Cette notion de projection quelconque revient, d'un point de vue mathématique et non perceptif, à la notion de vue en perspective. Ce problème de l'élévation en trois dimensions d'un dessin en perspective a été résolu par Phil Kuzo [Kuzo, 1999] et amélioré par Alexey Sosnov [Sosnov *et al.*, 2002; Sosnov, 2003] dans le cadre du projet GINA. Ils proposent une méthode de résolution de contraintes géométriques pour obtenir un modèle 3D polyédrique à partir d'un dessin 2D en perspective et des propriétés géométriques des éléments de ce dessin. Ces propriétés utilisées par le noyau mathématique sont les contraintes 3D sur les points, droites et plans : coplanarités, parallélismes, orthogonalités et incidences (voir figure 3.12 page ci-contre). Leur solution est algébrique (avec l'algèbre de GRASSMAN-CAYLEY) et utilise la géométrie projective. Cela entraîne plus de robustesse et de performance que les outils précédents basés en général sur des calculs numériques lourds (optimisations linéaires, sujettes à plus d'erreurs ou de cas limites non résolus), tout en permettant des fonctionnalités avancées (correction perspective du dessin

original, reconstruction des parties cachées, démarche itérative).



(a) Un dessin de bâtiment en vue perspective. (b) La reconstruction en 3D.

FIGURE 3.12 – Élévation 3D de dessins 2D en perspective avec GINA. Le dessin de la figure (a) est reconstruit en 3D (b) après identification et spécification des contraintes sur ses éléments géométriques.

Ce noyau mathématique est l'outil approprié à notre approche de modélisation 3D à partir de dessin. Toutefois, la nécessité des propriétés géométriques portant sur le dessin soulève un autre problème : celui de la compatibilité de cette saisie avec l'activité de dessin dans la tâche créative, et de la cohérence d'éventuelles interactions pour la saisie de contraintes. En effet, il serait dommage de se rapprocher d'un mode de saisie aussi naturel que le dessin à main levée, pour d'un autre côté interrompre ce processus par un saisie contraignante, simultanée ou a posteriori, de contraintes géométriques. Il nous semble alors primordial, afin de rester cohérent avec notre approche, que le système détecte le plus possible de ces contraintes de manière automatique en utilisant des méthodes de reconnaissance de formes pour interpréter le dessin. Ainsi, lorsque le dessinateur trace les traits de son dessin, le système les interprète et tente d'en trouver la signification d'un point de vue géométrique. Si par exemple le système a identifié qu'un cube avait été dessiné, il est capable d'en déduire les contraintes correspondantes.

Il reste toutefois à déterminer quelles données doivent être analysées. En effet, si l'utilisateur n'est pas contraint dans ce qu'il doit dessiner, il est nécessaire que le système soit capable de s'y adapter. De plus, les méthodes d'interprétation, en particulier dans notre cas de dessin libre créatif, peuvent nécessiter l'intervention de l'utilisateur dans le cas d'ambiguïtés d'interprétation. La question primordiale est alors quelles données retenir pour l'interprétation et « quand et comment » faire intervenir l'utilisateur dans cette interprétation du dessin ?

3.4.3 Délimitation de la partie d'intérêt du dessin

Un système de dessin libre tel que nous l'avons défini se trouve à la croisée des interfaces de dessin actuelles et du traditionnel papier/crayon. En effet, les systèmes actuels font du dessin une interaction entre le système et l'utilisateur, ne prenant que peu en compte les aspects spéculatif de celui-ci. Les possibilités de dessin avec ces outils sont limitées lors de leur conception, guidée par les fonctionnalité

du système. Par exemple, les interfaces de modélisation 3D actuelles qui utilisent le dessin obligent l'utilisateur à faire des gestes précis ou bien à dessiner uniquement les traits du modèle 3D. Le noyau fonctionnel est capable de traiter certaines données, l'interface de dessin est donc limitée à la saisie de ces données.

Pourtant nous avons vu que le dessin dans un contexte créatif était un mode d'expression et un support à la création, composé de traits qui ne sont pas tous forcement utiles à un noyau fonctionnel d'application même si ils sont important dans le processus créatif (style, embellissement, rythme, ...). Cet aspect est confirmé par le fait que peu des sujets de notre étude ont respecté la consigne de ne pas dessiner l'environnement, et que leurs dessins comportent aussi beaucoup de traits que nous avons qualifiés de *détail* et de *décor*. Ainsi, un système de dessin sur ordinateur se voulant *créatif* se doit d'intégrer cette notion, quelle que soit la finalité de la saisie du dessin. Certains outils permettent une telle démarche mais ne le font qu'a posteriori, dans d'autres modes, et cassent donc le processus créatif en phases séparées.

Notre approche est différente dans le sens où le dessin doit être plus qu'une interaction. Il doit garder son pouvoir d'expression et ses caractères figuratifs propres à la démarche créative. C'est alors le système informatique qui doit s'adapter et déterminer ce qu'il doit traiter. Nous cherchons ainsi à ne pas placer l'utilisateur dans une nouvelle méthodologie, mais à ce que l'outil informatique sache interpréter la sienne[(5)]. De fait, il doit pouvoir isoler la partie d'intérêt du dessin dont son noyau fonctionnel à besoin.

Lors des analyses de notre étude, nous avons identifié trois phases dans le dessin d'architecture en perspective, en particulier la *phase constructive*. Celle-ci est primordiale pour l'interprétation du dessin car elle est source des données les plus pertinentes (traits du bâtiment). La détection de ces phases en temps réel peut donc permettre le *filtrage* du dessin afin de ne transmettre au noyau fonctionnel que les traits qui lui sont utiles : ceux des phases constructives dans notre application à la modélisation 3D.

Mais paradoxalement, c'est aussi durant ces phases constructives, qui sont des plus créatives, que le dessinateur ne doit pas être perturbé par des interrogations du système. Non que nous considérions que les autres phases (complétion et style) ne sont pas créatives et puissent être interrompues, mais elles sont plus éloignées du but premier qui est la construction du volume. Nous rejoignons donc là le problème que nous avions soulevé précédemment : quand et comment faire intervenir l'utilisateur dans l'interprétation du dessin ? La détection des phases du dessin doit aussi permettre au système de s'adapter à la situation courante de l'utilisateur. Par exemple, les propositions d'une interprétation automatique ou des demandes de confirmation pour résoudre des ambiguïtés doivent être effectuées en dehors des phases constructives.

Proposition 3.9 *Un système d'interprétation en temps réel de dessins libres doit détecter les phases du dessin pour :*

1. *Isoler la partie d'intérêt du dessin afin de ne transmettre que les données utiles au noyau fonctionnel.*
2. *Proposer des interactions non intrusives et maîtrisées par l'utilisateur pour ne pas perturber l'activité en cours.*

[(5)]Le mot *outil* prend alors ici tout son sens. Nous ne parlons plus de logiciel mais d'outil informatique, dans le sens ou le système tend à devenir un outil figuratif de l'activité créative, tout en proposant des fonctionnalités supplémentaires (reconstruction 3D) sans en changer l'utilisation. Cette notion est très inspirée de la citation de H. ARENDT en début du chapitre suivant : « L'outil le plus raffiné reste au service de la main qu'il ne peut ni guider ni remplacer ».

Nous associons donc clairement ces phases à des contextes d'interaction. Nous verrons, dans la section 4.4.4 page 114, que les pourcentages et probabilités de transitions entre phases ainsi que les temps de pause entre les phases permettent leur détection en temps réel.

Finalement, permettre à l'utilisateur de dessiner librement à main levée, sans imposer de dessin vectoriel, guidé ou limité à des traits continus implique que le système soit capable de *structurer* et *épurer* le dessin pour en tirer sa structure géométrique.

3.4.4 Structuration et nettoyage du dessin

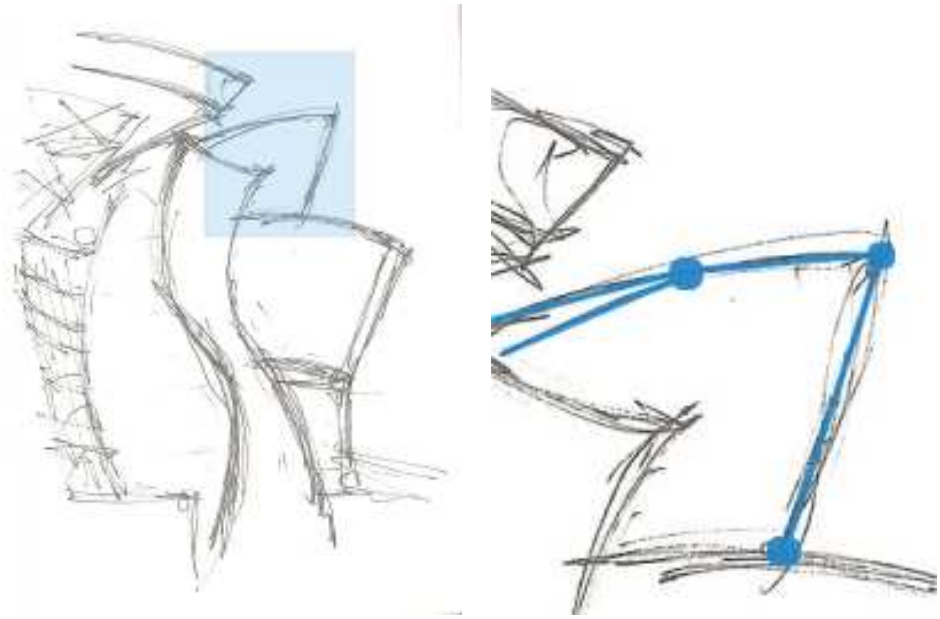
Penchons nous enfin sur le dernier point de la proposition, et pour cela, attachons nous à l'observation et l'interprétation du dessin spéculatif : comment rendre un croquis, la plupart du temps simple à comprendre pour le dessinateur ou un observateur, exploitable par un système informatique ?

Lors de la conception, nous avons vu que le dessinateur utilisait un mode de dessin, de croquis, dit spéculatif. Ainsi, il construit itérativement des espaces ou des volumes qu'il représente par leurs frontières, les traits du dessin. Un trait peut être considéré à ce niveau comme un objet atomique, qui assemblé, combiné à d'autres va produire un ensemble plus structuré avec une sémantique plus ou moins bien définie. Des groupes de traits sont ainsi constitués, puis assemblés, comme autant de révisions et de raffinements dans l'espace des solutions au problème de conception. Ainsi, un trait tracé en début de dessin pourra se voir modifié (repassé, rallongé, etc.) plus tard dans le processus de conception.

Nous avons d'ailleurs vu dans notre étude, avec l'analyse des transitions et nos propositions sur les phases de dessin, que plusieurs phases de *complétion* se retrouvent dans le dessin. Mais le cheminement, du premier au dernier trait au cours des révisions, n'est ni linéaire, ni invariant d'un dessinateur à l'autre. Il s'avère donc que, d'un point de vue géométrique, et quelle qu'en soit la chronologie de tracé, plusieurs traits peuvent composer une même caractéristique géométrique (par exemple un segment) qui a été construite par itérations successives, puis remodifiée ou ajustée, jusqu'à satisfaction. La sémantique en est claire pour le créateur, mais aussi pour les observateurs, dont la culture permet les regroupements sémantiques. Par exemple, même si les différents traits composant l'arête d'un mur ne coïncident pas exactement, ou si les points extrêmes de deux arêtes contiguës ne se rejoignent pas exactement en un même point, l'observateur les unifie implicitement et comprend ce qu'ils représentent. La figure 3.13 page suivante montre cet aspect d'historique des révisions et de relative imprécision d'un dessin (d'un point de vue géométrique).

Si nous transposons maintenant cette démarche dans le contexte de l'informatique, le dessinateur va utiliser des périphériques d'entrée qui, quels qu'ils soient, produisent des séries de points. Il va être indispensable de structurer les points reçus en entités de plus haut niveau, des traits dans un premier temps, pour opérer une première liaison, un dialogue cohérent entre le dessinateur et le système par l'intermédiaire des périphériques d'entrée. Le dessinateur et le système manipulent donc un premier type d'objets commun, le trait, qui en plus d'être à la base de la construction du dessin va aussi être la source des analyses du système.

Ensuite, ses traits doivent être transformés sous une forme exploitable par le noyau fonctionnel, qui lui manipule des points et des segments. Nous avons schématisé ces différentes transformations par la figure 3.14 page 81. Le dessinateur *communique* avec le système par l'intermédiaire de périphériques d'entrée qui produisent des points. Ces points sont regroupés par le système afin de produire une première classe de données communes au dessinateur et au système : les traits. Ceux-ci sont



(a) Croquis complet.

(b) Un détail du croquis et son interprétation géométrique.

FIGURE 3.13 – Imprécisions du dessin à main levée. Un croquis de Franck O. GEHRY, le musée Guggenheim de BILBAO. Nous constatons sur l'agrandissement que plusieurs traits représentent un même segment et que les extrémités des segments concourants doivent être ajustées.

enfin *interprétés* en entités géométriques, *mots* du langage établi entre l'interface du système et son noyau fonctionnel. Le système SVALABARD, que nous présenterons par la suite, regroupe toutes ces fonctionnalités : c'est le *contrôleur de dialogue* établissant la chaîne de communication entre les trois acteurs que sont le dessinateur, les interactions et le noyau fonctionnel.

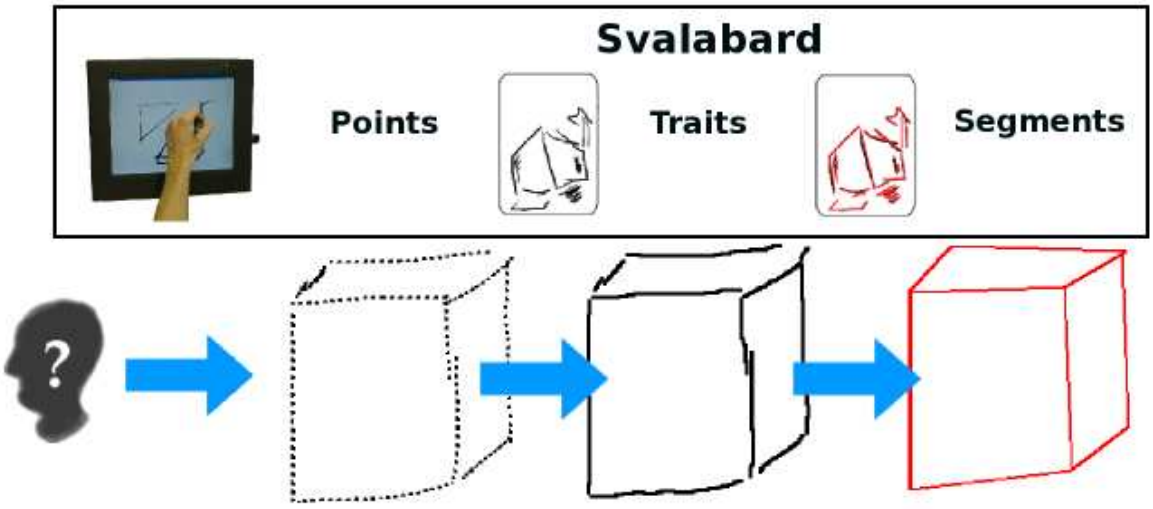


FIGURE 3.14 – De points aux segments. À gauche, les points reçus (dessin de l'utilisateur avec un périphérique d'entrée), transformés en traits au centre puis en segments géométriques à droite.

D'un point de vue technique, cette structuration du dessin passe par une interprétation des traits faisant intervenir plusieurs types d'algorithmes de *segmentation* et *fusion*. Hormis la segmentation de traits, qui est un problème bien connu, les problème majeurs de cette structuration sont dus aux caractéristiques des traits du dessin à main levée que nous avons évoquées précédemment : les révisions successives ou l'assemblage de plusieurs traits et les extrémités disjointes de segments contigus. Ces notions sont normales dans le cheminement du concepteur et implicitement assimilées par lui-même ou un observateur. Elles ne les gênent nullement dans la compréhension du dessin, permettant même des possibilités d'interprétation multiples nécessaires à la démarche créative. Par contre, le système informatique doit *comprendre* et *anticiper* le sens du dessin (au moins à un niveau géométrique). Il faut réduire et interpréter ces imprécisions afin de corriger les éventuels problèmes d'interprétation qu'elles peuvent entraîner au niveau de sa représentation interne. Nous proposons dans la section 4.4 page 110 de tels filtres de *nettoyage* du dessin en temps réel. Ces traitements sont d'ailleurs relativement allégés et soulagés par la détection des phases du dessin car ne leur sont transmis que les traits des phases utiles au noyau fonctionnel.

3.5 Conclusion

Dans ce chapitre, nous avons émis des hypothèses quant à la conception d'un outil informatique de dessin pour la modélisation 3D. Ces hypothèses sont essentiellement axées sur l'importance de la liberté qui doit être laissée au concepteur dans sa démarche et sur l'aspect écologique de l'interface du système.

Nous avons mené une étude sur le dessin d'architecture en perspective dans le cadre d'une tâche créative afin d'en extraire des invariants au niveau du trait. Il en ressort, outre la taxinomie des traits que nous avons introduite pour effectuer nos analyses, une décomposition en phases de l'activité. Bien que la chronologie de ces phases ne soit pas invariante, il s'avère qu'elles se caractérisent par des temps de pause significatifs et des probabilités de transition différentes. Nous avons aussi, malgré leur petit nombre, caractérisé les traits de construction pouvant éventuellement être dessinés pour aider au dessin en perspective.

Dès lors, nous avons été en mesure de proposer à partir de travaux précédents et des résultats de notre étude, des lignes directrices pour la conception d'un outil de modélisation 3D « créatif » basé sur le dessin en perspective libre. Le détail de la mise en pratique de ces propositions fait l'objet du chapitre suivant : nous y exposons la réalisation de SVALABARD, une *table à dessin virtuelle* pour la modélisation 3D.

Pertinence de l'étude et de ses résultats

Il convient tout de même de se poser la question de la pertinence d'une telle étude et de ses résultats. Dans un premier temps, le nombre de sujets évalués et leur qualité (étudiants, doctorants et architectes) sont-ils significatifs ? Est-ce qu'un nombre plus élevé de participants aurait produit les mêmes résultats et les aurait rendus plus pertinents ?

La taxinomie que nous avons proposée est basée sur une analyse et une connaissance approfondie du dessin au trait, relayée par des experts du domaine et basée sur une étude. Cette démarche lui confère une bonne qualité de couverture en terme de description d'un croquis à partir du trait. Dans ce sens, notre étude n'est pas une étude *ergonomique* de l'outil, mais une étude *ethnographique*, le trait tenant lieu d'*objet ethnographique de la conception créative par le dessin*[6]. Dans cette première optique, la qualité des sujets importait peu tant que ceux-ci étaient familiers de la tâche et de ses activités.

Notre expérience, et l'utilisation que nous avons faite de ces résultats, montre que l'intérêt principal de ce travail n'est pas d'obtenir des données numériques à prendre pour argent comptant. Bien sur, les analyses ont permis de formaliser une partie des caractéristiques de l'activité étudiée. Mais elles viennent surtout confirmer l'observation dans le cadre général et c'est pour cette qualité de *prise de connaissance et de contact* avec le domaine et la tâche qu'une telle étude est primordiale. Dans cette seconde optique, le nombre des sujets impliqués est suffisant.

Enfin, les chiffres de cette analyse sont un point de départ pour établir des heuristiques lors de la réalisation d'outils informatiques adaptés à cette activité dans un contexte créatif (analyse de traits, détection des phases, etc.). Ce n'est toutefois pas le nombre de sujets qu'il conviendrait d'augmenter par la suite, mais plutôt la capacité du système qui en découle à adapter ces pourcentages en fonction du profil spécifique de l'utilisateur.

[6] En ethnologie, l'objet ethnographique est défini comme « objet qui se révèle un outil d'interprétation indispensable à partir duquel il est possible d'expliquer, d'illustrer et de reconstituer une partie de la vie de l'homme sur terre ». L'emploi de ce terme pour le trait dans notre cadre d'étude caractérise l'importance que nous lui avons donné dans notre travail et l'orientation de notre étude.

Chapitre 4

Une table à dessin virtuelle : SVALABARD

« L'outil le plus raffiné reste au service de la main qu'il ne peut ni guider ni remplacer »

H. ARENDT.

Sommaire

4.1 Introduction

La proposition 3.8 que nous avons introduite dans le chapitre précédent définit les quatre lignes directrices de la réalisation d'un système de dessin pour la modélisation 3D plus proche des habitudes et aptitudes des utilisateurs créatifs :

1. proposer un ensemble complet et cohérent de périphériques d'entrée et d'interactions, dans un environnement graphique familier.
2. ne pas imposer de contraintes de projection.
3. ne pas imposer de sémantique du dessin.
4. ne pas imposer de contraintes de qualité de traits.

Nous avons aussi, dans ce même chapitre, cerné les problèmes que pouvaient entraîner ces lignes directrices, ainsi que les solutions que nous proposons pour les résoudre. Afin d'en démontrer la pertinence et la faisabilité, nous les avons appliquées à la conception du système SVALABARD.

(a) Au début du XX^{ème} siècle. (b) De nos jours.

FIGURE 4.1 – Tables à dessin, l'environnement adapté au dessin. Bien que proposant des outils plus orientés vers le dessin technique que le croquis, les tables à dessin sont toutefois un espace habituel de travail pour tous types de dessinateurs créatifs (designers, architectes, animateurs, etc.).

Ce système permet la saisie de dessin libre, donc à main levée, par l'intermédiaire d'outils interactifs de dessin et de manipulation directe : crayon, calques, déplacement de calques, gestes de commandes. Ces outils sont explicitement associés à des périphériques d'entrée (tablette écran, tablettes graphiques, périphériques isométriques) et leurs actions et comportements sur le système sont inspirés des outils réels qu'ils représentent. Les principes de conception que nous avons suivis reposent sur le modèle de l'interaction instrumentale [Beaudouin-Lafon, 2000], où toute action sur un objet d'intérêt de l'interface passe par l'utilisation d'un « instrument ». Notre approche pousse même

ce modèle à son extrême, les instruments étant des associations directes « périphérique d'entrée–outil de l'interface ». Ainsi, SVALABARD intègre une métaphore de **table à dessin** (voir figures 4.1 page précédente et 4.2), dans laquelle le dessinateur peut retrouver un environnement proche d'un environnement de dessin créatif et de ses habitudes (papier, crayons, calques), obéissant aux mêmes règles générales de manipulation (un outil physique pour une activité ou tâche).

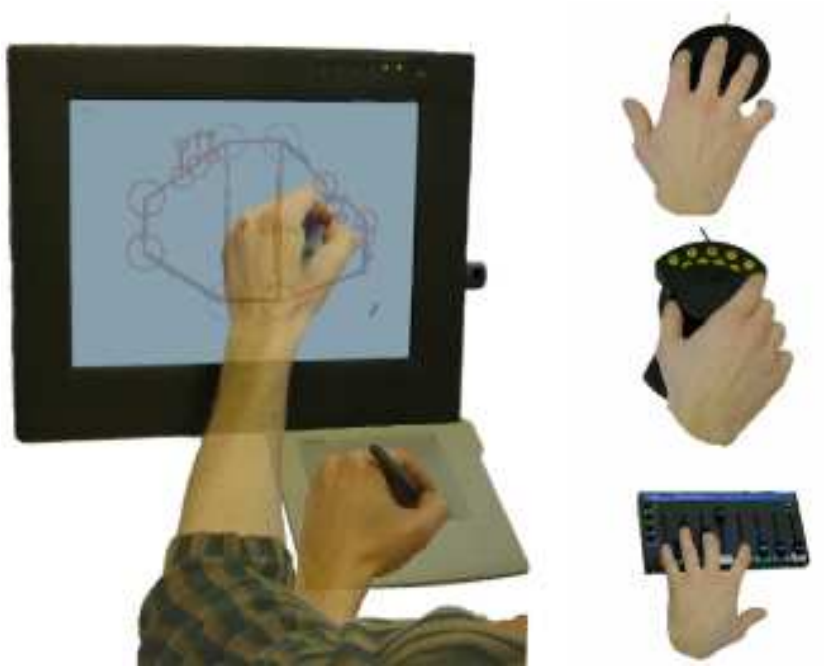


FIGURE 4.2 – Une table à dessin virtuelle. Un choix de moyens d'interagir avec le système.

Pour les traitements du dessin, SVALABARD intègre une *cascade* de filtres et interpréteurs qui permettent, à partir des traits tracés, d'obtenir en temps réel un modèle géométrique 2D. Ce modèle, que l'on appellera aussi parfois *vectoriel* de par l'analogie avec les outils de dessin du même nom, est composé de points et de segments, les données essentielles pour la construction d'une maquette 3D polyédrique par le noyau mathématique de GINA. Nous insisterons en particulier sur deux traitements originaux, permettant la détection des phases du dessin d'architecture que nous avons identifiées dans le chapitre précédent, et la réduction de l'imprécision du dessin par fusion de points et fusion de segments.

Ce chapitre débute par une description de l'interface et des interactions de SVALABARD. Les sections 4.2 page suivante et 4.3 page 91 détaillent la métaphore de table à dessin virtuelle, et introduisent le paradigme des *feuilles d'interaction*, qui représente les différents modes et outils de l'interface. Par la suite, les sections 4.4 page 110 et 4.4.7 page 120 portent sur les aspects techniques des filtres et interpréteurs de dessin que nous avons inclus dans ce système. Enfin, la conclusion dresse un premier bilan de notre approche. Une discussion plus prospective sur les travaux restant à accomplir, les points forts, les limites et les évolutions futures du système fera l'objet du chapitre 5 page 125.

Une partie des travaux exposés dans ce chapitre a fait l'objet d'une communication [Huot *et al.*, 2004c].

4.2 Le paradigme des « Feuilles d'interaction »

L'interface de SVALABARD est une interface post-WIMP [van Dam, 1997]. Elle n'est donc pas basée sur les composants (*widgets*) rencontrés habituellement dans les interfaces (menus, boutons, barres d'outils...). Les interfaces WIMP, et les composants standards qu'elles proposent, ont fait la preuve de leur utilité dans des domaines variés, mais elles ont aussi montrés leurs limites dans le cadre d'activités telles que le dessin, n'ayant tout simplement pas été conçues pour. La première de nos lignes directrices, portant notamment sur l'aspect familier de l'environnement graphique, nous a conduit à utiliser le principe d'une *feuille de papier virtuelle*, support visuel de notre métaphore de table à dessin.

SVALABARD propose trois modes d'interaction distincts : **dessin**, **interprétation du dessin** et **visualisation/manipulation 3D**. Dans beaucoup de systèmes existants, les modes de l'interaction sont présentés sous la forme de vues multiples ou de menus et barres d'outils contextuelles. De plus, les changements de modes sont souvent explicites, l'utilisateur devant spécifier par une commande son intention de passer dans un autre mode.

Pour supporter la métaphore de la table à dessin et de la feuille virtuelle, notre interface propose des changements de mode implicites (déterminés par les phases du dessin, la compréhension et l'analyse du dessin, l'activité courante du dessinateur) et des vues superposées (par transparence ou masquage). Chaque mode, et donc chaque activité lui étant associée, est représenté par une **feuille d'interaction**. Dans l'esprit du modèle multi-couches ([Fekete et Beaudouin-Lafon, 1996; Fekete, 1996]), ces feuilles deviennent alors les supports d'outils et d'interactions adaptées elles aussi au mode et à l'activité correspondante. Outre son aspect pratique pour la gestion de l'interaction et les possibilités de représentation graphique qu'elle offre (superposition des différentes feuilles, vue des feuilles du dessous par transparence, masquage de feuilles), cette organisation nous a en plus permis d'évoquer les trois modes du dessin d'architecte que l'on a décrit dans la section 1.3.2 page 19 :

1. le *dessin spéculatif*, représenté par la **feuille de dessin**,
2. le *dessin prescriptif*, représenté par la **feuille augmentée**,
3. le *dessin descriptif*, représenté par la **feuille 3D**.

Chacune de ces feuille apparaît et s'active selon les actions de l'utilisateur. Ainsi, les comportements du système sont guidés par la tâche et non l'inverse.

Toutefois, l'utilisateur a toujours la possibilité de configurer lui même les comportements du système afin de l'adapter à ses propres goûts et méthodes de travail. Il est important que l'Homme puisse « façonner l'outil », l'adapter à ses méthodes plutôt que le contraire. Cette approche est primordiale dans le domaine d'application de nos travaux où il fut trop souvent de mise de vouloir imposer les seules méthodes qu'il était techniquement possible de réaliser. Ainsi, notre système permet de redéfinir interactivement, de manière graphique, les actions qui vont déclencher les changements de modes implicites (temps d'inactivité, phases du dessin par exemple), ou même les rendre explicites par des commandes et des interactions choisies (commande gestuelle ou vocale, touche du clavier ou boutons d'un périphérique d'entrée par exemple). Notons aussi, comme nous le verrons dans la section 4.3 page 91, que les interactions et outils proposés peuvent entièrement être adaptés et configurés (en terme de périphériques d'entrée et d'interactions).

Proposition 4.1 *Au vu de ces notions, nous pouvons qualifier* SVALABARD *d'interface :*

– ***Contextuelle**, car proposant des outils et visualisations (modes) différentes et appropriées à l'activité courante de l'utilisateur,*

– ***Adaptable**, car changeant de mode de manière implicite selon les actions courantes de l'utilisateur,*
– ***Configurable**, car permettant à l'utilisateur d'en redéfinir les comportements.*

Détaillons maintenant ces modes, chacun étant représenté par une vue, une *feuille*.

4.2.1 La feuille de dessin

Il s'agit d'une feuille de papier virtuelle sur laquelle le concepteur dessine simplement (voir figure 4.3). Cette feuille représente le mode principal de SVALABARD, le mode de dessin spéculatif. Son aspect graphique évoque une feuille de papier, renforçant l'immersion du dessinateur dans un milieux familier. Plusieurs textures de papier sont proposées, paramétrables en couleur. Mais il est aussi possible d'utiliser un fond personnalisé sous forme d'image. La feuille de dessin devient ainsi plus qu'un support, mais un document à part entière, permettant de travailler à partir d'existant (simple support à la création, reconstruction de bâtiment en 3D dans l'esprit de nos travaux précédents [Huot, 2000; Huot et Colin, 2001] ou insertion dans l'existant). Dans la même optique de similarité avec l'environnement habituel du dessinateur, les traits dessinés sur cette feuille ont un aspect *crayonné*.

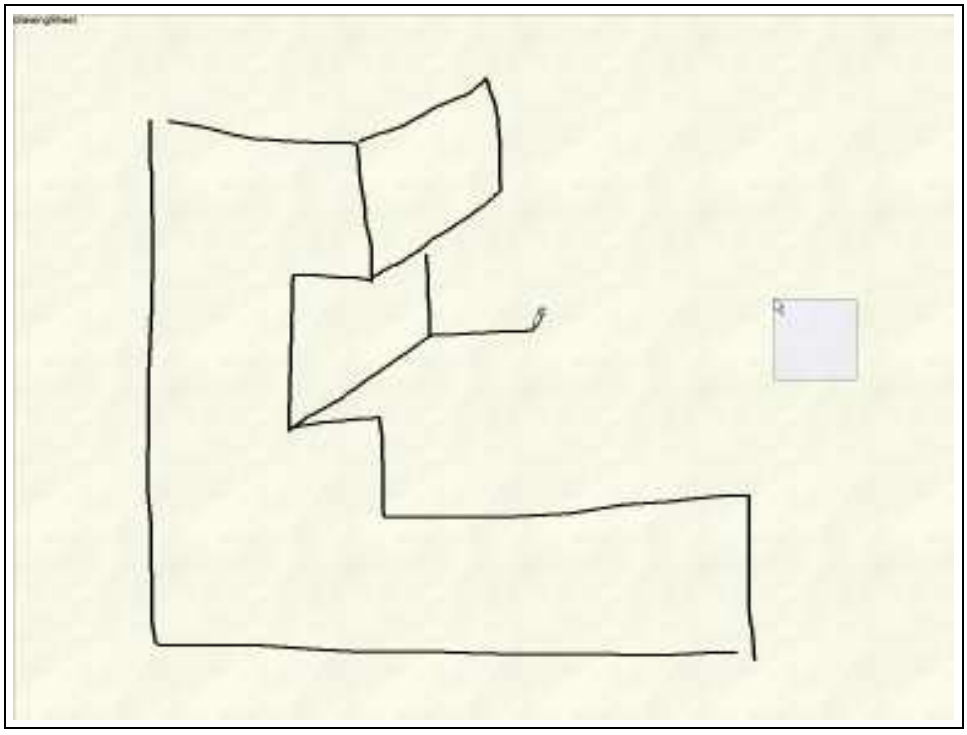


FIGURE 4.3 – Feuille de dessin.

Ce mode de dessin à main levée ne souffre d'aucune contraintes, ce qui est une évolution majeure par rapport aux systèmes de dessin pour la modélisation 3D actuels. Aucune vue (axonométrie, plane, etc.) ou technique particulière (dessin vectoriel, gestes, etc.) n'est imposée, conformément à nos lignes directrices. Dés lors, afin que le système puisse interpréter le dessin, cette feuille est connectée aux filtres de traitement décrits dans la section 4.4 page 110. Ces filtres sous-jacents, qui s'exécutent en

tâche de fond, permettent de structurer le dessin afin de le rendre exploitable pour le noyau fonctionnel de l'application.

Pour parfaire la métaphore de la table à dessin, et permettre à l'utilisateur d'exprimer ses idées comme il le ferait avec ses techniques habituelles, nous avons aussi inclus des **calques de dessin**. Ils peuvent être ajoutés, couvrant alors toute la surface disponible, et ensuite manipulés (dessin, masquage, suppression, reordonnancement, etc.). Ces calques n'ont pas, dans notre proposition, la même fonction de décomposition modulaire que dans les logiciels habituels. En effet, les calques ne servent pas seulement à organiser le dessin en modules distincts. Nous avons vu que le calque était avant tout pour l'architecte une *mémoire de la conception* et un support de la révision, de la modification. C'est dans cette optique que nous avons inclus des calques virtuels sur la feuille de dessin de SVALABARD. Ils permettent de dessiner différentes parties du dessin sur des supports différents, afin de proposer plusieurs possibilités, ou de structurer le travail. Ainsi, ils ont l'utilité que leur donne le dessinateur : décomposition ou révision.

Enfin, la détection des phases de dessin que nous présenterons dans la section 4.4.4 page 114, permet de ne pas limiter l'utilisateur au dessin de la forme à modéliser. Il peut, sur cette même feuille, embellir son dessin, dessiner l'environnement, sans avoir à changer explicitement de mode de dessin, ce qui selon nous correspond mieux à une démarche créative et artistique (contrairement aux systèmes existants qui imposent un ordre chronologique ou un changement de mode explicite).

4.2.2 La feuille augmentée

La vue de SVALABARD que nous appelons feuille augmentée est l'affichage des « valeurs ajoutées » que propose le système par rapport à une feuille de papier réelle. Cette vue en surimpression à la feuille de dessin sert tout d'abord à présenter le résultat des filtres et traitements du dessin. Cela permet à l'utilisateur, si il le souhaite, de visualiser les interprétations et choix du système pour éventuellement les valider, corriger et manipuler si besoin est (voir figure 4.4 page suivante). De plus, même si nous nous inscrivons dans une démarche « d'interface absente » proche des travaux de Pierre LECLERCQ pour le dessin en vue plane [Leclercq et Juchmes, 2002], nous sommes plus modérés quant à l'occultation pour l'utilisateur des résultats des traitements que réalise le système sur le dessin. En effet, l'utilisation d'un système informatique nécessite évidemment de connaître la finalité des actions qui vont êtres exécutées par le système en réponse aux actions effectuées par l'utilisateur [Bastien et Scapin, 1993]. Il est aussi nécessaire, et en particulier lors d'une tâche longue et complexe, d'avoir une connaissance des actions intermédiaires du système. Cette connaissance, ou représentation du fonctionnement du système, permet à l'utilisateur de maîtriser les actions possibles sur le système. Ainsi, représenter les étapes intermédiaires qui mènent au résultat final de la tâche, ou à un échec, évite à l'utilisateur d'élaborer sa propre représentation du fonctionnement du système (souvent erronée) et lui permet de mieux se l'approprier et le contrôler. Il lui est alors possible d'interpréter et de corriger ses erreurs (ou celles du système le cas échéant), mais surtout il ne perd pas en efficacité d'utilisation. Nous avons choisi de présenter les résultats des traitements « intermédiaires » du système (avant le rendu 3D) lorsque l'utilisateur ne dessine plus depuis un certain temps. La feuille augmentée apparaît alors en surimpression à la feuille de dessin et disparaît automatiquement lorsque le dessin reprend. Cette apparition contrôlée à un moment opportun permet de ne pas interrompre ou perturber l'activité de dessin. Le dessinateur peut alors observer comment son dessin a été interprété.

Pour ce qui est du mode de représentation de ces résultats, la feuille augmentée présente simplement l'interprétation géométrique du dessin à main levée sous la forme de points et de segments.

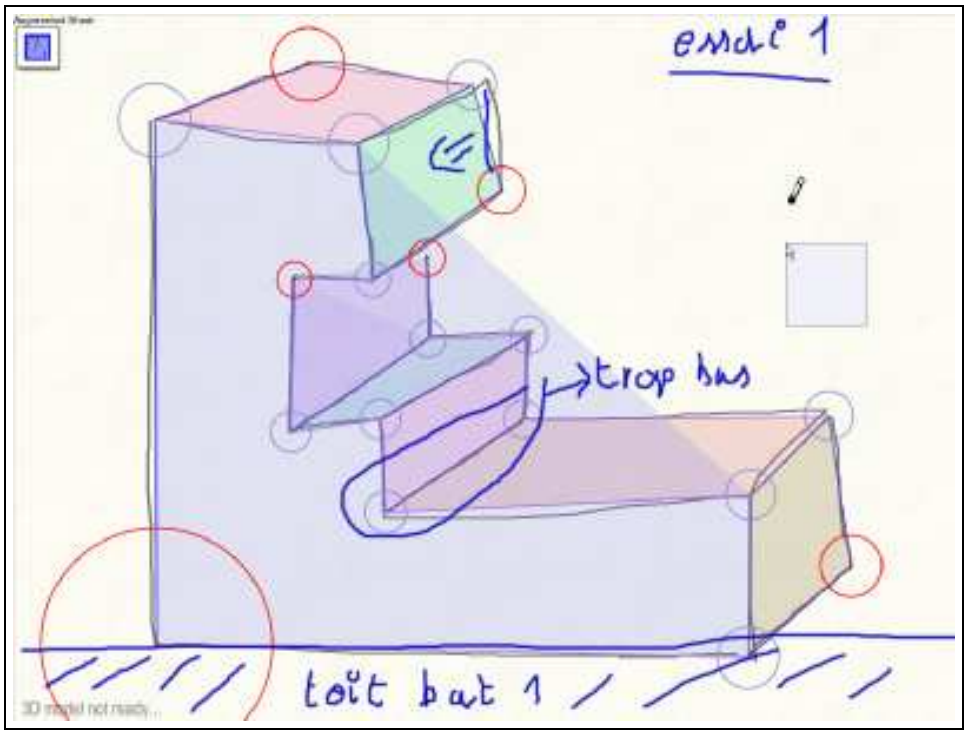


FIGURE 4.4 – Feuille augmentée.

Cette interprétation apparaît donc en surimpression au dessin original, ce qui permet une meilleure visualisation des relations entre les traitements et les traits du dessin original qu'une vue multiple séparée. Cela assure une bonne compatibilité cognitive (capacité de l'utilisateur à interpréter les données relatives aux concepts et à leurs représentations par le système). Si il s'avère que des traitements d'interprétation du dessin sont encore en cours lors de l'apparition de la feuille augmentée, leur affichage est actualisé en temps réel, ce qui renforce la perception des actions du système.

Les autres informations visibles sur la feuille augmentée sont les propriétés géométriques des éléments de l'interprétation du dessin : parallélisme, coplanarités et orthogonalité. Les faisceau de segments parallèles sont tracés d'une même couleur. Les points coplanaires forment des surfaces colorées. Nous avons envisagé plusieurs représentations graphiques pour les propriétés d'orthogonalité, mais aucune ne s'est avérée concluante jusqu'alors (symboles particuliers sur les segments, dessins des angles, etc.). Ces propriétés sont issues de saisies de l'utilisateur, comme nous le présentons dans la section 4.3.2 page 98. À terme, elles devront être en partie détectées automatiquement (nous discutons ces possibilités dans la section 4.4.7 page 120 et le chapitre 5 page 125).

Mais cette représentation de la structure géométrique du dessin sur la feuille augmentée lui donne aussi un sens habituel pour le concepteur : celle de dessin prescriptif. Certes, ce n'est pas un plan de construction métré et précis comme permettent de le faire les outils avancés de CAO. Toutefois, notre approche pourrait être améliorée dans ce sens. C'est dans cette optique que nous avons introduit la possibilité d'annoter le dessin sur cette feuille augmentée, de tracer des traits non pris en compte par le système dans un but d'explication et de prise de notes.

4.2.3 La feuille 3D

La troisième vue de SVALABARD est la feuille 3D. Ce mode permet de visualiser et manipuler le résultat de la reconstruction en trois dimensions du dessin réalisé sur la feuille de dessin, associé aux propriétés géométriques visualisées et saisie sur la feuille augmentée. Elle représente donc le mode descriptif du dessin, celui utilisé pour présenter le résultat de la conception. Le rendu 3D de la scène est de type *Non Photo-Réaliste*, afin de renforcer l'aspect conception et de garder une certaine cohérence avec le dessin (voir figure 4.5). Comme l'a expliqué Robert ZELEZNIK pour son système *SKETCH* [Zeleznik *et al.*, 1996] et comme cela a été montré dans [Schumann *et al.*, 1996], ce type de rendu est indiqué dans une démarche créative où les idées ne sont pas forcément fixées et arrêtées. Cela permet de renforcer l'aspect imprécis et non définitif des choix, de ne pas focaliser l'attention de l'utilisateur sur des détails précis de l'aspect visuel du modèle 3D.

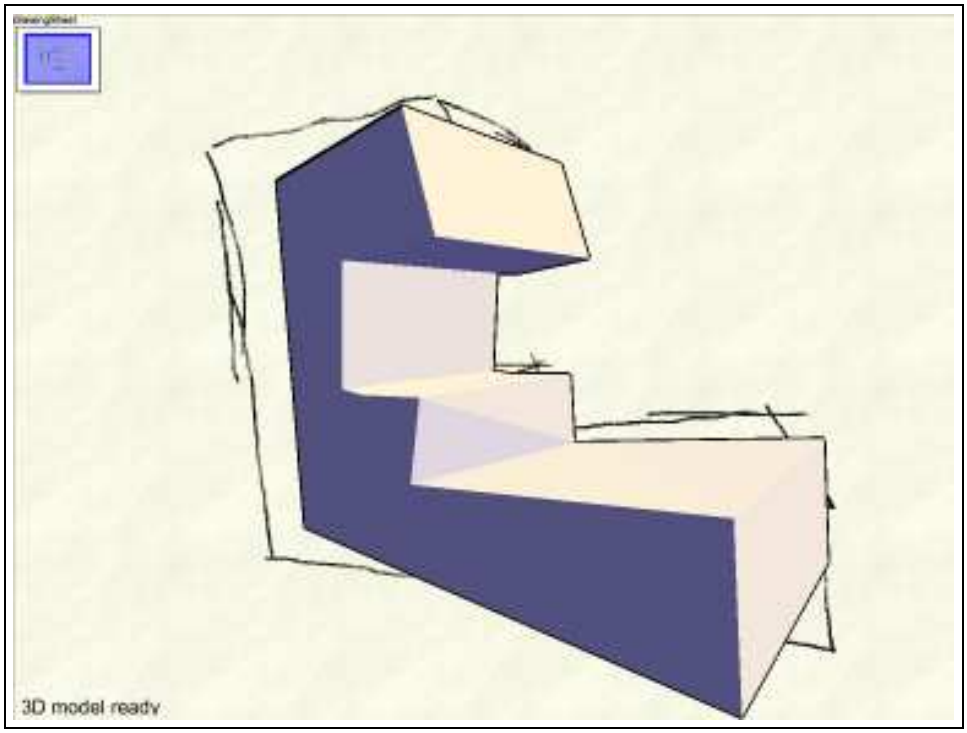


FIGURE 4.5 – Feuille 3D.

Pour ce qui est de son apparition et de sa visibilité, la feuille 3D n'est pas considérée comme « prioritaire ». En effet, dans le comportement que nous proposons, seule la feuille de dessin est visible lorsque l'utilisateur dessine. Lorsqu'il s'arrête, la feuille augmentée apparaît. La feuille 3D quant à elle, n'est visible que lorsque le modèle 3D est disponible (construit par le noyau) et que l'utilisateur souhaite le manipuler (en utilisant le périphérique d'entrée associé).

Ce n'est pas pour autant que cette feuille est inactive en arrière plan. Elle est en effet connectée au noyau de reconstruction de GINA et se charge donc aussi des retours de ce noyau : le modèle 3D évidemment, mais aussi tout autre information que renvoie le noyau (erreur de reconstruction, manque d'informations, etc.). Ce retour visuel du fonctionnement du noyau consiste en une zone de texte semi-transparente en bas à gauche de la zone de travail, signifiant l'état courant du noyau par

un message significatif, agrémenté d'un comportement de pulsation dépendant du type de message. Ce retour n'apparaît que lorsque l'utilisateur ne dessine pas garantissant ainsi un comportement non intrusif vis-à-vis de l'activité principale (le dessin).

Actuellement, la connexion de la feuille 3D au noyau de reconstruction n'est pas encore réalisée (la figure 4.5 page ci-contre est réalisée en « magicien d'Oz », le modèle 3D qu'elle présente étant codé). Toutefois, les comportements décrits de la feuille 3D sont d'ores et déjà réalisés, prêts à être connectés. La connexion avec le noyau, en plus évidemment de la reconstruction 3D, nous permettra aussi de réaliser une reprojection du modèle 3D sur la feuille de dessin après d'éventuelles manipulations de l'utilisateur. Ainsi, il pourra continuer à dessiner naturellement les parties du modèle qui n'étaient pas visibles auparavant. Bien entendu, il sera important d'étudier l'impact de cette reprojection et de sa représentation (rendu graphique, perte de l'historique de conception, etc.). Nous aborderons ces questions dans le chapitre suivant, portant sur les évolutions et perspectives de SVALABARD.

Le comportement global des feuilles d'interaction est donc de ne présenter que la feuille de dessin, le système laisse l'utilisateur dessiner. Lorsque celui-ci s'arrête, ou lorsqu'il le notifie explicitement, la feuille augmentée et la feuille 3D apparaissent avec une animation de transition. Il peut alors effectuer des opérations précises sur ces zones d'interaction, ou tout simplement recommencer à dessiner. Dans la section suivante, nous détaillons les outils et interactions de chacune de ces feuilles.

4.3 Instruments des feuilles d'interaction

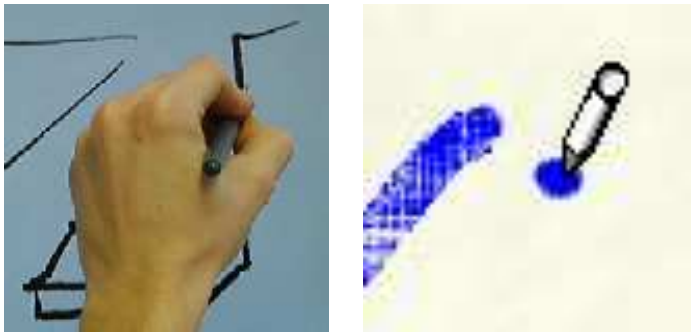
À chaque feuille sont associés des instruments qui définissent les interactions possibles avec le système. Toutes ces interactions sont des **outils**, le plus souvent directement associés à des périphériques d'entrée, renforçant alors la métaphore de table à dessin virtuelle (voir figure 4.2 page 85). Cette approche s'inspire directement du modèle de l'interaction instrumentale [Beaudouin-Lafon, 2000], extension de la manipulation directe [Shneiderman, 1983].
En plus de ces interactions et outils contextuels, il y a aussi des outils ou interactions que nous qualifions de « globaux », car actifs dans tous les modes.

4.3.1 Travail sur la feuille de dessin

La feuille de dessin, représentation du mode de dessin de l'interface, doit permettre deux activités principales et donc fournir deux outils adaptés : **dessin** et **manipulation des calques**.

Outil de dessin

Dans le souci d'appliquer les lignes directrices que nous avons proposées auparavant, la feuille de dessin représente un espace de dessin libre. Pour placer l'utilisateur dans ce contexte, l'outil correspondant se doit d'être de manipulation simple et de prise en main intuitive, afin de pouvoir rapidement « se faire oublier » : ne pas bouleverser les méthodes de travail, ni nécessiter un temps d'apprentissage trop long, phase pendant laquelle les activités de création seraient alors freinées. Enfin, pour être le plus fidèle possible aux méthodes traditionnelles, l'outil se doit aussi de procurer des résultats visuels proches du dessin papier/crayon, réduisant de fait l'écart entre la tâche réelle et son pendant informatique.



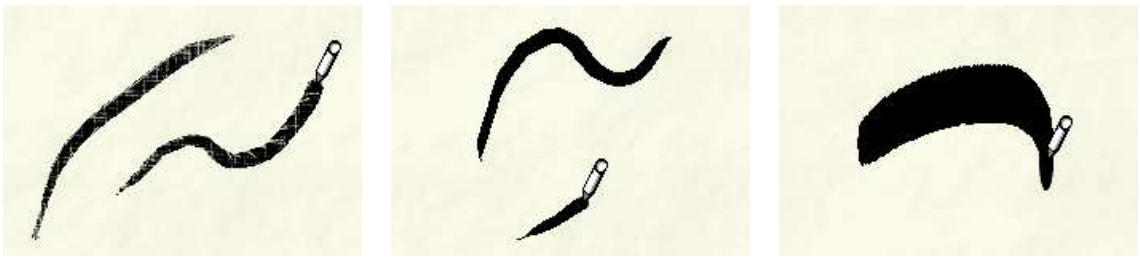
(a) La main et l'outil. (b) Détail du curseur.

FIGURE 4.6 – Outil de dessin de SVALABARD. Le retour graphique de l'outil renvoie, sous le curseur, des informations sur la taille et la couleur du trait (figure (a).

L'outil de dessin est conçu comme une adaptation des techniques traditionnelles (voir figure 4.6). Il permet de tracer naturellement les traits sur la feuille de dessin avec le **stylet** d'une **tablette écran**, directement sur la surface d'affichage (voir figure 4.7). C'est de cette association entre un outil logique de dessin et un tel périphérique d'entrée que la métaphore de « table à dessin virtuelle » prend son sens : le dessin libre et direct sur l'écran.

FIGURE 4.7 – Tablette écran utilisée pour dessiner avec SVALABARD.

Les traits produits peuvent alors présenter différentes apparences, selon la brosse sélectionnée : une brosse de type *crayon*, une de type *stylo* et une autre de type *calligraphie* (voir la figure 4.8 page ci-contre). L'épaisseur du trait est contrôlée par l'inclinaison du stylet par rapport à la tablette, son opacité par la pression exercée. Ces contrôles permettent de maîtriser l'aspect du trait d'une manière relativement proche de l'emploi d'un crayon.



(a) *Crayon*. (b) *Stylo*. (c) *Calligraphie*.

FIGURE 4.8 – Les brosses de l'outil de dessin de SVALABARD.

Enfin, la couleur de tracé peut être changée à l'aide d'un **boîtier de glissières** (voir figure 4.9). Ce type de périphérique d'entrée est de plus en plus utilisé dans les milieux de l'audiovisuel, proposant un mode d'interaction identique aux systèmes de mixage analogique (audio ou vidéo). Dans notre cas, nous avons utilisé ces « faders » pour contrôler les canaux *Rouge*, *Vert* et *Bleu*[(1)] de la couleur de tracé.

FIGURE 4.9 – Boîtier midi utilisé pour contrôler la couleur de l'outil de dessin de SVALABARD.

L'outil de dessin, tel qu'il est représenté dans la figure 4.6(b) page ci-contre, rassemble quatre retours graphiques. Tout d'abord, le curseur, permettant d'identifier l'outil. Ensuite, sous le point actif du curseur, se situe une représentation de la brosse de dessin utilisée par l'outil. Cette représentation de la brosse est assujettie aux changements de couleur et d'épaisseur de l'outil, ce qui permet d'identifier rapidement ces paramètres. Notons tout de même que le retour graphique lié à l'épaisseur du trait (taille de la brosse) étant dynamiquement lié au variations de l'inclinaison du stylet au cours d'un trait, il n'est pas d'une grande utilité lorsque l'utilisateur dessine, le trait dessiné faisant office de retour.

[(1)] Suite à des discussions avec des concepteurs et graphistes, en particulier Jean-Luc VINOT, concepteur d'interfaces au CENA, nous envisageons de proposer dans le futur un modèle de couleurs plus complet et adapté que le RVB, tel que les modèles de la normalisation CIE (CIE XYZ, CIE L*A*B, etc...[ISO/CIE, 1991]). Outre le fait que le RVB ne couvre pas tout l'espace des couleurs visibles, sa manipulation n'est pas *naturelle* pour le choix d'une couleur car basé sur une décomposition liée au matériel (pixels) et non au mode de pensée du graphiste [Vinot, 2004].

Toutefois, dans le cas où l'utilisateur déciderait de ne pas associer cette inclinaison à l'épaisseur du trait, ce retour est important pour visualiser l'épaisseur de trait choisie.

L'association d'une tablette graphique à un outil de dessin numérique compose un instrument tout particulièrement adapté, relativement proche du dessin *papier-crayon* dont il est inspiré. De plus, la réunion Tablette/Écran a permis une meilleure intégration de l'outil informatique pour une population plutôt réticente, en réduisant fortement les perturbations et le temps d'apprentissage induis par :

- l'utilisation d'un périphérique relatif tel qu'une souris pour dessiner,
- l'utilisation d'une tablette avec un écran standard, le principal problème étant de regarder ailleurs que là où la main travaille.

Les différentes brosses, et les rendus graphiques des traits qui leurs sont associées, ainsi que leurs épaisseur, couleur et opacité, font rentrer en jeu la notion de style de dessin dans notre système, permettant à l'utilisateur de varier les traits et styles d'expression : il influe sur la perception directe de l'œuvre et sur la signification qu'il donne à ses traits (trait majeur, révision, etc.). C'est à notre avis un apport important pour la conception créative, où le dessin est fait de révisions, de combinaison de traits plus ou moins appuyés.
Il est aussi important de noter que les interactions associées à la gestion de ces paramètres ne tendent pas à casser le processus de dessin comme le feraient des techniques standard basées sur des widgets de contrôle. Notre approche directement *instrumentée* par des périphériques d'entrée permet de contrôler les propriétés de l'outil simultanément à l'action de dessin : incliner le stylet ou varier la pression exercée dessus est un geste naturel pour le dessinateur, et l'utilisation des glissières pour varier la couleur peut être faite avec la main non dominante sans interrompre le dessin. Ce principe de changement de couleur pendant le dessin peut même être utilisée pour des effets de style, dans un contexte plus « artistique ».

Du point de vue du système et de l'interprétation du dessin, ces différents paramètres de l'outil n'influent pas de manière directe, le système d'interprétation étant directement connecté au périphérique d'entrée (tablette). Par contre, ils influent de manière indirecte, en offrant au dessinateur des possibilités d'expression plus complètes et artistique qu'un simple système de dessin au trait uniforme : épaisseur, couleur et opacité variables au cours d'un même trait, brosses aux aspects et comportements différents, etc.

Enfin, cet outil joue un rôle primordial dans les comportements de l'interface du système. Nous le qualifions **d'outil modal**, car son utilisation déclenche automatiquement les traitements d'interprétation du dessin ainsi que la disparition des feuilles augmentée et 3D. Ainsi, lorsque l'utilisateur dessine, il ne voit plus que son dessin, il n'est pas « distrait » par les comportements du système et les autres outils sont inactifs (à l'exception du contrôle des calques). Lorsque l'outil de dessin devient inactif, les autres feuilles redeviennent visibles et leurs outils accessibles. Ce paradigme de changement de mode implicite nous parait une alternative efficace et adaptée à des tâches créatives, en opposition aux interfaces standard, trop souvent surchargées de contrôleurs (widgets), de vues et de commandes qui ne sont pas conçus pour être utilisés dans un tel contexte, facilitant plus la vie des programmeurs que celle des utilisateurs (inadaptés à la manipulation directe des objets).

Manipulation des calques

La manipulation des calques de dessin doit permettre plusieurs opérations : créer un nouveau calque, sélectionner le calque courant de dessin ou les réordonner. L'approche traditionnelle pour la

manipulation de tels objets dans les logiciels en général consiste à proposer un widget représentant des vues miniatures des calques. L'utilisateur peut alors sélectionner des propriétés simples des calques ou les réorganiser à l'aide de la souris, ou de sa tablette graphique. Les fonctionnalités étendues sont accessibles par menu contextuel. Ce mode d'interaction pose selon nous deux problèmes majeurs. Tout d'abord, il interrompt l'activité de dessin, obligeant à déplacer le pointeur sur le widget de manipulation (les solutions par raccourcis clavier ne font, à notre avis, qu'ajouter à la peine de l'utilisateur par leur mémorisation difficile). Ensuite, l'espace occupé par un tel composant sur la zone de travail est conséquent, surtout si beaucoup de fonctionnalités sont proposées. D'autres solutions ont été proposées, comme par exemple l'utilisation d'onglets dans CPNTools [Beaudouin-Lafon et Lassen, 2000]. Bien adaptée à la notion et la manipulation de calques d'interaction (comme le sont nos *feuilles*, d'ailleurs), cette solution n'est à notre avis pas assez directe pour l'interaction avec des calques de dessin.

FIGURE 4.10 – *Shuttle* (périphérique réunissant molettes et boutons) utilisé pour la création et sélection des calques de la feuille de dessin de SVALABARD.

Nous proposons un approche directe de manipulation des calques de dessin avec un périphérique d'entrée particulier : le **Shuttle**. Nous l'avons utilisé pour créer, supprimer, sélectionner et ordonner les calques (voir figure 4.10). Ce périphérique est composé de deux molettes superposées, une absolue et une relative, ainsi que de boutons. La molette absolue (*isotonique*) permet de sélectionner le calque actif (celui sur lequel l'utilisateur dessine) parmi les calques existant, de manière incrémentale. La molette relative (*élastique*), permet de plier le calque courant afin de visualiser les calques du dessous (nous reviendrons plus loin sur cette interaction de pliage). Les boutons du dispositif proposent quant à eux les fonctions de création et suppression d'un nouveau calque de dessin, ainsi que le contrôle de leur position relative.

Les résultats des manipulations des calques sont visibles à la fois sur la feuille de dessin elle-même et sur un composant graphique dédié à leur représentation. Lorsque le calque actif change, seul celui-ci reste visible sur la feuille de dessin pendant un cours laps de temps afin de le signifier

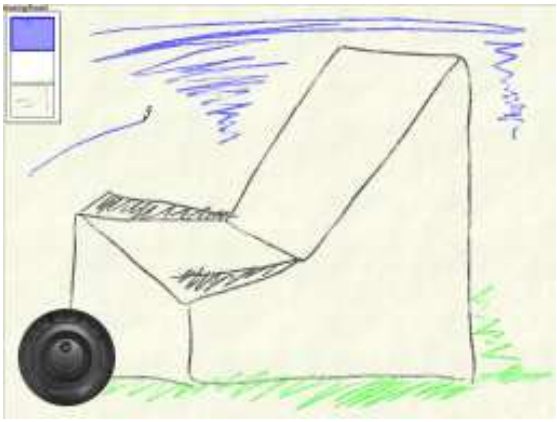
clairement à l'utilisateur. Le composant graphique des calques (voir figure 4.11) présente quant à lui des vues miniatures du contenu de chaque calque, vues ordonnées du bas vers le haut conformément leur disposition. La vue miniature du calque actif est surlignée et agrandie par rapport aux autres et les transitions lors des changements de calque ou leur reordonancement, en plus d'être naturellement visible sur la feuille de dessin, sont animées sur la vue miniature. Pour ne pas gêner ou masquer des zones du dessin, ce composant graphique peut évidemment être déplacé interactivement. Mais il peut aussi être rendu semi-transparent ou complètement invisible.

FIGURE 4.11 – Miniatures des calques de dessin. La vue du calque actif est surlignée et agrandie.

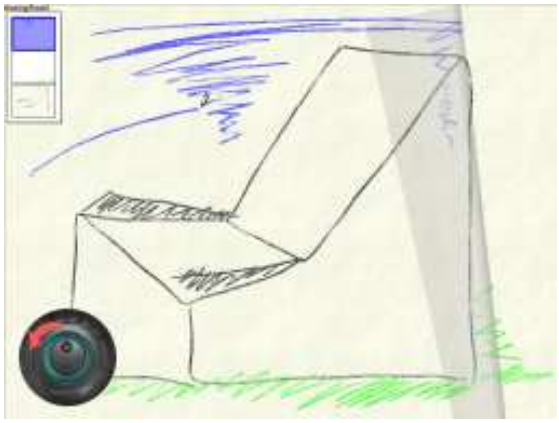
Pliage des calques

Comme nous l'avons déjà évoqué, la molette élastique du périphérique permet une interaction originale avec le calque actif : le *pliage*. Michel BEAUDOUIN-LAFON, dans [Beaudouin-Lafon, 2001], a proposé d'utiliser des techniques usuelles de manipulation de feuilles pour manipuler des fenêtres d'application. Nous proposons d'utiliser l'une de ces méthodes, le pliage, de manière encore plus naturelle dans notre contexte de dessin sur calques. En effet, quoi de plus habituel que de replier un calque pour voir celui du dessous, ou même y dessiner. C'est ce que permet la molette élastique du shuttle dans SVALABARD. Comme le montre la figure 4.12 page suivante, le déplacement vers la gauche de cette molette plie le calque actif de la droite vers la gauche (et inversement de la gauche vers la droite). Le pliage suit la position de la molette : au centre, le calque n'est pas plié ; à fond sur la gauche, le calque est totalement plié, laissant apparaître complètement ce qui est en dessous. Réciproquement, la rotation de la molette vers la droite permet de plier le calque de la gauche vers la droite. Si, comme dans la figure 4.12(c) page ci-contre, le calque que plie l'utilisateur n'est pas au sommet de la pile, les calques au dessus de lui sont aussi pliés. Finalement, et c'est l'un des atouts de cette technique, il est possible lorsqu'un calque est plié de dessiner sur le calque du dessous.

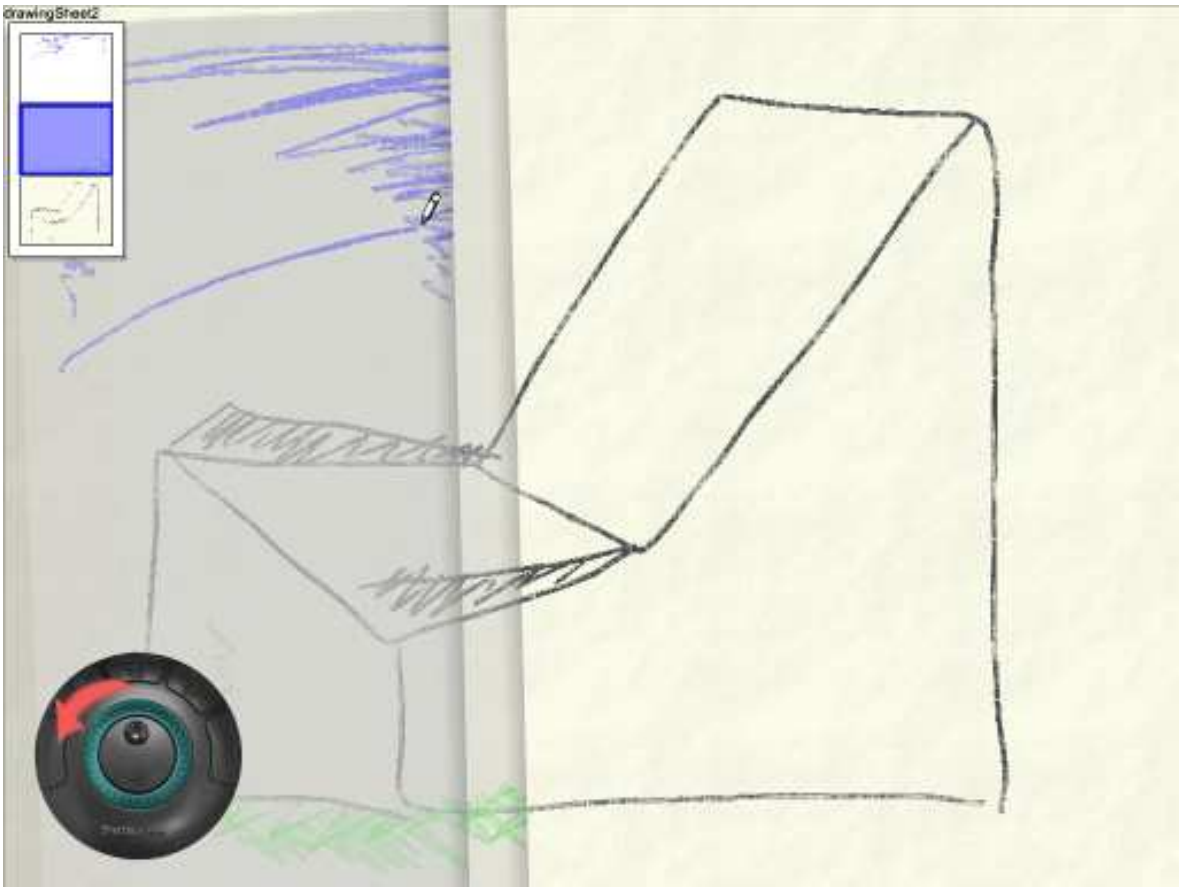
Nous voyons deux avantages majeurs à cette technique. Premièrement, elle permet de rapidement visualiser le contenu d'un calque recouvert ou de dessiner dessus sans avoir à changer de calque actif. Cela évite la notion troublante de calques virtuels qui permettent de dessiner sur un calque masqué. Ensuite, l'aspect naturel du pliage, de par sa réalisation graphique (semi-transparence, rendu de la partie pliée, ombre portée) et l'interaction associée (manipulation de la main non dominante permettant de continuer le dessin avec la main dominante), renforce encore une fois notre métaphore de table à dessin virtuelle.



(a) La feuille de dessin comporte trois calques.



(b) Le calque actif (celui du sommet, avec le dessin en bleu) est plié.



(c) Lorsque le calque actif n'est pas au sommet, tous les calques au dessus de lui sont aussi pliés.

FIGURE 4.12 – Pliage des calques de dessin dans SVALABARD.

L'instrument dédié aux calques réunit donc un périphérique d'entrée et un widget pour les contrôler aisément. La polyvalence du *Shuttle* permet de regrouper toutes ses commandes sur ce seul périphérique, un unique « espace » d'interaction. Ainsi, son rôle est rapidement assimilé par l'utilisateur. Il doit tout de même mémoriser les commandes, que nous avons essayé de disposer au mieux sur le dispositif. Mais nous pouvons tout de même supposer que cette étape d'apprentissage est anecdotique par rapport à la mémorisation de raccourcis claviers, ou le temps perdu à manipuler un widget trop complet. De plus l'utilisateur peut lui même redéfinir les attributions des fonctions du périphérique (voir section 4.3.5 page 109).

Enfin, le widget associé (vues miniatures) n'est présent que comme retour graphique, ne faisant pas office de contrôleur de l'interaction avec les calques. Bien que les actions soient visibles sur la feuille de dessin, il nous semble aussi utile de fournir une vue hiérarchique de la pile de calques. C'est pour cela que nous avons repris cette notion de séparation des calques, courante dans les logiciels de dessin. Mais bien qu'habituelle, elle ne fait appel à aucune représentation logique ou sémantique particulière. Elle permet juste une appréciation rapide de l'ordre des calques et de celui qui est actif. Évidemment, un utilisateur plus « averti » peut s'en passer, et n'utiliser que le pliage pour « naviguer » dans la pile. Cette notion de pliage, par contre, a une sémantique précise, liée à une action du quotidien. C'est dans ce sens bien plus qu'un simple gadget graphique car replaçant le dessinateur dans un environnement familier.

4.3.2 Travail sur la feuille augmentée

La feuille augmentée regroupe les fonctionnalités supplémentaires que peut apporter un outil de dessin numérique par rapport au traditionnel papier/crayon. Elle offre une représentation graphique des analyses effectuées sur le dessin par le système, et propose donc des outils d'édition liés à ces analyses : la saisie de propriétés géométriques. Outre cette saisie, elle permet aussi d'annoter le dessin.

Annotations

Les annotations sont courantes sur les croquis de conception, permettant des explications et améliorant aussi la communication des idées entre collaborateurs. Il est évident qu'un système de dessin numérique doit aussi proposer un tel outil pour ne pas rompre avec les habitudes du concepteur, mais aussi pour renforcer cette notion d'explication et d'historique des croquis dans un contexte où les possibilités d'archivage et de fouille documentaire sont en constante évolution (voir [Coüasnon et Camillerapp, 2003] et plus généralement la revue *Document numérique* [Lavoisier-Hermes science publications, 2000 2004]).

Il reste encore à rendre cette saisie d'annotations des plus naturelles, en évitant de rompre avec la ligne directrice de notre interface basée sur le dessin (proscrire la saisie au clavier, par exemple). La saisie des annotations dans SVALABARD est donc aussi une interaction de dessin à main levée. Elle ne doit alors pas interférer avec l'action de dessin, et c'est pour cela que nous avons choisi de placer cette saisie d'annotations sur la feuille augmentée : c'est une première discrimination sémantique (les traits sur la feuille de dessin sont du dessin, sur la feuille augmentée des compléments au dessin). Ces traits ne seront donc pas interprétés par le système d'analyse du dessin, et ne seront visibles que sur la feuille augmentée (voir figure 4.13 page ci-contre).

Pour la manipulation de cet outil, une **seconde tablette** est utilisée. Elle est disposée à proximité

FIGURE 4.13 – Outil d'annotations de la feuille augmentée de SVALABARD.

de la tablette écran, permettant ainsi de passer rapidement d'un outil à l'autre en utilisant le même stylet sur des tablettes différentes (voir figure 4.14). Cette disposition est dans le même esprit que le périphérique proposé par la société *Wacom*, le *C*intiqPartner™ [Wacom Europe GmbH, 2003], en complément de leurs tablettes écran[(2)].

FIGURE 4.14 – Interaction avec deux tablettes graphiques. La tablette écran est utilisée pour dessiner, alors que la seconde tablette permet de tracer de simples annotations avec le même stylet. Le changement d'outil est naturel, sans avoir à changer de stylet.

Il aurait été envisageable d'utiliser deux stylets et une seule tablette, chaque crayon correspondant alors à un outil logique différent. Toutefois, cette approche aurait impliqué un changement de crayon pour changer d'outil, ce qui n'est pas le cas dans la réalité. Un architecte réalisant un croquis et voulant l'annoter utilisera le même crayon. C'est pourquoi nous avons conservé notre métaphore du *changement d'espace* : le changement d'espace virtuel (feuille de dessin/feuille augmentée) est associé

[(2)]Notons que nous avions étudié cette disposition et cette configuration à deux tablettes avant la présentation de cet outil par Wacom.

à un changement d'espace physique. Ce changement de surface de contrôle (tablette écran/tablette standard) fournit cette fois une discrimination physique entre les deux activités et outils similaires : dessiner/annoter.

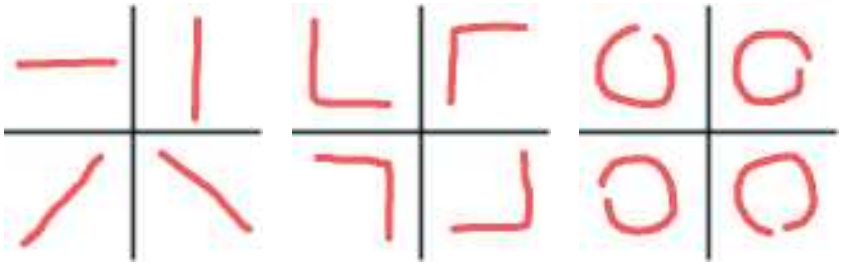
L'outil d'annotations de SVALABARD, par sa manipulation similaire à l'outil de dessin, ne brise pas la cohérence dans les interactions du système. Les espaces de travail distincts, supportés par l'utilisation d'espaces de visualisation (feuilles) et d'interaction (tablettes), permet en outre une association claire entre les outils et leur fonction, réduisant ainsi à notre avis les erreurs qui pourraient être induites par la saisie d'annotations directement sur la tablette écran, avec un autre stylet ou un modificateur (bouton) par exemple.

Saisie de propriétés géométriques

Un autre outil est actif lorsque la feuille augmentée est visible : un outil de saisie de contraintes et propriétés. En effet, l'élévation 3D de dessins à main levée nécessite la connaissance de propriétés géométriques des objets à reconstruire (parallélisme, orthogonalités, incidences). Bien que détectant déjà des propriétés importantes de la structure du dessin (voir la section 4.4 page 110), SVALABARD devra proposer à terme une détection et le traitement automatique de propriétés de plus haut niveau. Nous proposons actuellement un outil original de spécification des contraintes. Celui-ci permet de saisir des contraintes de parallélisme ou d'orthogonalité entre segments et des contraintes de coplanarité entre points par **reconnaissance de gestes**. Notons toutefois que, même en présence d'une détection automatique, un tel outil justifie sa présence pour modifier et corriger d'éventuelles erreurs du système.

Spécifier des propriétés entre des objets géométriques, comme par exemple le parallélisme de plusieurs droites, revient à sélectionner ces objets et à annoncer la propriété qui les lie (de la même manière qu'un énoncé géométrique dirait *les droites AB* et *CD* sont parallèles). La réalisation d'un tel outil dans notre système pourrait reprendre ce principe d'énoncé en proposant un outil de sélection, pour désigner les segments, et un outil de commande pour spécifier le parallélisme. Une telle manipulation directe aurait déjà l'avantage d'être plus cohérente avec notre approche qu'une méthode de saisie des contraintes par boîtes de dialogue ou un langage de script (que l'on trouve fréquemment dans des systèmes de modélisation 3D ou de dessin géométrique). Nous proposons une méthode plus efficace en regroupant ces deux phases de sélection et d'énoncé dans une interaction de saisie de contraintes par reconnaissance de gestes : les **gestes de commande**, proposés par Stéphane CHATTY dans [Chatty et Lecoanet, 1996]. Cette saisie s'opère par manipulation directe des segments et points issus de l'interprétation des traits du dessin, sur la feuille augmentée.

Chaque propriété géométrique est représentée par un groupe cohérent de gestes (voir figure 4.15 page suivante). Un groupe contient plusieurs gestes, qui ne différent que par leur orientation. Ainsi, le parallélisme est représenté par un *trait*, avec huit orientations possibles (voir figure 4.15(a)), l'orthogonalité par un angle droit (voir figure 4.15(b)) et la coplanarité par un cercle (voir figure 4.15(c)). Ces gestes spécifient la contrainte, mais permettent aussi la sélection des objets, combinant alors les fonctions *épistémique* et *ergotique* [Cadoz, 1994]. Il suffit de tracer le geste correspondant sur chaque segment ou point auquel on veut appliquer la propriété désirée. Cela permet la sélection successive d'un nombre illimité d'objets, un par geste identique, pour leur appliquer une propriété commune. L'interaction, et donc la saisie, est validée automatiquement au bout de quelques secondes si aucune nouvelle sélection n'intervient, ou si un objet est sélectionné avec un geste différent.



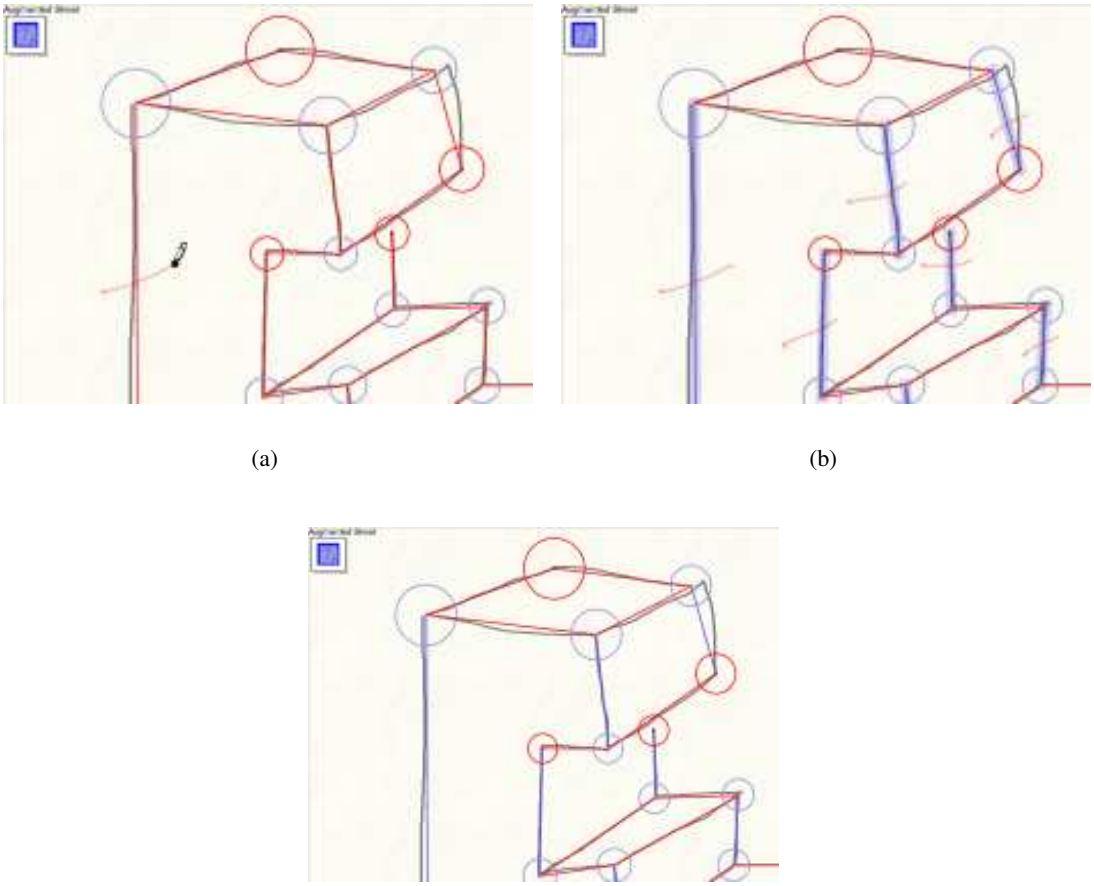
(a) Parallélisme. (b) Orthogonalité. (c) Coplanarité.

FIGURE 4.15 – Gestes associés aux contraintes. Les points de départ des gestes ne sont pas représentés, ceux-ci pouvant être initiés d'une extrémité ou de l'autre.

Suivons l'exemple de la spécification d'une contrainte de parallélisme sur trois segments à l'aide de la figure 4.16 page suivante. La première figure présente la feuille augmentée. Le dessinateur a tracé plusieurs traits à l'aide de l'outil de dessin de la tablette écran et peut voir leurs interprétations et les segments qui en découlent sur la feuille augmentée. En utilisant le côté *gomme* du stylet sur la seconde tablette, il sélectionne avec le geste approprié les segments qu'il veut spécifier parallèles (voir figure 4.16(a)). Dans la figure 4.16(b), plusieurs segments sont sélectionnés. Cette sélection est représentée par le surlignage des segments. Le geste utilisé persiste pour rappeler la propriété en cours de spécification. Enfin, la troisième figure présente le résultat de l'action, validée après un temps sans sélection. Les segments ne sont plus surlignés, mais sont affichés de la même couleur pour pouvoir rapidement identifier les faisceaux.

La spécification de contraintes d'orthogonalité sur des segments ou de coplanarité sur des points se déroule de la même manière, avec le geste approprié (voir figure 4.17 page suivante pour les points coplanaires). Nous avons, dans notre prototype, réalisé un module très simple et basique de maintient de la cohérence des propriétés géométriques (transitivité, propagation, etc.) dans le but de prouver la faisabilité de l'interaction. Il est évident qu'une fois le système connecté au noyau mathématique de reconstruction 3D, les propagations et vérifications sur ces contraintes lui seront entièrement déléguées.

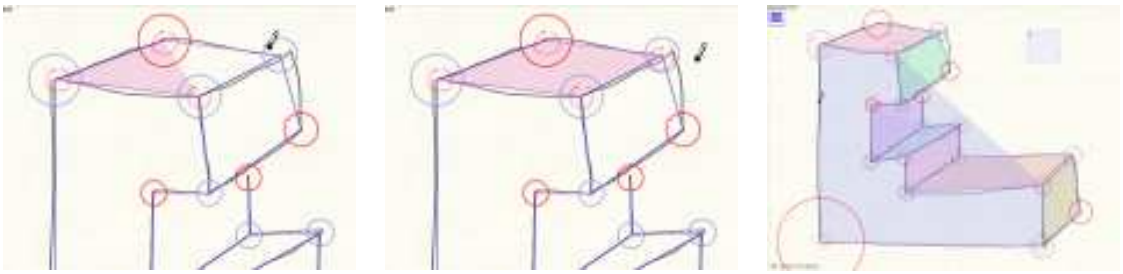
De la même manière que pour les annotations, cette saisie des propriétés géométriques se déroule sur la seconde tablette et non sur la tablette écran. Cela reprend le principe de l'association feuille de dessin/tablette écran, feuille augmentée/tablette. Nous avions envisagé, dans un premier temps, de réaliser cette interaction sur la tablette écran. Mais bien que cela aurait rendu plus précise la sélection des objets de la feuille augmentée, cela aurait aussi brisé l'association périphériques/vues qui nous semble plus importante. Dans ce choix, l'utilisation du côté *gomme* du stylet permet de distinguer cet outil de celui des annotations à main levée.



(a) (b)

(c)

FIGURE 4.16 – Étapes de la notation de droites parallèles. Les segments sont sélectionnés avec le geste approprié. Ils sont alors surlignés et les gestes de sélection restent visibles (figure (a) et (b)). Au bout d'un temps sans sélection, la saisie est validée. Les segments sont affichés de la même couleur pour notifier leur parallélisme.



(a) Sélection de trois points. (b) Une autre sélection. (c) Après plusieurs étapes.

FIGURE 4.17 – Étapes de la notation de points coplanaires.

4.3.3 Manipulations sur la feuille 3D

FIGURE 4.18 – Interaction 3D avec le *Magellan* (ou *Space-Mouse*).

L'interaction sur la feuille 3D se résume à la manipulation de la vue du modèle reconstruit par le système. Pour cette interaction, nous utilisons un dispositif isométrique à six degrés de liberté, le **Magellan** (ou « Space Mouse »). La figure 4.18 représente ce dispositif, ainsi que ses axes : trois axes de translation et trois axes de rotation. L'emploi de ce périphérique permet la manipulation directe et précise[(3)] de la vue dans toutes les dimensions avec la main non dominante et donc sans poser le stylet de dessin ni changer d'outil.

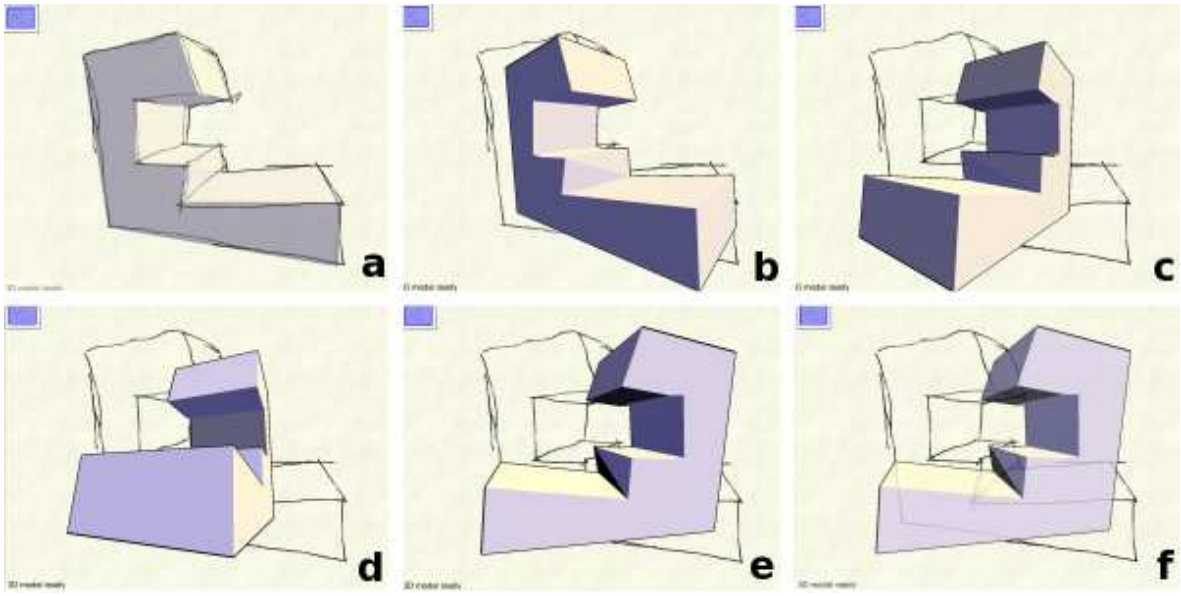


FIGURE 4.19 – Manipulation du modèle 3D. Après son apparition, la feuille 3D reste semi-transparente si aucune autre action n'est effectuée (cadre *a*). Lorsque la vue est manipulée, elle devient opaque (cadres *b* à *e*) pour redevenir semi-transparente avant de redisparaître si aucune action n'est effectuée ou si le dessin reprend (cadre *f*).

[(3)] Cette notion de précision est importante pour la manipulation d'une vue en trois dimension. Elle est essentiellement due à la conception même de ce dispositif isométrique, la force déployée pour le manipuler influant directement sur les données qu'il émet.

Une action sur ce périphérique va faire apparaître la feuille 3D si le modèle est disponible. Dès lors, elle va rester un instant semi-transparente avant de redisparaître si l'utilisateur interrompt l'action. Il est ainsi possible de voir à la fois le modèle 3D et le dessin original par superposition. Lorsque l'utilisateur continue à agir sur le périphérique pour déplacer la vue, la feuille 3D devient opaque avant de redisparaître un instant après la manipulation ou lorsque l'utilisateur reprend son dessin (voir figure 4.19 page précédente).

Comme nous l'avons déjà exposé lors de la description de la feuille 3D, l'intérêt de cette manipulation de la vue est aussi de permettre au concepteur de compléter son modèle dans d'autres points de vue, de créer en dessinant dans toutes les dimensions. Encore une fois, notre approche par l'instrumentation directe des fonctionnalités du système rend de telles techniques plus naturelles : elle permet dans ce cas précis de changer de point de vue avec la main non dominante, la main dominante étant alors prête à dessiner dans le nouveau point de vue.

4.3.4 Interaction globale

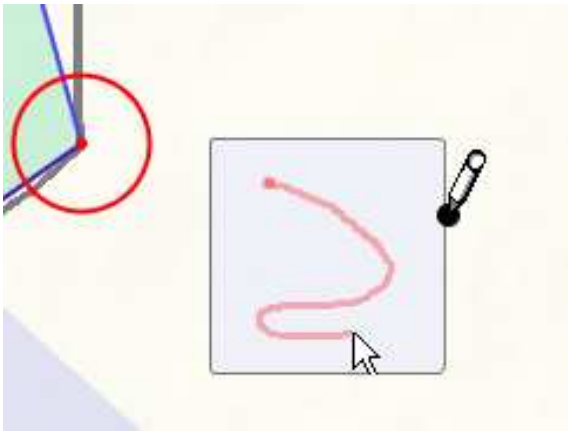
En complément des outils contextuels liés aux différentes feuilles, il est aussi nécessaire d'inclure dans SVALABARD un outil global. Cet outil permet de contrôler les fonctionnalités globales de l'application (quitter, configuration, etc.) et c'est pourquoi il doit être accessible dans tous les modes. Toutefois, il est important qu'il ne brise pas non plus la cohérence de la métaphore de table à dessin avec des modes d'interaction radicalement différents de ceux utilisés pour les outils contextuel. C'est tout naturellement que ce principe nous à conduit à adopter des techniques de reconnaissance de gestes. Nous proposons deux alternatives dans SVALABARD : une zone de reconnaissance de gestes et des *marking menus*.

Zone et outil de commande gestuelle

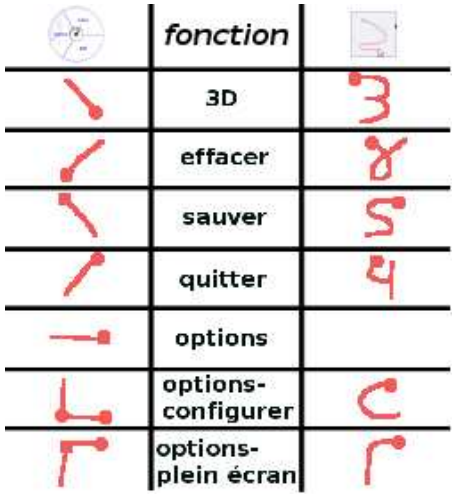
Afin de rester dans un paradigme d'interface de dessin, il est essentiel d'utiliser un style d'interaction tel que la reconnaissance de gestes pour interagir avec le système. Toutefois, cela soulève des problèmes d'interférence entre les interactions dans le cadre d'une interface de dessin comme la notre. Admettons en effet qu'un système d'interprétation de gestes soit intégré directement à notre système de saisie de dessins. Comment différencier alors un trait du dessin en cours de réalisation, d'un geste de commande ? Cette question est un des problèmes récurant des « sketching-interfaces ». Dans un cadre précis comme celui choisi dans TAPAGE [Faure et Julia, 1993; Julia, 1995], où les commandes peuvent être clairement différenciées des formes dessinées imposées ou associées à des commandes parlées, cette ambiguïté est levée par les interpréteurs appropriés. Il serait envisageable, dans notre approche *instrumentée*, d'utiliser un autre périphérique d'entrée pour saisir les gestes (autres stylets ou boutons de stylet, tablettes), mais nous pensons que des changements d'outils trop fréquents deviendraient trop contraignants pour l'utilisateur, quitte même à entraîner des erreurs.

C'est pourquoi nous proposons de conserver le même outil (physique et logique) que l'outil de dessin ou d'annotations, mais de spécialiser sa fonction à la reconnaissance de gestes dans les limites graphiques d'une zone spéciale, présente sur toutes les feuilles sous la forme d'un rectangle semi-transparent (voir figure 4.20 page ci-contre). À l'intérieur de cette zone, l'outil manipulé par le stylet sur la tablette écran ou sur la seconde tablette est entièrement dédié à la reconnaissance de gestes, selon le principe déjà connu de la zone de saisie des assistants personnels. Ainsi, pour effectuer une commande lors de n'importe quelle tâche, nul besoin de changer d'outil ou de mode, que l'utilisateur

travaille sur la feuille de dessin (tablette écran) ou sur la feuille augmentée (tablette standard). Il suffit de tracer le geste dans la zone de commande (qui peut aussi être déplacée pour ne pas gêner).



(a) Zone graphique de reconnaissance de gestes.



(b) Gestes associés aux commandes. À gauche, les gestes des « marking-menus » à droite, ceux de la zone de geste.

FIGURE 4.20 – Zone de reconnaissance de gestes et gestes associés aux commandes.

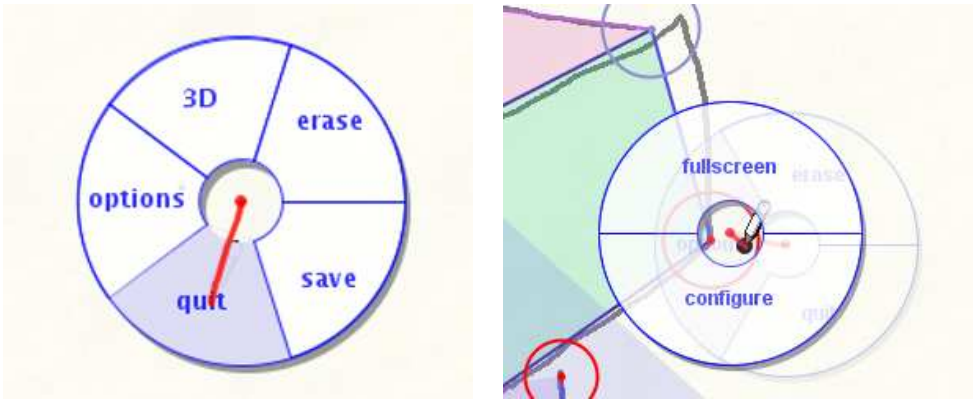
Cette solution permet d'utiliser des gestes *sémiotiques* pour accéder aux fonctionnalités de l'application, sans toutefois interférer avec le dessin et sans changer explicitement de périphérique d'entrée, ni de mode (ou activer un modificateur tel un bouton ou une touche du clavier). Toutefois, cette méthode peut imposer des déplacements fréquents sur l'espace de travail. Nous proposons alors une autre alternative : les « marking menus », fusion de menus circulaires et de reconnaissance de gestes.

Alternative aux gestes : les marking menus

En 1991, Gordon KURTENBACH et William BUXTON [Kurtenbach et Buxton, 1991] ont introduit les « marking menus », combinaison de menus circulaires (« pie menus ») [Wiseman *et al.*, 1969; Hopkins, 1991] et de gestes rectilignes simples. Les gestes permettent de sélectionner l'entrée désirée du menu circulaire et peuvent être enchaînés dans des sous-menus hiérarchiques. Ces menus ont été conçus pour être utilisés par des utilisateurs de plusieurs niveaux d'expertise. Ainsi, si l'on trace directement le geste correspondant à un item du menu, celui-ci n'apparaît pas (mode expert). Par conte, un appui plus long sur le périphérique d'entrée (stylet ou souris) permet de faire apparaître le menu afin de « rafraîchir » la mémoire de l'utilisateur. Dans une étude plus poussée sur leurs travaux [Kurtenbach et Buxton, 1994], les auteurs on montré les avantages des « marking menus » par rapport aux menus linéaires, ainsi que leurs apports dans les interfaces, en particulier celles basée sur les périphériques de type « stylet » (cohérence, rapidité et apprentissage).
Au vu de ces avantages et des possibilités offertes par notre plate-forme de développement, nous avons

rapidement introduit un tel menu dans l'interface de SVALABARD, afin de proposer à l'utilisateur une alternative à la zone de gestes exposée précédemment.

Les « marking menus » de SVALABARD sont contrôlés par le stylet, en pressant son bouton latéral comme modificateur à l'outil actif (dessin ou annotations). De la même manière que pour la zone de geste, le menu est accessible depuis les deux tablettes. Il comporte deux niveaux, comme le présente la figure 4.21. Le menu principal contient les fonctions « quitter, effacer, options, sauvegarder et 3D » (voir figure 4.21(a)). Notons que les deux dernières fonctions de ce menu ne sont pas encore implémentées dans le système. Le choix « options » présente un menu de second niveau offrant l'accès aux fonctions « plein écran et configuration » (voir figure 4.21(b)). Ces menus se comportent comme ceux décrits par Gordon KURTENBACH, à savoir qu'ils n'apparaissent pas lorsque la « marque » (le geste) est tracé rapidement (le délai d'apparition et la visibilité du geste sont paramétrables). Les gestes associés aux items de ces menus sont représentés dans la figure 4.20(b) page précédente.



(a) Menu de premier niveau. (b) Menu de second niveau.

FIGURE 4.21 – Marking menus de SVALABARD.

Bien que similaires, ces deux méthodes de contrôle des fonctionnalités de l'application ne sont pas conçues pour cohabiter (les gestes de la zone de dessin sont différents de ceux induits par la disposition des items des Marking menus). L'utilisateur doit faire un choix entre ces deux méthodes, et nous pensons qu'une fois familiarisé avec l'une d'elle, il lui serait difficile d'utiliser l'autre à moins d'une période d'apprentissage (bien que l'ensemble des gestes de commande puisse être redéfini et que les items des menus puissent être replacés).

Vue d'ensemble

La figure 4.22 page suivante présente la disposition de tous les périphériques qui font de SVALABARD une **table à dessin virtuelle**. Les deux tablettes forment deux espaces d'interaction : la feuille de dessin pour la **tablette écran** (dessin à main levée), la feuille augmentée pour la **tablette standard** (saisie de propriétés géométriques). Ensuite, chaque périphérique est affecté à une action précise : la **boîte à glissières** pour contrôler les paramètres de l'outil de dessin, le **shuttle** pour la manipulation des calques de dessin et le **Magellan** pour la vue de la feuille 3D (éventuellement une souris pour dépla-

cer des composants). Avec cette disposition, alors que la main dominante peut dessiner et « piloter » le système sur les espaces d'interaction, la main non dominante peut agir sur le contexte et les outils, sans interrompre la tâche [Guiard, 1987].

FIGURE 4.22 – SVALABARD, la *Table à dessin virtuelle* dans son environnement concret.

Considérons alors l'adéquation des dispositifs utilisés par rapport aux activités à réaliser. Le tableau 4.1 récapitule les associations que nous avons proposées entre les périphériques d'entrée et les modes de l'application (feuilles d'interaction), selon qu'ils sont utilisés pour réaliser une activité ou manipuler le contexte.

	Activité	*Contexte*
Feuille de dessin	**Tablette écran** (dessin)	**Shuttle** (calques), **Sliders** (couleurs)
Feuille augmentée	**Tablette standard** (annotations et contraintes)	
Feuille 3D		**Magellan** (vue 3D)
Global	**Tablettes** (gestes et Marking menus)	

TABLEAU 4.1 – Récapitulatif des associations périphériques d'entrée/outils de l'application.

Tout d'abord, les activités sont réalisées avec les périphériques *isotoniques*[(4)] que sont les **tablettes graphiques**. Ces dispositifs étaient évidemment les plus appropriés, les outils de l'application étant

[(4)]Les périphériques isotoniques bougent librement (souris, tablette, etc.). Ils sont adaptés au contrôle de position. Les périphériques isométriques sont quasiment immobiles (Magellan, Trackball, etc.), c'est la force exercée qui détermine l'action. Ils facilitent plus le contrôle de la vitesse. Certains emploient le terme d'élastique pour des périphériques à mi-chemin entre les deux (Joystick).

tous basés sur des interactions de position inspirées du dessin (dessin, gestes et Marking menus). Au delà de l'adéquation de ces dispositifs avec les différentes activtés de dessin, l'intérêt est aussi d'utiliser une tablette par espace d'interaction pour faciliter la perception des modes de l'application et le passage de l'un à l'autre.

Ensuite, les actions sur le contexte sont réalisées à l'aide de dispositifs fixes sur lesquels sont montées des commandes mobiles (boîte à glissières, molettes, etc.). Ces commandes nécessitent peu de précision dans leur manipulation et en font des dispositifs adaptés à la main non dominante. Ils s'utilisent comme des périphériques *isométriques* limitant au maximum le déplacement de la main et se manipulant du bout des doigts.

- Le **Shuttle** combine deux molettes : l'une est bien isométrique, l'autre est isotonique.
- La **boîte à glissières** n'enregistre pas une force, mais un déplacement linéaire contraint.
- Le **Magellan**, propose trois degrés de liberté homogènes pour chacune des deux opérations de déplacement (translation et rotation).

Le Magellan présente toutes les caractéristiques nécessaires à la manipulation d'une vue 3D, c'est même pour cela qu'il a été conçu. Une alternative possible serait d'employer d'autres dispositifs possédant des groupes de trois degrés de liberté homogènes tels qu'une *SpaceBall*.

Les glissières permettent une manipulation continue de chaque composante de couleur, offrant en plus une indication visuelle des réglages (même si elle est imprécise, elle est toutefois renforcée par le retour graphique de l'outil). Une autre approche serait de baser la sélection des couleurs sur le cercle chromatique en utilisant alors une seule glissière plutôt que trois, ou bien un dispositif tel qu'une *TrackBall* pour se positionner sur un *disque de couleurs*. Il serait alors indispensable de fournir un retour graphique complexe dans l'interface, ce que la décomposition évite en permettant de n'afficher qu'un retour minimal sur la couleur choisie.

Le Shuttle est le dispositif dont l'emploi est le plus éloigné de son utilisation habituelle. Les molettes sont généralement utilisées pour des opérations de sélection ou de défilement d'objets dans une liste (ce que l'une des molettes permet d'ailleurs avec la pile de calques). Par contre, plier une feuille est normalement une tâche de déplacement, ce que n'évoque pas ce dispositif. Nous aurions alors pu proposer une interaction employant un périphérique isotonique, reproduisant la manipulation du monde réel. Mais une telle action est moins adaptée à la main non dominante qu'une simple variation de taux de pliage comme nous l'avons proposée avec la molette élastique. Cette même interaction pourrait être aussi effectuée avec une glissière ou avec un *TouchPad*. Par contre, nous briserions alors le principe de notre interface d'associer un dispositif à un ensemble d'actions sur un objet d'intérêt.

Pour conclure, nous nous référerons aux travaux de William BUXTON portant sur l'interaction en entrée. Il suggère que les périphériques d'entrée ont une part importante dans l'appréciation que se font les utilisateurs d'un système et des activités possibles, insistant aussi sur l'importance de l'adéquation du dispositif avec la tâche à réaliser [Buxton, 1986]. Cela soulève le problème de la *compatibilité* entre un dispositif et une tâche, par la similitude des actions physiques et de leur résultats (*compatibilité stimulus/réponse*), mais aussi par l'accord entre les *propriétés intrinsèques* du dispositif et les caractéristiques de la tâche [Buxton, 1986; Dragicevic, 2004b].
Par nos hypothèses de conception et nos choix de dispositifs, nous avons tenté d'offrir à la fois une bonne *visibilité* du système et un haut niveau de *compatibilité* entre les dispositifs et les activités. Bien sur, seule une évaluation de l'utilisabilité de SVALABARD dans un contexte réel d'utilisation confirmera ces propositions (nous reviendrons sur ce point dans le prochain chapitre). Mais cette disposition pose d'ores et déjà le problème de son utilisation et de son adaptation à d'autres contextes

de travail, ou d'autres configurations matérielles (tout le monde ne dispose pas des périphériques que nous avons employés). Un autre aspect important est aussi la possibilité d'adapter le système aux goûts et habitudes de l'utilisateur.

4.3.5 Configurabilité et adaptabilité du système

Avant même son évaluation, un des aspects importants de notre approche permet d'anticiper sur les futurs besoins qu'elle entraînera : la *configurabilité* et l'*adaptabilité*. Ces deux notions que nous allons aborder maintenant assurent que notre système pourra être rapidement adapté et raffiné dans l'optique d'une meilleure interaction avec l'utilisateur. Nous ne rentrerons pas dans les détails techniques de la réalisation, ceux-ci feront l'objet de la partie II page 147 de ce mémoire, et tout particulièrement du chapitre 7 page 157 et de la section 8.4.3 page 203. Nous ne détaillerons ici que les impacts de ces apports pour l'utilisateur de SVALABARD.

Les boîtes à outils que nous avons utilisées (ICON) ou développées (MAGGLITE) pour concevoir ce système permettent un contrôle total sur la configuration en entrée de l'interface (utilisation et affectation de périphériques d'entrées aux outils et fonctions de l'application) et sur ses comportements (apparition des feuilles d'interaction, par exemple). Cette configuration est interactive, en utilisant le langage graphique proposé dans ICON [Dragicevic, 2004b].

Nous pouvons, par exemple, proposer en quelques minutes une configuration permettant d'utiliser SVALABARD avec uniquement des périphériques standard : une souris et un clavier. Le bouton gauche de la souris contrôle l'outil de dessin, le bouton du milieu le marking menu et le bouton droit la saisie des contraintes. La molette de la souris peut permettre de changer de calque de dessin actif. Ensuite, l'utilisation conjointe d'une touche du clavier (« control », par exemple) et du bouton gauche de la souris permet de tracer des annotations. Enfin, toutes les autres actions (couleur de dessin, gestion des calques, vue 3D) peuvent être affectées à des touches du clavier. Certes, une telle utilisation du système briserait la cohérence et les métaphores que nous avons introduites. Par contre, elle prouve l'adaptabilité du système à des entrées appauvries, permettant son utilisation dans des contextes différents : sur un ordinateur portable, ou un TabletPC dans le cadre de l'informatique mobile, par exemple.

Mais, de la même manière le système peut être reconfiguré pour utiliser des entrées enrichies (autres périphériques ou modalités d'entrée). Un exemple serait d'introduire de la reconnaissance de la parole pour les fonctions de commande du système. Par exemple, les fonctionnalités du marking menus pourrait être données sous forme d'ordre. Mais nous avons aussi expérimenté avec succès la saisie multimodale des propriétés géométriques, en combinant gestes et parole. Nous avons gardé le principe du geste pour sélectionner les objets d'intérêt (segments à spécifier parallèles, par exemple) et transféré le choix de la propriété à appliquer sur un ordre parlé. Cette interaction, reprenant le principe éprouvé du « Put That There » de Richard A. BOLT [Bolt, 1980], présente l'avantage de ne plus nécessiter qu'un seul geste, pour la sélection. Bien qu'utilisée dans un cadre favorable (association du pointage et de l'ordre, peu de commandes), elle demande toutefois plus de contrôle au niveau de la saisie afin de lever des ambiguïtés et d'éviter des erreurs plus fréquentes : commande inapliquable à un objet, séquence de commandes incohérentes, etc.

Cette configurabilité est aussi importante dans le cadre de l'adaptation du système à l'utilisateur. En effet, nous avons proposé des associations de périphériques proposant souvent une interaction bimanuelle, ou des affectations de fonctions sur un périphérique dans un ordre logique. Il est possible pour l'utilisateur de changer l'affectation de ces fonctions des périphériques aux outils du système se-

lon ses préférences ou capacités (changer l'ordre des boutons sur le shuttle, ou sur la boîte à glissières, par exemple).
De la même manière, il peut lui même régler le comportement des différents composant graphiques de l'interface comme il l'entend. Nous avons automatisé l'apparition de la feuille augmentée lorsque l'utilisateur ne dessine plus. Il est possible de redéfinir ce comportement pour la faire apparaître sur pression d'un bouton, par exemple. Et d'autres comportements pourront même être imaginés par les utilisateurs : par exemple, le filtre de détection de la phase de dessin que nous proposons dans la section 4.4.4 page 114 peut être utilisé pour changer automatiquement la couleur du trait de dessin ou le calque de dessin actif lors d'un changement de phase.

Nous avons donc présenté la partie « visible » de SVALABARD : son interface, ses interactions et son utilisation. Cette approche revêt dans son ensemble un aspect unique de par l'intégration que nous avons voulue cohérente de beaucoup de principes inspirés des habitudes des concepteurs (du contrôle de l'outil de dessin par le stylet à l'interaction bimanuelle, en passant par le pliage des calques). Mais bien plus que cela, la notion de *feuilles d'interaction* permet une séparation claire, et pourtant non abrupte, des différentes fonctionnalités, tout en unifiant dans un même outil, par un paradigme d'interaction commun, les trois fonctionnalités du dessin : le spéculatif, le prescriptif et le descriptif.

4.4 Analyse automatique du dessin

Comme nous l'avons exposé dans la section 3.4.4 page 79, le système doit interpréter les données reçues de l'utilisateur pour composer son modèle interne du dessin, utilisable en entrée par un noyau fonctionnel. Le dessin produit est assimilé à des flux continus de points, les traits. Ces traits vont être utilisés par les filtres d'analyse de SVALABARD pour structurer et nettoyer le dessin afin de produire une structure exploitable par un système de reconstruction 3D. La structuration du dessin revient à transformer les traits du dessin en objets de plus haut niveau (points et segments) et à en identifier les relations (extrémités de segments, segments concourants, etc.). Nettoyer le dessin consiste à simplifier les données et supprimer celles qui sont inutiles au noyau fonctionnel (élimination des traits des phases non constructives, fusion de points, etc.). Le résultat en est une représentation *vectorielle* (voir figure 4.23 page ci-contre).

SVALABARD n'inclut pas de système d'interprétation de courbes (*Bézier* ou *NURBS*). Bien qu'il soit possible de dessiner tout type de traits et de surfaces, ceux-ci seront toujours représentés sous la forme de points et de segments. Rien ne garantit alors dans ce cas la justesse et la précision de l'interprétation ni, surtout, la possibilité d'obtenir une reconstruction 3D. Cette restriction est due aux limites actuelles du noyau mathématique de GINA qui ne permet que de reconstruire des objets polyédriques. Nous discuterons toutefois dans le chapitre suivant les possibilités d'évolution globales sur ce point.

Nous présentons dans cette section les principes et algorithmes que nous proposons, ainsi qu'un filtre de *détection du contexte*. Cet algorithme permet de filtrer les traits des différentes phases que nous avons proposées dans la section 3.3.3 page 70, et peut donc être utilisé dans un but de filtrage du dessin, mais aussi pour adapter l'interface à la phase courante de dessin (affichage de résultats ou propositions, démarrage de traitements, etc.).

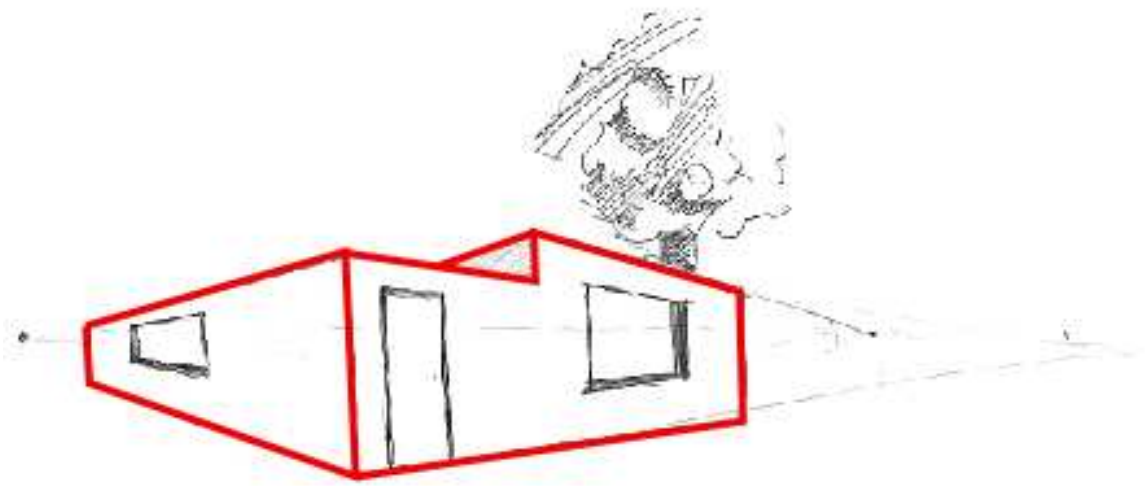


FIGURE 4.23 – Interprétation d'un dessin au trait. En rouge, l'interprétation *vectorielle* attendue d'un dessin pour sa reconstruction en 3D.

4.4.1 Organisation des filtres de dessin

Beaucoup de travaux on été réalisés dans le domaine de l'interprétation de dessins (et plus généralement d'images). Nos travaux se situent dans un cadre bien particulier : l'interprétation *temps réel*, ou *interactive*. En effet, l'approche à adopter pour interpréter un dessin dont on connaît la chronologie n'est pas la même que pour une image terminée, sans indication temporelle (un atout indéniable pour l'extraction des propriétés du dessin). Généralement, la démarche interactive est la suivante :

1. **Capture des traits** : reconnaître les différents points qui composent un trait.
2. **Caractérisation des traits** : pour chaque trait, déterminer la (ou les) caractéristique(s) géométrique(s) qu'il représente (segment, courbe, etc.).
3. **Connection des extrémités** : réduire l'imprécision du dessin à main levée en détectant les entités concourantes et connectant leurs extrémités.

Cette démarche générale est bien adaptée au dessin à main levé mais tout de même dans un cadre particulier : celui où le dessinateur trace des traits continus, d'une manière plus ou moins contrainte. Or, nous inscrivant dans un paradigme de dessin libre pour la conception, nous avons du compléter cette approche à plusieurs niveaux : l'élimination des traits inutiles au noyau fonctionnel (les phases non constructives), et l'assimilation et le nettoyage des traits de révision (démarche itérative).

Dès lors, nous avons inséré deux traitements supplémentaires, qui sont **la détection du contexte** de dessin et **la fusion de segments**. La détection du contexte, à partir des caractéristiques des traits et des segments construits, va permettre de ne conserver que les données des phases constructives. La fusion des segments va permettre de regrouper en un seul segment des traits repassés, révisés, rallongés. La figure 4.24 page suivante illustre donc la chaîne de traitement que nous proposons, ponctuées par le module de reconstruction 3D.

Les traitements sont séquentiels et suivent un paradigme de flot de données. Chaque filtre est un module réactif, qui reçoit des données en entrée et transmet au module suivant les données reçues

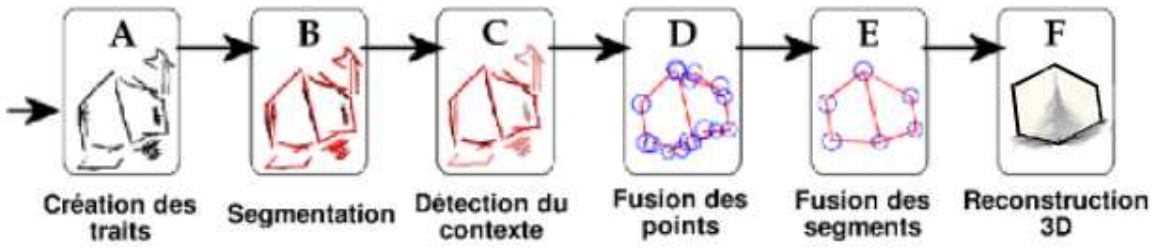


FIGURE 4.24 – Filtres séquentiels de dessin.

et les traitements produits. Nous parlons de *traitements en cascade*. Cet ordre obéit à la logique de traitement des traits et à une structuration progressive du dessin. Il est toutefois possible d'insérer simplement de nouveaux traitements dans la séquence, ou d'en changer l'ordre (en respectant toutefois leurs entrées-sorties), grâce à la modularité et la flexibilité des outils logiciels utilisées pour l'implémentation (voir les boîtes à outils ICON [Dragicevic et Fekete, 1999; Dragicevic et Fekete, 2004; Dragicevic, 2004b] et MAGGLITE dans la partie II page 147 de ce mémoire). Cette modularité est une partie de l'originalité de notre approche du traitement du dessin, contrairement à nombre d'approche ou il est ardu de modifier la séquence de traitement.

En outre, cette architecture permet l'intégration des traitements dans la description des interactions et des comportements du système, décrits dans le même paradigme de flot de données (voir le détail dans le chapitre 7 et la section 8.4.3 page 203). Ainsi, les filtres sont appliqués en temps réel et intégrés aux interactions du système. Cela confère une plus grande adaptabilité aux données reçues contrairement aux méthodes dites *paresseuses* qui repoussent les traitements en fin de saisie.

4.4.2 Création des traits

Nous définissons un trait comme une collection de points tracés sur la zone de dessin (trait sur une feuille). Leur saisie est initiée par une action (pression sur un stylet ou sur un bouton de périphérique), effectuée en déplaçant le pointeur et terminée par une autre action (relâchement du stylet ou du bouton), comme le montre la figure 4.25 page suivante. Ce mode de saisie est identique au tracé d'un trait sur une feuille.

Le premier filtre (cadre A de la figure 4.24) permet d'enregistrer les traits, collections de points, ainsi produits. Cet algorithme simple construit un trait en enregistrant les points qui suivent le déplacement de l'outil de dessin est utilisé. Pour chaque point, le système enregistre la position 2D, le temps et la pression (si le dispositif d'entrée le permet). L'algorithme effectue aussi un filtrage des points saisis, afin d'éliminer des points identiques ou trop proches dans un même trait. En effet, la fréquence d'échantillonnage(5) de plus en plus élevée des périphériques d'entrée actuels entraîne la saisie de plusieurs points identiques, ou de points très proches qui ne seront ni significatifs visuellement, ni pour les algorithmes de traitement. Éliminer les points trop proche ne change donc pas l'aspect visuel des traits dessinés, et ne fausse pas les algorithmes de traitement (cela les rend même plus performants

(5)Nous regroupons sous le terme « fréquence d'échantillonnage » deux caractéristiques des périphériques d'entrée : leur résolution temporelle, c'est à dire le nombre de mises à jour de ses données en une seconde, mais aussi leur résolution spatiale, c'est à dire le seuil limite de déplacement qu'ils détectent. À titre d'exemple, les tablettes écran produites par Wacom atteignent des résolutions de 0.05mm par point, pour des échantillonnages de 205 points par seconde.

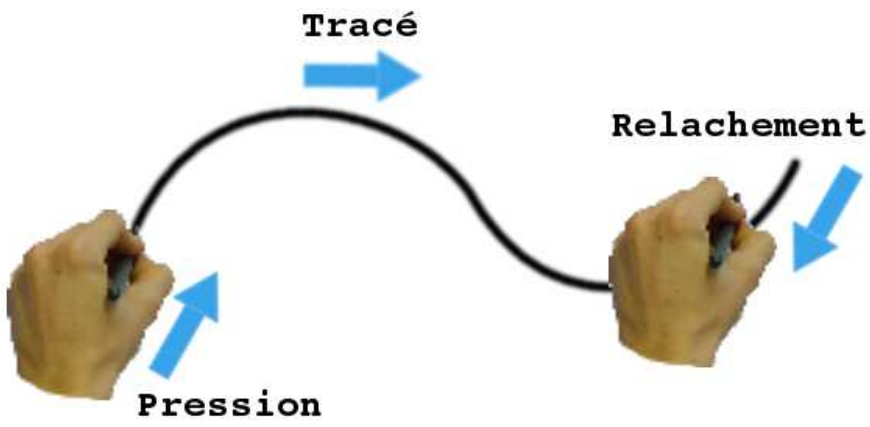


FIGURE 4.25 – Dessin d'un trait.

en terme de rapidité, sans perte de qualité de traitement). Ainsi, l'algorithme de création des traits propose un seuil réglable de filtrage, basé sur la distance entre deux points consécutifs.

Ce module de traitement réalise aussi un autre filtrage : celui des traits de construction. Nous avons constaté dans le chapitre 3 que les traits de construction n'étaient pas nombreux sur les dessins des sujets de notre étude. Toutefois, ils ne sont pas inexistants. Il est important de les détecter, car n'étant pas nécessaires au noyau fonctionnel du système, il n'est pas utile de leur appliquer tous les traitements. Ils doivent toutefois être propagés dans la séquence de filtrage, étant utilisés par l'algorithme de détection du contexte.

Cette détection est réalisée en regard de nos observations : ce sont de simples points (ou des traits de taille très réduite), ou des traits dont la pression est très faible. Ainsi, le filtre de création des traits permet de marquer les points seuls ainsi que les traits d'une taille ou d'une pression moyenne inférieures à des seuils. Nous avons réglés ces seuils de filtrage de manière expérimentale, à partir de notre propre expérience et des tests informels que nous avons réalisé avec quelques dessinateurs. Toutefois, ceux-ci sont facilement paramétrables et peuvent être adaptés à l'utilisateur.

4.4.3 Segmentation des traits

Afin de structurer le dessin et de l'adapter aux futurs traitements, il faut simplifier et décomposer les traits en un ou plusieurs segments formés uniquement de deux points (cadre B de la figure 4.24 page précédente). Le principe est d'éliminer tous les points « inutiles » situés entre deux points critiques (appelés coins) qui formeront les extrémités des segments (voir figure 4.26 page suivante).

On trouve beaucoup de ces méthodes de segmentation de traits, appelées aussi détection de coins, dans la littérature (vingt et une d'entre elles sont recensées et comparées dans [Rosin, 1997]), adaptées à différents cas d'utilisation (traitement d'images numérisées, traitement de vidéos, dessin temps réel, etc.). Nous avons utilisé et adapté la méthode proposée par Tevfik SEZGIN [Sezgin *et al.*, 2001], conçue pour le dessin en temps réel, qui combine les informations de vitesse et de courbure pour

FIGURE 4.26 – Segmentation de traits. Un trait et son approximation

extraire les points critiques. La prise en compte de ces deux caractéristiques confère à cette méthode plus d'efficacité que celles basées uniquement sur l'une des deux, sans pour autant en diminuer la performance.
L'algorithme calcule la courbure et la vitesse en chaque point du trait. Une première approximation est composée à partir des deux points extrêmes du trait et du point de plus petite vitesse et de plus grande courbure. Ensuite, deux segmentations de plus en plus précises sont composées en ajoutant le point de plus petite vitesse pour l'une et celui de plus grande courbure pour l'autre. La meilleure des deux solutions, en terme de distance au trait original, est conservée et l'algorithme itère ainsi jusqu'à ce qu'un taux de précision fixé soit atteint.

Ce filtre reçoit donc un trait en entrée et produit en sortie un ou plusieurs segments, approximation du trait dessiné (il ne fait rien si le trait avait été marqué comme trait de construction). Il reste tout de même à éliminer certains segments ainsi construits, de façon à épurer le dessin pour ne garder que les données utiles à la reconstruction 3D.

4.4.4 Détection du contexte

Nous avons déterminé, dans la section 3.3.3 page 72, trois phases dans la tâche de dessin en perspective pour l'architecture. Nous proposons un traitement du dessin en temps réel pour la détection automatique de ces phases. Il y a deux objectifs à caractériser ces phases :

1. *Traitement du dessin*. Dans l'optique d'obtenir une reconstruction 3D à partir des tracés de l'utilisateur, il n'est pas utiles de fournir au noyau de reconstruction les traits produits lors des phases de *complétion* et de *style*[(6)].
2. *Adaptation des interactions et de l'interface*. Un problème récurent des interfaces de dessin au stylet est une certaine incompatibilité de cette modalité avec les commandes « traditionnelles » des interfaces WIMP. Ainsi, devoir presser un bouton, ou changer d'outil, de visualisation en utilisant le stylet de dessin dans le mode de dessin peut provoquer des ruptures dans la tâche créative. Parallèlement, cela oblige aussi à contrôler à la fois la tâche et le fonctionnement du programme (modes), comme souligné dans [Nielsen, 1993]. Ce filtre de détection du contexte est une solution possible à ces problèmes, pour adapter l'interface et les outils d'interaction à la phase courante du dessin, c'est à dire opérer à des changements de mode implicites

[(6)]Notons sur ce point que ce traitement de *filtrage* des traits est le seul de la chaîne qui dépend de la finalité du dessin (modélisation 3D pour l'architecture). Nous montrerons dans le chapitre 5 que notre démarche reste générale dans son ensemble et que c'est ce point précis (étude et filtrage selon le domaine) qui doit être particulièrement adapté à la finalité de la tâche.

dans l'esprit du protocole introduit dans [Saund et Lank, 2003]

Ce filtre (cadre C de la figure 4.24 page 112) identifie automatiquement le contexte (ou phase de dessin) à partir des segments reçus, de la phase courante et des données collectées dans l'évaluation présentée dans la section 3.3 page 58. Ces phases, au nombre de trois et que nous avons nommées *construction, amélioration et style*, sont identifiées par un algorithme basé sur les résultats de notre étude et des heuristiques simples.

À la réception du premier trait d'un dessin, l'algorithme initialise le système en contexte « constructif » (un dessin débute toujours par cette phase). Ensuite, chaque nouveau trait est analysé à sa réception de manière à évaluer la possibilité de passer dans l'une des trois phases à partir de la phase courante. Cette évaluation repose sur le calcul d'un score pour chaque phase potentielle. Ces scores sont tout d'abord initialisés à la probabilité de transition depuis la phase courante vers celles qu'ils représentent. Ces probabilités ont été calculées d'après les données de notre étude (table 4.2).

Phase	*Construction*	*Amélioration*	*Style*
Construction	95	3,5	1,5
Amélioration	2,5	96	1,5
Style	4	4	92

TABLEAU 4.2 – Probabilité des transitions entre contextes (%).

Ensuite, les scores sont pondérés par l'analyse des propriétés du trait reçu, du contexte courant et des propriétés du dessin. Tout d'abord, nous avons constaté qu'il n'y avait pas de gribouillage en phase constructive dans les dessins de notre étude. Ces traits sont donc détectés et le score des phases d'amélioration et de styles sont pondérées en conséquence. Ensuite, si le trait reçu est marqué comme trait de construction (par le premier filtre), le score de la phase de construction est augmenté.

L'algorithme prend alors en compte la position du trait reçu par rapports aux traits précédemment dessinés en phase de construction. Appelons « zone constructive » la boîte englobante des traits de la phase de construction qui ont déjà été dessiné. Un trait recoupant ou « à l'intérieur » de cette zone constructive sera d'une phase constructive ou d'amélioration (de par la définition même et la caractérisation de ces phases). Cela induit une augmentation des scores des phases de construction et d'amélioration.

Les différentes pondérations appliquées aux scores de bases des phases de dessin potentielles lors du tracé d'un nouveau trait sont représentées dans le tableau 4.3. Ces valeurs ont étés déterminées par le calcul des pourcentages de tels traits dans les différentes phases des dessins de notre étude. Nous les avons ensuite corrigées pour obtenir un comportement cohérent lors de l'application de l'algorithme sur les dessins de l'étude.

Enfin, l'algorithme analyse aussi les temps de pauses entre les traits afin d'appliquer les résultats que nous avons introduit sur l'augmentation des temps de pause lors d'un changement de phase. Le tableau 4.4 représente les pondérations des scores des différentes phases en fonction de la phase courante, pour des temps de pause considérés comme « longs ». Ces valeurs sont proportionnelles aux moyennes calculées lors de notre étude, et de plus pondérées par le temps de pause. Bien sur, la notion de « long temps de pause » est subjective et ce seuil peut être configuré dans l'algorithme. Toutefois, les résultats de notre étude avaient mis en évidence des temps de pause variant de une à deux secondes lors des changements de phases. C'est pourquoi nous avons fixé ce seuil à deux secondes, par défaut.

Trait courant	*Score de construction*	*Score d'amélioration*	*Score de style*
Pression$< 0,1$	$+80$		
Longueur< 20	$+70$		
Trait dans la zone « constructive »	$+100$	$+100$	
Gribouillage		$+100$	$+100$

TABLEAU 4.3 – Pondération des scores selon le trait courant.

Contexte courant	*Score de construction*	*Score d'amélioration*	*Score de style*
Construction		$+80$	$+15$
Amélioration	$+75$		$+25$
Style	$+50$	$+50$	

TABLEAU 4.4 – Pondération des scores selon les temps de pause.

Nous avons fixé ces différentes valeurs de pondération des scores de manière expérimentale, par l'analyse des dessins recueillis lors de notre étude. En appliquant l'algorithme sur ces dessins enregistrés, ainsi que sur de nouveaux dessins, nous avons pu ajuster ces valeurs expérimentales et ces heuristiques. Les résultats sont satisfaisants sur ces dessins exemples et sur de nouvelles réalisations en temps réel, mais nous n'avons toutefois pas encore évalué ses performances (en terme de « taux de réussite ») dans des conditions réelles d'utilisation.

Notons enfin que cette méthode tire son efficacité de la combinaison de ces différentes heuristiques. En effet, l'utilisation conjointe des caractéristiques moyennes des traits et des phases de dessin (issues de l'étude préliminaire), ainsi que de quelques caractéristiques du dessinateur (temps de pause) permet de déterminer la phase courante. Il demeure toutefois des erreurs pouvant se révéler être une limitation évidente du système, occasionnant alors des interprétations erronées du dessin ainsi que des déroutes éventuelles de l'utilisateur. C'est pourquoi cette proposition d'algorithme reste un prototype, une preuve de faisabilité de la méthode, qui ne sera pas utilisée en l'état dans une évaluation globale du système. Il sera important, pour fiabiliser cette approche, d'étudier l'introduction de techniques de personnalisation et d'adaptation de l'algorithme à l'utilisateur et à son style de dessin (paramètres personnalisés et apprentissage).

4.4.5 Fusion des extrémités de segments

Lors de dessin a main levé avec des entrées discrètes (tablette graphique, souris, etc.), les extrémités des traits ayant un point commun coïncident rarement. Le système doit donc, afin de construire et affiner la structure du dessin après l'opération de segmentation, détecter et regrouper ces points (cadre D de la figure 4.24 page 112). Deux approches peuvent êtres mises en pratique pour réaliser ce traitement. Une première méthode est de contraindre la fusion de points concourants dès la saisie du dessin, c'est à dire lors du tracé des traits. Cette technique, dite de *magnétisme* ou *snapping*, contraint à débuter un trait à partir d'un point du dessin existant dès lors que le pointeur de l'outil est proche de ce point existant (à une distance inférieure à un seuil fixe). Cette technique d'interaction est très utilisée en dessin vectoriel, mais est inadaptée à notre approche de dessin libre à main levée car par

trop contraignante. La seconde méthode consiste à appliquer le même principe de magnétisme entre les extrémités des segments du dessin, mais cette fois après leur tracé. Ainsi, lorsque le dessin est terminé et la segmentation des traits effectuée, les points dont la distance est inférieure à un certain seuil sont fusionnés. Cette méthode est déjà plus en accord avec les principes de notre système mais pose toutefois deux problèmes :

- d'un point de vue technique, tout d'abord, le principe d'utiliser un seuil fixe pour la fusion des points peut entraîner des erreurs d'interprétation car cela ne prend pas en compte la précision du dessin et ses particularités locales.
- pour ce qui est du fonctionnement de notre outil, nous proposons que les traitements et interprétations soient réalisées en temps réel, pendant le dessin. Même si cela soulève des problèmes, notamment pour la présentation interactive des résultats, l'interprétation en temps réel du dessin permet de tout de suite tirer partie des valeurs ajoutées du système informatique dans un contexte créatif.

Nous proposons donc pour la fusion des extrémités des segments, une adaptation au traitement temps réel de la méthode de Moshe SHPITALNI [Shpitalni et Lipson, 1997]. Cette méthode est une amélioration de la fusion des points par seuil fixe. Le principe est de calculer, pour chaque extrémité de segment, une zone de tolérance dont la taille correspond à l'incertitude sur la position du point. Cette zone est un cercle dont le rayon est fonction de la distance entre le point et tous les segments du dessin. Des extrémités doivent être fusionnées lorsque chacune est située dans la zone de tolérance de l'autre (voir figure 4.27). Le barycentre du groupe de points remplace donc les points et les segments sont mis à jour (voir l'algorithme A.3 en annexe A page 3).

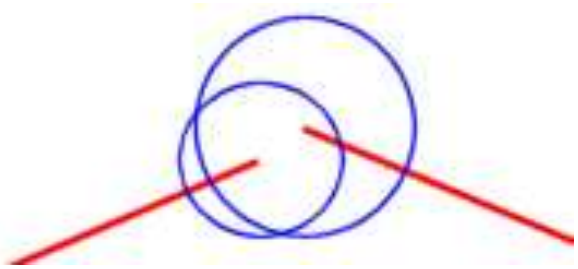


FIGURE 4.27 – Fusion de points. Deux extrémités à fusionner.

L'efficacité de cette méthode est due en grande partie aux zones de tolérance adaptables, qui prennent en compte la densité du voisinage de chaque point, et donc les aspects locaux du dessin. Originalement conçue pour un dessin terminé, notre adaptation pour le traitement temps réel se situe au niveau du calcul des zones, pondéré par la résolution (taille globale) du dessin, et donc évoluant au cours du temps. De plus, l'algorithme ignore les cas extrêmes (zones maximales) en début de dessin, et nécessite un nombre minimal de segments avant de regrouper des points pour ne pas occasionner de fusions indésirables.

4.4.6 Fusion de segments

Nous avons déjà souvent exposé le principe des révisions dans le croquis de conception. Cela se traduit, en terme d'analyse de ce dessin, par des cas où le dessinateur a tracé plusieurs traits pour

ne former qu'un segment (comme nous l'avions déjà montré avec le dessin de conception de Franck O. GEHRY, figure 3.13 page 80). Un trait peut en prolonger un autre, mais la plupart du temps les traits se recoupent ou se chevauchent (les traits « principaux secondaires » de notre taxinomie). Nous proposons donc un algorithme original permettant de fusionner de tels segments pour composer et nettoyer la représentation structurelle d'un dessin à main levée (cadre E de la figure 4.24 page 112).

Nous considérons deux types de critères à retenir pour analyser de telles situations et donc effectuer la fusion de segments : les critères positionnels et les critères relationnels. Nous considérerons dans la suite deux segments $S1$ et $S2$ pour lesquels nous voulons analyser les possibilités de fusion.

Critères positionnels

Le premier critère positionnel est la position globale des segments dans le dessin. Ainsi, la zone de dessin est divisée en zones rectangulaires égales (dont le nombre varie selon la taille de la feuille de dessin, voir figure 4.28). Les deux segments $S1$ et $S2$ doivent donc appartenir partiellement ou entièrement à une même zone pour être candidats à la fusion.

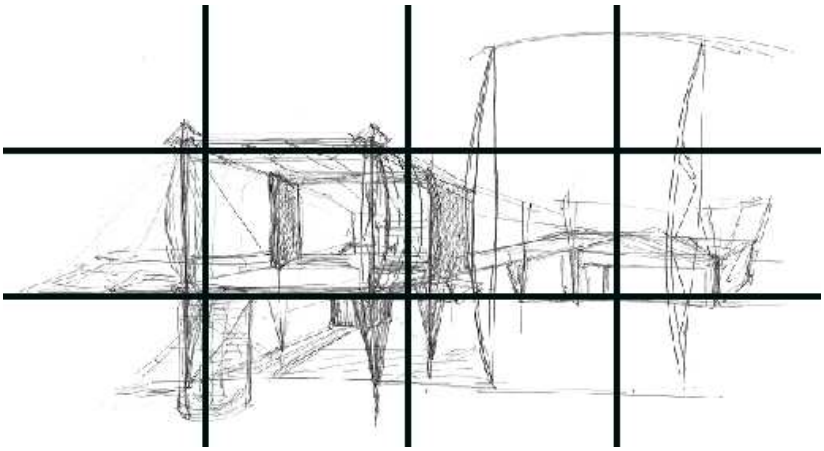
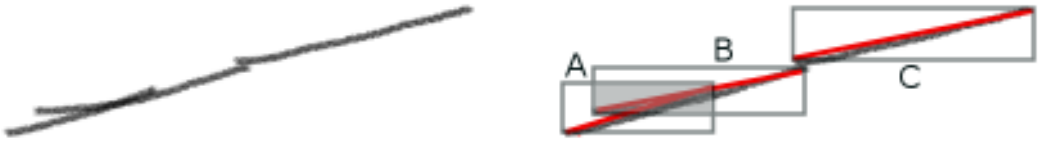


FIGURE 4.28 – Fusion de segments. Division de l'espace de dessin en zones.

Dans le cas ou les segments sont dans la même zone de dessin, l'algorithme procède alors au calcul de l'intersection de leurs boîtes englobantes (voir figure 4.29(b) page suivante). Si l'aire de cette intersection est inférieure à un certain seuil, les segments sont potentiellement à fusionner. Dans le cas contraire, la distance entre les segments tout de même prise en compte pour déterminer un éventuel prolongement (voir figure 4.29 page ci-contre). Précisons que nous définissons la distance entre deux segments par la plus petite des quatre distances de leurs extrémités respectives à l'autre segment. Un seuil de distance entre deux segment est fixé pour lequel les segments sont candidats à la fusion. Nous proposons des seuils déterminés de manière expérimentale mais paramétrables.

Critère relationnel

Dès lors que deux segments remplissent les critères positionnels que nous avons exposés, il reste un critère relationnel à examiner : l'angle formé par les deux segments. Si cet angle est en dessous



(a) Traits originaux. (b) Segments à fusionner.

FIGURE 4.29 – Cas d'une fusion de segments. Les segments de la figure (b) sont analysés. Les segments A et B seront fusionnés car leurs boîtes englobantes ont une intersection conséquente. Le segment résultant sera fusionné au segment C, même si leurs boites englobantes ne s'intersectent pas (leur distance est faible).

d'un certain seuil, les segments remplissent alors une condition supplémentaire.

Il faut tout de même considérer le cas particulier des segments concourants, tout spécialement lors de dessin en perspective :

- Si deux segments ont une extrémité commune, alors le seuil doit être plus bas pour éviter de fusionner des segments formant un angle écrasé par la projection perspective (segments AB et BC de la figure 4.30).
- Si 3 segments (ou plus) ont un point commun (sommet B de la figure 4.30), le seuil d'angle à considérer est encore plus petit. Ce cas évite la fusion de segments connectés par un point caractéristique du dessin perspective (jonction en T ou partie saillante d'un volume projeté). Si toutefois les segments sont fusionnés dans ce cas, la propriété d'incidence du point aux segments fusionnés est conservée.

Ces différents seuils peuvent être considérés comme différents niveaux de tolérance de la fusion de segments pour les caractéristiques du dessin en perspective.

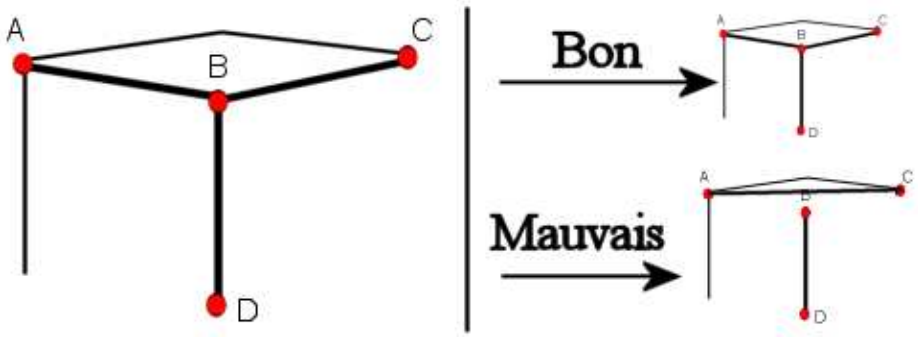
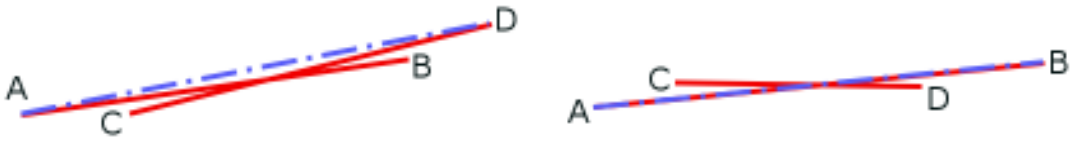


FIGURE 4.30 – Fusion de segments : cas de segments concourants.

Si les deux segments $S1$ et $S2$ remplissent tous ces critères ils sont alors fusionnés en un seul, selon un principe simple : le segment résultant sera le plus long segment composé par combinaison des extrémités des segments originaux. Des quatre extrémités, les plus « à gauche » et « à droite » composent un premier segment, les plus « en bas » et « en haut » un autre. Le plus long des deux est conservé. Comme le montre la figure 4.31 page suivante, nous parlons alors de *composition* lorsque le seg-

ment résultant est composé à partir de deux extrémités issues de segments différents et d'*assimilation* lorsque l'un des segments est conservé.



(a) Composition. (b) Assimilation.

FIGURE 4.31 – Fusion de deux segments. Les deux cas pouvant survenir lors de la fusion effective des segments sont *la composition*, figure (a) et *l'assimilation*, figure (b).

Nous avons vu le cas de l'étude de deux segments pour la fusion. L'algorithme global (voir l'algorithme A.4 dans l'annexe A page 3) opère sur l'ensemble des segments résultants des traitements précédents. Ils les compare deux à deux et réalise les éventuelles fusion jusqu'à atteindre un point fixe sur l'ensemble des segments (plus de fusions) ou jusqu'à ce qu'un nouveau segment soit ajouté.

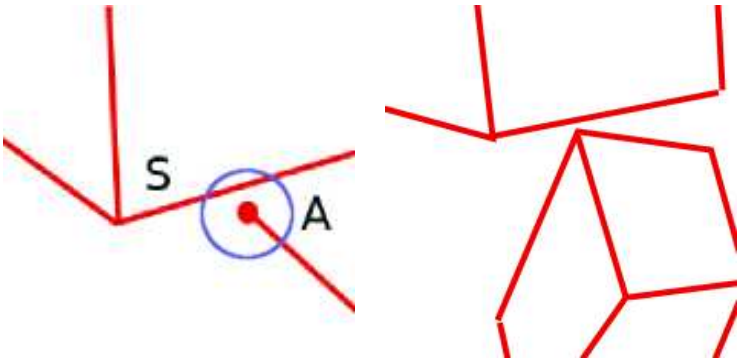
Ce filtre est utilisé ici dans un but de « nettoyage » du dessin, sans retour direct pour l'utilisateur (nous entendons ici sans être visible sur la feuille de dessin, mais les résultats des fusions sont toutefois visibles sur la feuille augmentée). Mais son comportement pourrait être raffiné, notamment au niveau de la détermination du segment résultant, afin de fournir des interactions avancées et novatrices dans un système de dessin vectoriel (prolongement ou déplacement naturel de segments, dans l'esprit de l'interaction pour la modification de courbes complexes proposée par Thomas BAUDEL dans « Ligne Claire » [Baudel, 1994]).

4.4.7 Compréhension du dessin pour et par le noyau

Au fur et à mesure du dessin, cette cascade de filtres en temps réel compose donc une structure géométrique du dessin. Cette structure forme les premières données transmissibles au noyau de reconstruction, représenté comme le dernier élément de la chaîne dans la figure 4.24 page 112. À l'heure actuelle, les informations que contient cette représentation issue de nos filtres de traitement sont essentiellement de bas niveau :

1. points et segments composant l'enveloppe du modèle à reconstruire,
2. segments concourants (déterminés implicitement par la fusion des points),
3. points incidents à des segments.

Nous n'avons pas détaillé le dernier point concernant l'incidence de points à des segments. Cette détection de points situés sur un segment est encore en cours de développement. Elle se situe au niveau de l'algorithme de fusion des points. En effet, nous pensons que les zones d'incertitudes utilisées pour déterminer la fusion de points peuvent aussi être utilisées pour déterminer si un point n'est pas situé sur un segment (voir figure 4.32(a)). Il reste toutefois à régler certains problèmes avec cette méthode, notamment la prise en compte de la projection perspective qui ne garantis pas que le point, même si il parait appartenir au segment sur la projection en deux dimensions, est sur le segment en trois dimensions (voir figure 4.32(b)).



(a) (b)

FIGURE 4.32 – Incidence point/segment. La figure (a) suggère la question de l'appartenance du point A au segment S (le segment S coupe la zone d'incertitude du point). La figure (b) soulève une incertitude dès lors que l'on a une vision plus globale du dessin.

Ces données sont toutefois insuffisantes pour permettre au noyau de reconstruire un modèle 3D, celui-ci nécessitant des relations de plus haut niveau entre les éléments du dessin : parallélismes, orthogonalités et coplanarités. Bien que les outils interactifs que nous avons proposés pour saisir ces données soient à notre avis plus efficaces que des boîtes de dialogue ou un langage de script, cette démarche n'en reste pas moins contraignante pour décrire les relations existantes sur un dessin complexe, provoquant alors une rupture dans la démarche de conception. Il est alors indispensable d'inclure un mécanisme de détection de telles contraintes dans la séquence de filtrage/interprétation du dessin, avant le module de reconstruction 3D. Nous proposerons plusieurs approches possibles dans ce but dans le chapitre suivant. Il est toutefois important de souligner que la flexibilité qu'offre l'architecture modulaire des filtres de traitement du dessin que nous proposons permet d'en ajouter, ou d'en remplacer, mais aussi d'en prototyper de nouveaux sans avoir à modifier tout le processus de traitements.

4.4.8 Calques et interprétation du dessin

Enfin, il est nécessaire de préciser le comportement de ces modules d'interprétation lors de l'utilisation de plusieurs calques de dessin. Du point de vue de l'utilisateur, les calques peuvent être utilisés pour structurer son dessin (l'utilisation habituelle avec les logiciels de conception) ou pour effectuer des révisions sur des points déjà spécifiés (l'utilisation plus habituelle à la conception architecturale). Du point de vue de l'interprétation automatique du dessin, ces deux modes d'utilisation impliquent un même principe : une fusion des interprétations issues des différents calques composant le dessin. Il est alors envisageable de réaliser cette fusion de deux manières possibles :

1. une fusion *a priori*, où les traits tracés sur chaque calque sont interprétés par une même cascade de filtres (voir figure 4.33(a) page 123).
2. une fusion *a posteriori*, où chaque calque est connecté à une cascade de filtres différente, la fusion s'opérant sur les interprétations (voir figure 4.33(b)).

Actuellement, le système fusionne *a priori* les parties du dessin tracées sur des calques différents. Dès lors, il n'y a qu'une seule interprétation du dessin, tous les calques sont « connectés » à une unique séquence de filtres. Cette approche soulève le problème de l'éventuel retrait d'un calque : les interprétations issues de ce calque ont été incorporées à l'interprétation du dessin et ne peuvent plus en être retirées. Cela entraîne alors une incohérence entre le dessin et son interprétation : les données interprétées persistent, même si le calque a été retiré.

Nous envisageons, pour remédier à cette incohérence, de permettre l'interprétation indépendante de chaque calque, ainsi, la fusion des interprétation s'effectuera *a posteriori*, en aval des interprétations, juste avant la reconstruction 3D (voir figure 4.33(b) page suivante). Il pourra ainsi y avoir une historique des interprétations, celles-ci étant construites à plusieurs niveaux. Cette démarche pose toutefois le problème de déterminer le niveau de dépendance des traits de chaque calque : les traits tracés sur un calque sont-ils nécessaires à l'interprétation de ceux d'un autre calque ? Nous sommes d'avis que cette interdépendance est forte, même si nous n'avons pas encore fouillé plus avant la question. Dans ce cas là, notre approche constituée d'une seule chaîne d'interpréteurs est la plus efficace, traitant le dessin d'emblée dans sa globalité, même si moins souple en terme de retour en arrière (suppression d'un calque).

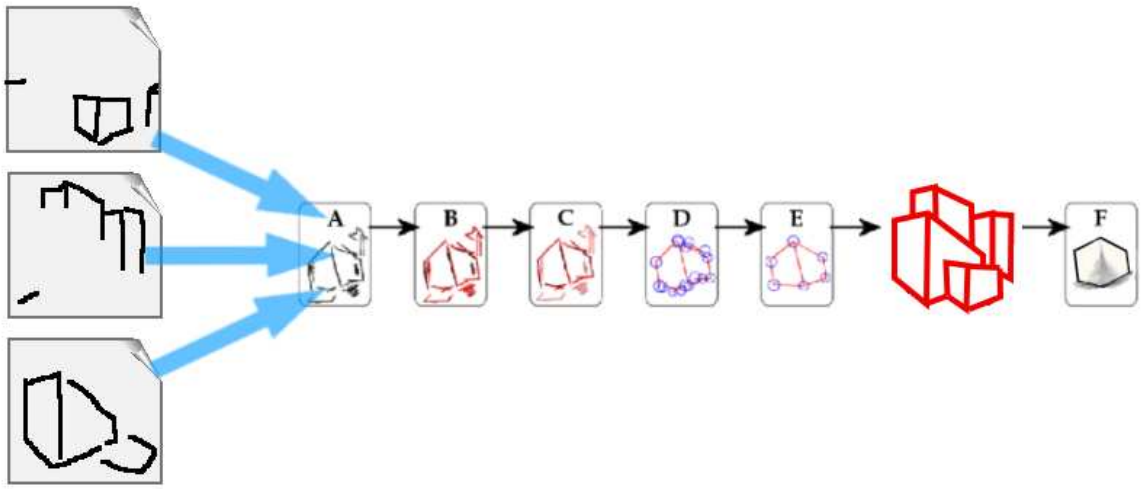
4.5 Conclusion

Nous avons présenté dans ce chapitre le système SVALABARD, un outil de dessin pour la modélisation 3D dans un contexte de conception créative. Sa réalisation met en œuvre les lignes directrices que nous avions proposé dans le chapitre précédent. Ainsi, le mode de communication choisi entre l'homme et la machine est le **dessin libre**. Mais plus qu'un simple interpréteur de dessins, nous avons montré que la conception même de SVALABARD est axée sur un approche globale d'intégration de l'outil logiciel dans le processus créatif.

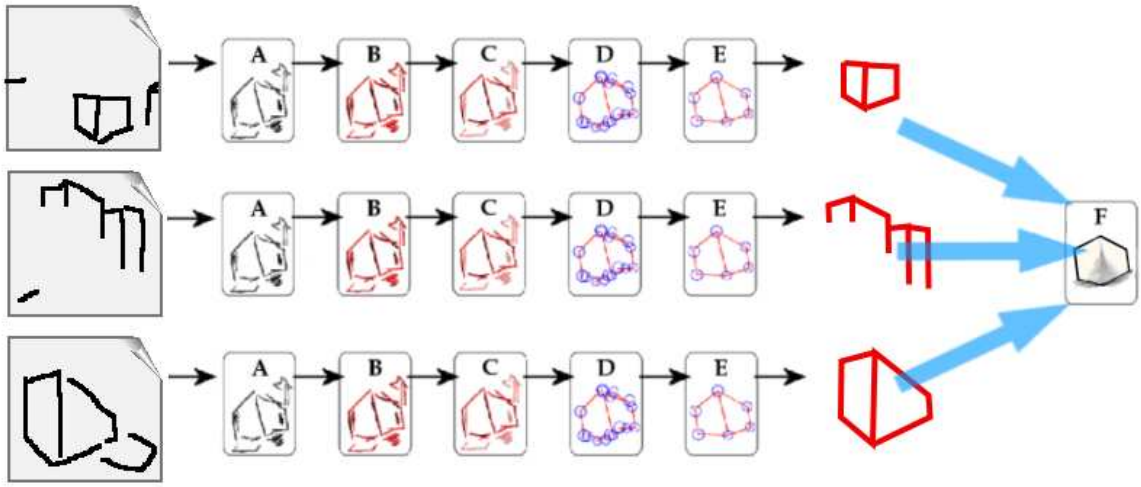
C'est pourquoi nous avons proposé une métaphore de **table à dessin virtuelle**, prenant sa source dans :

1. le concept original des **feuilles d'interaction** : chacune de ces feuilles d'interaction évoque un niveau sémantique du dessin de conception (spéculatif, prescriptif et descriptif), leur association créant un lien entre ces trois *modes*. La feuille de dessin, associée à son espace d'interaction (tablette écran), permet le dessin libre et spéculatif, avec des outils inspirés des méthodes habituelles (calques pliables, brosses de dessin). La feuille augmentée, associée à la tablette standard, présente l'interprétation géométrique du dessin et offre des interactions permettant la saisie de données précises sur les éléments du dessin. La feuille 3D, enfin, est le support de présentation de l'objet créé, permettant sa visualisation en trois dimensions et sa présentation. Le comportement de ces feuilles inscrit l'utilisateur dans une démarche non guidée, tout en offrant, par leur superposition graphique, une vision globale du projet et de sa compréhension par le système.
2. l'utilisation de **périphériques d'entrée non-standard** : employés dans un modèle d'interaction instrumentale et au maximum de leurs possibilités, ils permettent une réalisation efficace de la manipulation directe. Cela permet alors une vision claire et non ambiguë des possibilité du système, tout en procurant des interactions naturelles (basées sur le dessin et l'interaction bimanuelle).

Nous pensons que cette approche permet alors de qualifier un tel système **d'outil** de conception.



(a) Fusion *a priori*.



(b) Fusion *a posteriori*.

FIGURE 4.33 – Filtres de traitement et calques. Dans la figure (a), les calques sont tous connectés à la même chaîne de filtres du dessin (approche actuelle). Dans la figure (b), le dessin de chaque calque est interprété séparément. Les interprétation sont fusionnées au moment de la reconstruction 3D.

Nous avons aussi proposé des filtres de traitement en temps réel du dessin en perspective, des interpréteurs permettant de déterminer la structure du dessin. Reprenant en partie des traitements éprouvés, nous y avons toutefois introduit des algorithmes originaux permettant leur adaptation au dessin d'architecture en perspective : la détection des phases de dessin et la fusion de segments. Ces deux traitements permettent l'interprétation de dessin libre et ont des répercussions importantes sur le système. Ils permettent en effet de ne pas contraindre le dessin à réaliser par l'utilisateur, de ne pas trop orienter sa démarche par la finalité du système, en prenant en compte les spécificités du dessin de conception (révisions de traits, détails et environnement, traits de construction).

Enfin, plus qu'une application, SVALABARD peut être vu comme une architecture modulaire pour un système d'aide à la conception, pouvant être amélioré et configuré selon les besoins. C'est un système ouvert, à tous les niveaux : interactions, comportements et filtres de dessin.

Dans le prochain chapitre, le dernier de cette partie, nous identifions les apports essentiels de notre approche, ainsi que les points indispensables à sa finalisation (modules à rajouter et évaluation). Nous analyserons alors ses forces et ses faiblesses, et tenterons de répondre aux questions plus générales qu'elle soulève.

Chapitre 5

Discussions et perspectives

« Les ordinateurs facilitent beaucoup de choses, mais la majeure partie de ces choses qu'ils rendent plus faciles ne nécessitait pas d'être faite »

A. ROONEY.

Sommaire

5.1 Introduction

Dans ce chapitre, nous identifions les apports essentiels de SVALABARD, mais aussi ses limites et les points qui nécessitent d'être approfondis afin de respecter les lignes directrices que nous avions énoncées dans le chapitre 3.

Nous détaillerons d'abord les modules devant être finalisés pour que SVALABARD permette la création de modèles 3D : la connexion au noyau mathématique de reconstruction et la détection des propriétés du dessin (contraintes géométriques). Outre les problèmes d'ordre technique, nous verrons que ces derniers modules soulèvent des questions quant à leur impact sur l'interaction.

Ensuite, nous identifierons en trois points les apports qui font de SVALABARD une étape vers un *outil* de modélisation 3D : l'environnement proposé, la démarche libre et la configurabilité/adaptabilité du système. Mais, pour chacun de ces points, nous détaillerons aussi les limites actuelles et proposerons des pistes de solutions pour leur amélioration.

Dans les dernières sections de ce chapitre, nous proposerons différentes voies de recherche pour des travaux futurs pouvant utiliser ou s'inspirer de notre approche. Nous évoquerons dans un premier temps l'évaluation du système, indispensable à son amélioration et son évolution. Ensuite, nous verrons des propositions d'extension possible de notre approche à d'autres domaines d'application, mais aussi à d'autres environnements matériels.

5.2 Finalisation du système

Nos travaux apportent des solutions interactives pour l'analyse du dessin. Ils peuvent être complétés sur la détection (et l'aide à la saisie) des propriétés du dessin nécessaires pour sa reconstruction 3D et la connexion au noyau mathématique de GINA. Nous avons commencé à explorer les problèmes que soulèvent encore ces derniers éléments.

5.2.1 Détection des propriétés du dessin

Afin de concrétiser complètement notre approche de dessin libre pour la modélisation 3D, il est important d'introduire des modules de détection automatique des propriétés géométriques du dessin afin d'en obtenir le plus d'informations possibles pour le noyau de reconstruction.
Car, même si nous avons proposé des interactions permettant de la soulager, la saisie de ces contraintes géométriques reste un facteur limitant la créativité. Outre la quantité non négligeable d'informations à saisir, cette démarche impose une spécification précise du modèle, ce qui s'avère être contraire à la démarche floue et progressive des premières étapes de la conception. De plus, il s'agit d'une spécification basée sur un langage géométrique, et non architectural (dans notre domaine d'étude) ou du domaine du concepteur (design, ingénierie). Il nous semble alors primordial que le système prenne en charge le plus possible cette tâche, tout en proposant d'éventuels retours à l'utilisateur dans un langage ou un formalisme qui lui soit adapté.

Nos premiers traitements sur le dessin à main levée permettent d'en obtenir une représentation structurée. Dans la continuité de ces traitements, le système doit prendre en charge, du moins en partie, la construction d'une représentation de plus haut niveau, une interprétation *sémantique*. Une telle analyse s'apparente au domaine de la reconnaissance de formes, sur lequel nous ne nous attarderons

pas plus ici. Nous citerons les travaux récents de Peter A. C. VARLEY sur l'interprétation de dessins d'objets polyédriques [Varley, 2003; Varley *et al.*, 2004]. Les méthodes employées et les résultats obtenus par les différents modules qu'il propose nous semblent adaptables à :

1. notre problématique, car permettant la détection des propriétés et des parties cachées du dessin pour le noyau de reconstruction ;
2. notre domaine d'application, car bien que réalisées pour l'ingénierie, ces propositions nous paraissent adaptables à l'architecture par la définition de nouvelles heuristiques ;
3. notre architecture logicielle, car suffisamment modulaires pour permettre leur insertion dans notre cascade de traitements.

Mais il s'avère que le problème majeur de cette détection des propriétés est son insertion dans une démarche itérative, où les données ne sont connues qu'au fur et à mesure de la construction et peuvent de plus être révisées à tout moment. C'est pourquoi il sera aussi important d'envisager l'utilisation de techniques compatibles avec une telle démarche. Nous pensons notamment aux travaux récents sur les ontologies et leur utilisation dans les bases de connaissances [Fürst, 2004]. Cette approche permet une représentation de haut niveau des données, offrant aussi des techniques de raisonnement adaptées à la démarche itérative. L'enjeu est alors de déterminer précisément le domaine des connaissances manipulées : si les connaissances du domaine de la géométrie projective sont évidemment celles qui se prêtent le mieux au raisonnement pour l'interprétation de haut niveau du dessin [Fürst *et al.*, 2003], elles ne nous semblent pas adaptées au langage de l'architecte (ou du designer) pour ce qui est d'éventuels retours à l'utilisateur.

Car quoi qu'il en soit de la fiabilité de ces méthodes, il nous semble mal indiqué de vouloir penser « tout automatique » dans un système dédié à la conception, ne serait-ce que par l'ambiguïté ou la multiplicité des interprétations possibles. Ainsi, nous pensons que ces techniques devront être insérées dans le but de *soulager* l'activité inévitable de précision du modèle qu'induit l'outil informatique (activité qui n'existe pas dans le processus habituel de conception). Ainsi, les interactions que nous avons proposées pour la saisie des contraintes pourront être une première piste pour *guider* le système ou corriger d'éventuelles interprétations erronées.

Il convient alors de se poser la question de savoir si une telle démarche ne contredit pas nos principes de ne pas imposer un chemin défini d'avance au concepteur en lui demandant d'expliciter trop tôt ses choix. Nous aborderons ce sujet du dialogue à établir entre l'utilisateur et le système dans la section 5.3.2, où nous discuterons une solution possible.

5.2.2 Connexion au noyau de reconstruction

Nous n'avons pas réalisé le dernier module de notre chaîne de traitement du dessin : la connexion au noyau de reconstruction 3D, ayant jugé comme prioritaire la spécification et la conception des interactions. En effet, les problèmes soulevés par la connexion au noyau s'avèrent être surtout d'ordre technique. Il est par contre évident que cette dernière étape est indispensable pour tester et évaluer le système.

Le noyau proposé par Alex SOSNOV [Sosnov, 2003] est un solveur de contraintes géométriques incrémental. Il permet, à partir des caractéristiques 2D d'un dessin (points, segments et plans) et de leurs propriétés géométriques (incidences, parallélismes et orthogonalités) de reconstruire le modèle 3D polyédrique correspondant. Outre ses atouts liés à la performance et l'efficacité, cette approche permet une reconstruction incrémentale, tout a fait adaptée à notre application pour deux raisons :

1. Cette notion de reconstruction incrémentale permet une saisie et une gestion de données incomplètes, même si celles-ci n'aboutissent pas à la reconstruction d'un modèle. Le noyau va alors maintenir la cohérence des données déjà reçues, et effectuer des propagations permettant d'inférer de nouvelles contraintes.
2. Cette même notion permet en outre l'ajout de propriétés et le raffinement du modèle.

Cette démarche s'inscrit donc bien dans les premières phases de conception où sont manipulées et raffinées des données incomplètes.

La connexion entre notre système d'interprétation des dessins et le noyau va devoir établir un dialogue entre les deux entités. Le système d'interprétation devra envoyer les données recueillies au fur et à mesure au noyau, celui-ci envoyant en retour son état, ainsi que les propriétés qu'il aura pu inférer (modèle reconstruit, propriétés géométriques déduites, erreurs ou propriétés manquantes). Se pose alors le même problème que précédemment à propos de la présentation de ces retours à l'utilisateur ; en terme de quantité, tout d'abord (doit-il tout voir et tout savoir ?), de sémantique (contraintes, géométrie, ou éléments du domaine ?) et de mode de représentation. Nous avons déjà abordé la représentation des propriétés géométriques sur la feuille augmentée dans le chapitre précédent. Nous reviendrons sur la représentation d'éventuels problèmes de reconstruction dans la section 5.3.2 page 135 de ce chapitre.

Amélioration du rendu 3D

Nous souhaitons aussi apporter des améliorations au rendu du modèle 3D que nous avons proposé. Dans [Schumann *et al.*, 1996], les auteurs montrent par une étude comparative que les architectes perçoivent différemment un modèle 3D rendu de manière « standard » et non photo réaliste. Ils constatent qu'un tel mode de visualisation rend plus ouverte la discussion et la spéculation sur le modèle, offrant plus de liberté en cours de conception. Bien que notre rendu actuel évoque déjà une certaine « imprécision » de par son aspect non-photoréaliste, nous pensons qu'un rendu des lignes de type crayonné serait encore plus efficace pour conforter l'utilisateur dans sa démarche créative. Le logiciel SKETCHUP que nous avons déjà présenté permet d'ailleurs une représentation de type croquis, basée sur une jointure imprécise des lignes, ainsi que leur prolongement. Nous pensons utiliser une méthode similaire, inspirée de [McGuire et Hughes, 2004] et [Nienhaus et Döllner, 2004].

Croquis en 3D

Enfin, nous devons aussi réaliser notre proposition de projection du modèle 3D dans la feuille de dessin après manipulation par l'utilisateur. Ainsi, il aura la possibilité de continuer son dessin dans un autre point de vue afin de le compléter, de l'affiner.

Bien que proche du concept proposé par Takeo IGARASHI dans TEDDY [Igarashi *et al.*, 1999], où il est possible de changer de point de vue pour modifier la forme en construction avec des gestes, une telle technique de croquis de conception en trois dimensions n'a à notre connaissance pas encore été proposée. Dès lors, plus que l'intérêt de son implémentation, il sera à notre avis important d'évaluer l'impact, positif ou négatif, de cette technique sur l'utilisateur et sa démarche. Nous pensons particulièrement à une perte des informations de conception existantes dans le dessin original. Comme nous l'avons vu dans le chapitre 1, le dessin n'est pas qu'une simple « image » de la pensée du concepteur. Il est aussi l'historique et le support des révisions au cours de la conception. Peut-on, mais aussi doit-

on, reproduire ces notions lors de la reprojection du modèle 3D dans un nouveau point de vue sur la feuille de dessin ?

5.3 Apports et discussions

Dans cette section, nous identifions les apports, qui, selon nous, font de notre approche une nouvelle étape vers la conception de véritables *outils* informatiques de modélisation 3D pour la conception créative. Parallèlement à ces apports, nous positionnerons nos travaux par rapport à l'existant afin d'ouvrir des discussions. Ces discussions s'articulent autour des trois caractéristiques qui dénotent de l'intégration d'un système informatique dans un contexte créatif (selon les lignes directrices de la proposition 3.8 page 75) :

1. l'environnement proposé (métaphore, dispositifs physiques et interactions) ;
2. la démarche induite par le logiciel ;
3. son adaptabilité.

5.3.1 Un environnement de conception

Notre approche se distingue des outils traditionnels de CAO, mais aussi de la majorité des travaux de recherche précédents, par le fait que nous avons voulu SVALABARD comme un environnement de conception créative s'intégrant « physiquement » dans la démarche du concepteur. Ainsi, comme nous l'avions énoncé dans nos lignes directrices, nous sommes partis du principe que l'utilisation de l'outil informatique dans les premières phases de la conception ne passait pas uniquement par une saisie et une interprétation de dessins, mais aussi par une métaphore et des interactions adaptées, proches des habitudes de l'utilisateur. Dans cette optique, SVALABARD repose sur une métaphore liée à la conception et au *croquis* qui reprend et intègre différents résultats de recherche (interaction gestuelle, interprétation de dessin, manipulation directe, etc.) et innove selon nous par leur insertion cohérente dans un environnement global. Il est évident que tant que nos choix de conception n'ont pas été évalués et validés par des études, nous ne pouvons pas en affirmer l'efficacité et l'utilisabilité. Nous pouvons toutefois en identifier les originalités qui permettent d'envisager des résultats intéressants : le paradigme des *feuilles d'interaction*, la notion d'*espaces d'interaction* et les *calques de dessin* qui donnent lieu à la métaphore de la *table à dessin virtuelle*.

Feuilles d'interaction

La métaphore visuelle du *papier* a souvent été employée en informatique et de plus en plus efficacement (grâce aux progrès matériels). Mis à part l'aspect visuel *attirant* et *confortant* qui permet de placer l'utilisateur dans un contexte particulier [Denoue *et al.*, 2003], cette métaphore a aussi permis l'introduction d'interactions plus souples et intégrées [Beaudouin-Lafon, 2001; Roussel, 2003]. De ces points de vue, les feuilles d'interaction de SVALABARD sont particulièrement innovantes dans le domaine de la modélisation 3D en particulier, et des outils informatiques pour la conception créative en général.

Des modes implicites
Dans un premier temps, les feuilles d'interaction peuvent être vues comme les différents modes du

système : un mode de dessin et de création (la feuille de dessin), un mode d'interprétation et de précision des éléments du dessin (la feuille augmentée) et un mode de visualisation et de manipulation de l'objet de la conception (la feuille 3D). Un avantage certain concerne la représentation de ces modes : une représentation *verticale*, basée sur la superposition des feuilles semi-transparentes. Cette approche évite les problèmes habituels des environnements multi-fenêtrés tels qu'une gestion délicate de l'espace de travail et des occlusions.

De plus, nous avons proposé un changement de mode implicite d'une feuille à l'autre, piloté par les actions de l'utilisateur et ses besoins. Les feuilles apparaissent et disparaissent selon les actions (dessin, manipulation 3D, etc.) et même lorsque toutes les feuilles sont présentes sur l'espace de travail, elles restent visibles par superposition et transparence.

Un lien entre les fonctions du dessin
En allant plus loin, cette superposition en transparence présente un autre avantage. Elle permet la mise en relation implicite des données supportées par chacune des feuilles. Il est ainsi aisé de voir les parties correspondantes de la feuille augmentée, de la feuille de dessin et de la feuille 3D. Cette notion de vue superposée des différentes données des feuilles d'interaction permet de comprendre le fonctionnement du système. Si l'on prend par exemple le cas de l'interprétation du dessin sur la feuille augmentée, sa superposition avec le dessin original permet de comprendre les mécanismes d'interprétation du système, et donne donc un retour précis de ses actions mais aussi de ses éventuelles erreurs.

D'un point de vue plus conceptuel, ces feuilles peuvent être associées aux différentes fonctions de la représentation figurative, notamment en conception architecturale : spéculatif (la feuille de dessin), prescriptif (la feuille augmentée) et descriptif (la feuille 3D). Bien que notre système ne soit pas conçu pour la production de plans de construction précis, il constitue toutefois une première approche vers l'intégration de ces trois « modes » d'une manière efficace (utilisation du dessin spéculatif pour construire les deux autres, possibilité de concevoir des interactions plus adaptées pour la saisie de plans, etc.).

Instruments

Un autre principe majeur de notre approche est l'utilisation de périphériques d'entrée adaptés aux tâches à réaliser, dans un souci d'accès rapide et intuitif aux fonctionnalités du système. Car, bien que de nombreux dispositifs aient été proposés par les fabricants spécialisés, peu de travaux ont étudié leurs apports dans un environnement de conception créative avancé (à part les tablettes graphiques).

Notre utilisation de ces périphériques ne se contente pas d'en faire des « souris avancées ». Nous avons construit les instruments de l'application en étroite relation avec leurs capacités. Ainsi, s'il était évident que les six degrés de liberté du *Magellan* s'imposeraient pour la manipulation de l'objet 3D, notre proposition de manipulation des calques avec toutes les fonctionnalités du *Shuttle* a nécessité plus de réflexion et d'itérations : quel dispositif utiliser pour manipuler les calques de manière naturelle et comment tout faire sur les calques avec ce dispositif. Mais aussi comment utiliser toutes les capacités du dispositif afin d'optimiser la bande passante en entrée. Cette démarche nous a permis de concevoir des interactions a priori naturelles entre l'utilisateur et l'instrument de gestion des calques (le pliage direct, la navigation entre les calques).

Il s'ensuit alors une connexion complète entre les dispositifs et les outils de l'application pour composer des *instruments*. Dans ce sens, cette *instrumentation* physique de l'interface rappelle la no-

tion d'outils physiques, non ambiguë pour le concepteur. Nous pensons que ce principe favorise la prise en main et la mémorisation des fonctionnalités du système (bien que nécessitant tout de même un apprentissage et une description des associations que nous évoquerons plus loin). De plus, c'est une réalisation efficace de la *manipulation directe* par le modèle d'*interaction instrumentale* [Beaudouin-Lafon, 1997; Beaudouin-Lafon, 2000], améliorant son efficacité en réduisant le coût d'activation des instruments de l'interface, ne nécessitant ni pointage ou sélection préliminaire et permettant des interactions bimanuelles naturelles.

Finalement, ce paradigme des feuilles d'interaction peut être vu comme une réalisation étendue du modèle multi-couches [Fekete et Beaudouin-Lafon, 1996; Fekete, 1996]. Il en reprend l'aspect modulaire et flexible pour son implémentation et son utilisation ainsi que les possibilités de décomposition sémantique et d'organisation pratique et fonctionnelle. Toutefois, il permet en plus un contrôle fin des interactions au niveau de chaque couche, rendant possible l'association de périphériques d'entrée et d'outils, ainsi que la description du comportement dynamique des feuilles.

Limites

Cette approche soulève tout de même le problème des dispositifs d'entrée spécifiques, mais aussi de l'apprentissage des fonctionnalités du système qui ne sont pas toujours visibles au premier abord.

Dispositifs spécifiques
En effet, nos propositions reposent autant sur des interactions avancées et adaptées que sur leur association avec des périphériques d'entrée non standard qui ne sont pas toujours présents sur toutes les configurations matérielles.

Ce problème de la disponibilité et de l'adaptation à différentes configurations a été pris en compte dés le début de la conception de SVALABARD, notamment en utilisant des boîtes à outils et une architecture logicielle modulaire et flexible. Dès lors, le système est complètement indépendant des dispositifs et peut être rapidement adapté à des configurations plus standards ou même plus avancées. Bien sur, utiliser un clavier et une souris pour interagir avec SVALABARD brise une grande partie de la métaphore de la table à dessin, construite aussi bien par les interactions que les dispositifs utilisés. Mais cela est tout de même envisageable.

Visibilité et apprentissage des fonctionnalités
Du point de vue de l'utilisation du système, et de sa prise en main, il est clair que nous devons nous poser la question du temps et des moyens nécessaires à l'apprentissage de ses fonctionnalités. Car nous sommes conscients qu'une telle approche n'est pas en tous points écologique, les outils ne signifiant pas toujours d'eux-mêmes ce qu'ils permettent de réaliser. Par exemple, s'il va être évident de comprendre que le stylet sur la tablette écran permet de dessiner, il sera moins sûr que le *Shuttle* permette de manipuler les calques. De la même manière, rien n'indique a priori à quoi sert la zone de reconnaissance de gestes, et même lorsque sa fonction est connue, il faut connaître les gestes et les fonctions correspondantes. Ce problème de « visibilité » des actions est récurrent dans les interfaces basées sur la reconnaissance de marques [Baudel, 1995]. L'emploi des « marking menus » peut permettre de remédier à cela, en fournissant un rappel des gestes, mais rien n'indique non plus explicitement comment ils sont activés (le bouton latéral du stylet dans notre cas).

Nous pensons toutefois qu'une courte explication des fonctionnalités du système, assortie d'une présentation des dispositifs d'entrée suffit à se faire une idée de son utilisation. De plus, le regrou-

pement des outils logiques en espaces d'interaction, souvent sur un même dispositif, peut permettre une meilleure mémorisation de ces fonctionnalités. Il serait même envisageable d'introduire un mécanisme d'aides contextuelles, basées sur des macros ou des vidéos, qui signifierait les actions possibles à l'utilisateur lors de la première apparition d'une feuille d'interaction (dans l'esprit de l'image que nous avons réalisée pour la figure 4.12 page 97, présentant en vignette la manipulation du *Shuttle* pour plier les calques).

Ce ne sont que des propositions, et il est évidemment indispensable de les évaluer afin de les confirmer, mais il nous semble que le temps d'apprentissage (et d'éventuelles erreurs) induit par une approche non standard telle que la nôtre sera minime par rapport aux gains qu'elle peut apporter, particulièrement en termes d'intégration dans une démarche créative.

5.3.2 Une démarche libre

En plus de son environnement matériel et logiciel, SVALABARD se veut aussi un support à la création de par le fait qu'il n'impose pas une démarche précise et figée. Il n'y a ni ordre, ni primitive de construction.

La notion de retouche

Les systèmes actuels, commerciaux ou prototypes, permettent la retouche du modèle au cours de sa création, mais rarement dans le sens où le concepteur l'utilise. En effet, manipuler des données précises (primitives géométriques) au tout début de la conception limite de fait les possibilités créatives, mais rend la modification et la révision du modèle plus ardue. Dans un premier temps, parce que des « chemins » de conception ont étés fermés. Dans un second temps, parce-que les techniques d'interaction et de modification de ces données ne sont pas aussi naturelles que le dessin, représentation figurative qui offre à la fois l'incertitude nécessaire à la spéculation mais aussi les techniques simples d'itération.

Or, dans les approches actuelles, révision et amélioration sont plus assimilables à de la gestion de versions, car trop liées à l'emploi de structures de données précises et globales (primitives de haut niveau). Certes, cette approche offre un avantage certain en terme de gestion de projet, mais n'est pas vraiment en accord avec les structures manipulées par l'utilisateur. Notre démarche se démarque sur ce point, car axée autour du *trait* comme donnée commune à l'utilisateur et au système, dans un contexte de *dessin libre*. Il est vrai que l'utilisation et l'interprétation du dessin comme médium entre le concepteur et le système informatique n'est en rien une nouveauté. Toutefois, notre approche se distingue par la notion de dessin libre, sans contrainte ni de vue (grâce aux possibilités du noyau de reconstruction utilisé) ni de qualité du dessin (vectoriel, limitation à la saisie de l'objet d'intérêt, etc.). Ainsi, même si dans les traitements le système d'interprétation va transformer ces données pour les exploiter, leur transformation et leur évolution vers des solutions plus précises aura toujours pour support le croquis et ses traits, associé à l'utilisation de calques.

Filtres et traitements du dessin
Les filtres de traitement du dessin prennent une place importante dans la façon dont le système va s'intégrer dans la démarche de conception : ils déterminent les possibilités de transformation des traits saisis vers une structure de données exploitable par le système. Ils sont aussi intimement liés à la tâche et au domaine d'utilisation et doivent être développés dans un contexte particulier. C'est ce

que nous avons tenté de réaliser par nos proposition, notamment par les filtres de *détection des phases de dessin* et de *fusion des segments* dans le cadre du croquis d'architecture en perspective.

Tout en conservant la démarche libre du concepteur en architecture, et notamment la révision grâce à la fusion des segments, nous pouvons produire en temps réel une structure vectorielle du dessin, un graphe 2D, utilisable pour des interprétations de haut niveau et la reconstruction 3D. Il convient toutefois de mesurer ce discours, de par le fait que ces traitements n'ont pas encore été évalués dans des conditions réelles d'utilisation, et ne s'appliquent pas non plus pour la production d'autres structures de données telles que des courbes. Toutefois, ils sont adaptés à la fois à la notion de démarche libre et itérative et à la finalité de notre système, son noyau mathématique.

Calques de dessin
Enfin, le principe des calques de dessin que nous avons proposé est aussi un atout pour le support de la conception et de sa démarche. En effet, outre leur apport à la métaphore des outils habituels de conception, les calques offrent dans notre système la même capacité de « mémoire » et de révision que leurs équivalents réels. C'est un autre point sur lequel notre système reproduit et emprunte ses outils aux méthodes de conception pour favoriser la démarche libre et créative. Il reste toutefois, comme nous l'avons souligné dans le chapitre précédent, à clarifier le mode de connexion des interpréteurs aux calques de dessin afin d'en améliorer la pertinence.

Limites

Certains points de SVALABARD peuvent encore contraindre la démarche de l'utilisateur. Tout d'abord, la fiabilité des filtres de dessin sous leur forme actuelle peut engendrer des erreurs d'interprétation, de reconstruction 3D, et, plus grave, de comportement du système (détection d'une mauvaise phase de dessin pour le filtrage des traits, par exemple). Dans un même ordre d'idée, les solutions que nous avons proposées pour la détection des propriétés de haut niveau du dessin peuvent conduire à des échecs ou des ambiguïtés (solutions multiples). Il nous semble donc que dans le cadre où nous avons inscrit nos travaux (le dessin libre pour la modélisation 3D), il est nécessaire de s'orienter vers une coopération entre le système et la machine pour régler ces problèmes plutôt que de viser une automatisation complète du processus d'interprétation du dessin.
Après avoir identifié les problèmes techniques actuels de nos filtres et proposé tout de même des pistes pour les améliorer, nous proposerons un compromis quant à l'intervention de l'utilisateur dans le traitement du dessin et la place du système dans la démarche de l'utilisateur.

Amélioration des filtres bas niveau
Nos algorithmes originaux de *fusion de segments* et de *détection des phases de dessin d'architecture* sont essentiellement basés sur des heuristiques et des analyses issues de notre étude. Bien que fiables dans le cas général, il n'en demeure pas moins quelques singularités entraînant des erreurs parfois pénalisantes pour le traitement du dessin, telles que la fusion erronée de segments (cas des « écrasements » dus à la vue perspective) ou un mauvais filtrage de traits.

Pour ce qui est de la détection des phases du dessin, il nous semble impératif de rendre ce traitement adaptatif à l'utilisateur de manière à prendre en compte les spécificités de ses méthodes et ses habitudes. L'étude de cette notion de *profil utilisateur* est une voie qui suffirait probablement à identifier et trouver les limites de notre approche.

Dans le cas de la fusion des segments, la principale limite est due à une considération locale du

problème. Il serait intéressant d'envisager une méthode plus globale, associant les heuristiques que nous avons proposées à une analyse de la topologie du graphe 2D formé par les points et segments détectés (recherche de *pendants*[(1)], de *cycles*, etc.). Nous reviendrons plus en détail sur cette approche dans l'annexe A page 3.

Finalement, même si il est indispensable d'approfondir les travaux que nous avons engagés sur les filtres et traitements du dessin, notre approche offre surtout une architecture logicielle permettant cette amélioration et cette évolution. La décomposition du traitement en plusieurs modules, qui peuvent être connectés dynamiquement, induit la flexibilité nécessaire à l'amélioration, l'insertion ou le remplacement de traitements sans pour autant remettre les autres en question.

Saisie des contraintes
Actuellement, la saisie des propriétés de haut niveau nécessaires à la reconstruction 3D est réalisée par l'utilisateur. Il est évident que cette saisie est fastidieuse, en particulier dans le cas d'un dessin important et complexe. Mais elle n'est toutefois pas si intrusive du point de vue de la démarche de conception, pouvant être réalisée à n'importe quel moment (au cours du dessin, ou à la fin). Bien entendu, cela pose tout de même le problème d'une longue tâche supplémentaire à réaliser, et ne permet pas de tirer tous les avantages de l'outil informatique. En particulier, si la saisie de ces propriétés est effectuée en fin de conception, le concepteur ne bénéficie pas de la construction d'un modèle 3D en parallèle à sa démarche.

Si l'on revient aux hypothèses originales du projet GINA, l'idée de départ était une saisie simultanée du dessin et des contraintes en utilisant plusieurs modalités : le stylet pour le dessin et la parole pour les contraintes. Ainsi, l'utilisateur décrirait son dessin au fur et à mesure du tracé. Nous n'aborderons que de loin les problèmes techniques de ce principe (mise au point d'un vocabulaire de description et fusion multimodale), nous nous concentrerons essentiellement sur la démarche qu'il engendre.

Dans le cadre de la modélisation 3D dans les premières phases de la conception architecturale, un tel principe n'est à notre avis pas adapté. En effet, il est tout d'abord important de définir un vocabulaire de description adapté au domaine. Or dans le cadre de l'architecture, le vocabulaire est tout de même d'un niveau assez élevé, posant à notre avis un problème de fusion multimodale trop délicat (beaucoup de traits du dessin, qui de plus ne seront pas forcément tracés consécutivement, vont représenter une caractéristique architecturale déclarée). Il serait alors nécessaire de contraindre fortement la démarche du concepteur. En l'obligeant, dans un premier temps, à ne dessiner en séquence que ce qu'il vient d'énoncer (ou l'inverse, expliquer ce qu'il vient de dessiner). Mais aussi en le contraignant à préciser peut être trop tôt ses choix de conception. Toutefois, si l'on fait abstraction de la contrainte qui consiste à obliger le concepteur à « penser tout haut », cette approche multimodale serait un atout pour délimiter les phases de conception associées à des objets d'intérêt [Leclercq *et al.*, 2004], plutôt que pour leur description précise (et géométrique).
Ce principe de description précise nous semble bien plus adapté à la modélisation 3D dans un cadre géométrique ou pour la saisie de modèles 3D déjà conçus ; en bref, lorsque la géométrie est bien définie. Le vocabulaire peut alors être de plus bas niveau, basé sur les points, droites, parallélismes, etc., facilitant alors la fusion avec le dessin (mais à notre avis incompatible avec les concepts que manipule l'architecte).

Il est selon nous important de s'orienter vers une collaboration entre l'utilisateur et le système pour l'interprétation du dessin et son élévation en 3D, notamment au niveau de la détection des pro-

(1) Les *pendants* dans un graphe sont des sommets de degré 1.

priétés. Bien que ce principe puisse paraître en contradiction avec notre proposition de dessin et de démarche libre, il nous semble un compromis nécessaire pour éviter d'ajouter des tâches « annexes » à la conception trop lourdes pour le concepteur, ainsi que pour éviter des interprétations erronées par le système.

Retours, corrections et collaboration à l'interprétation
Du fait de l'imprécision des données traitées dans les premières phases de la conception, il est inévitable que des traitements automatisés aboutissent à plusieurs solutions possibles au terme de leurs analyses. Ils doivent donc être vus comme des outils d'aide à la génération de solutions, comme nous l'avions évoqué dans le premier chapitre de ce mémoire, sans pour autant que ce soit le système qui fasse le choix d'une solution. Le problème majeur est alors la présentation de ces interprétations et solutions à l'utilisateur, de manière ni contraignante, ni intrusive.

Nous sommes d'avis qu'un principe d'interface suggestive, dans l'esprit de celle proposée par Takeo IGARASHI dans [Igarashi et Hughes, 2001], est une solution efficace. Cette technique, proche du principe des « What-if Tools » de Ben SHNEIDERMAN [Shneiderman, 2000] présente différents avantages, le plus important étant de ne pas interrompre la tâche de l'utilisateur tout en lui permettant de suivre et guider le système. Mais l'impact n'est à notre avis pas négligeable non plus sur sa démarche, lui offrant la possibilité de concrétiser plus rapidement des solutions ou de poursuivre à son propre gré.

Un tel mécanisme de suggestions serait un atout certain dans notre système pour la présentation des retours du système d'interprétation, permettant de conserver la notion de démarche libre. Dans notre paradigme des feuilles d'interactions, ces suggestions pourraient être présentées sur la feuille augmentée sous forme de vignettes interactives contenant une représentation graphique des propositions, renforçant ainsi l'aspect non-intrusif de cette technique (la feuille augmentée n'apparaît que lorsque l'utilisateur ne dessine plus). Cette approche est un bon compromis pour une collaboration entre le système et l'utilisateur, à plusieurs niveaux : interprétation bas niveau du dessin, détection des propriétés et proposition de solutions.

5.3.3 Un système configurable et adaptable

Le dernier point sur lequel SVALABARD se démarque des travaux antérieurs est sa configurabilité et son adaptabilité en terme de dispositifs d'entrée, d'interactions et de comportements. En effet, l'utilisateur peut configurer simplement le système selon ses besoins et sa configuration matérielle. Mais cette conception modulaire simplifie aussi l'ajout de nouvelles fonctionnalités pour les développeurs ou d'interactions pour les concepteurs d'interfaces, faisant tendre notre proposition vers une architecture pour un système de support informatique à la créativité.

Nous ne développerons pas plus avant ces notions dans ce chapitre, celles-ci étant largement exposées dans la deuxième partie de ce mémoire où nous proposons un nouveau modèle d'architecture logicielle pour la conception des interfaces, implémenté dans la boîte à outil MAGGLITE (voir chapitre 8). Nous reviendrons précisément dans la section 8.4.3 page 203 sur le développement de SVALABARD avec ces outils.

5.4 Perspectives

Pour conclure ce chapitre, et cette première partie, nous proposons un survol non exhaustif des travaux futurs et perspectives de recherche que peut selon nous engendrer notre approche. Dans un premier temps, nous évoquerons l'évaluation de notre système, étape qu'il sera indispensable de réaliser afin de vérifier nos hypothèses. Nous proposerons ensuite des pistes pour l'amélioration de SVALABARD (traitement de courbes, autres types de dessins architecturaux) mais aussi pour son ouverture et son adaptation à d'autres domaines ou modes de conception (informatique mobile, design, art, conception collaborative).

5.4.1 Évaluation

SVALABARD est actuellement un prototype issu de nos études et de nos choix de conception. Il est évident que des évaluations seront nécessaires pour valider et améliorer ces propositions. Bien que n'ayant pas encore mis en place ces tests, nous avons toutefois envisagé leur contenu et leur déroulement. L'évaluation de l'utilisabilité de SVALABARD sera essentiellement basée sur la perception que peuvent en avoir d'éventuels utilisateurs, ainsi que sur les performances du système (évaluation des techniques d'interaction et des « taux de réussite » des algorithmes).

Choix des sujets
Il nous semble important d'évaluer notre système sur un panel de professionnels, tout particulièrement des architectes exerçants ou ayant exercé dans la conception de projets. Nous sommes toutefois conscients que la mise en place d'une telle évaluation avec un groupe de professionnels risque de ne pas être aisée. C'est pourquoi nous nous tournerons probablement, comme dans notre étude sur le trait, vers des personnels et étudiants d'Écoles d'Architecture.

Tâche libre
La partie importante de cette évaluation sera la proposition d'une tâche libre, d'un but à atteindre en utilisant le système comme le sujet l'entend. La consigne pourra reprendre par exemple celle que nous avions proposée pour notre étude préliminaire (concevoir un bâtiment nouveau ou en reproduire un connu sans supports graphiques). Cette partie de l'évaluation permettra de vérifier en partie la démarche que permet le système dans un contexte créatif. En partie, car il est évident que le temps d'apprentissage réduit ne permettra pas une appropriation complète de l'outil. Cela mériterait une étude plus longue, avec diffusion du système, mais nécessitant alors beaucoup plus de travail et de moyens (système plus stable et robuste car pas d'assistance directe durant les manipulations, besoin des dispositifs d'entrée, etc.).

Plusieurs données pourront alors être considérées, dépendant des points à évaluer : une évaluation subjective du système par les sujets (questionnaire) et une analyse quantitative basée sur des mesures pendant les activités (choix de l'utilisateur et performances du système).

La mise en place d'un tel travail nécessite beaucoup de temps et de moyens pour être réalisée dans de bonnes conditions. C'est en partie pourquoi nous n'avons pas encore pu la mettre en place. De plus, l'évaluation que nous proposons ne prend en compte que la première approche du système, son utilisabilité et sa prise en main. Bien qu'étant indispensable pour une première validation ou invalidation de nos choix afin de les améliorer, il nous semble qu'il serait aussi important de mettre en place une évaluation de plus longue durée, concernant l'appropriation du système par les utilisateurs

(pour vérifier en particulier la notion de liberté dans son utilisation et sa configuration).

5.4.2 Interpréteurs

Courbes

Comme nous l'avons précisé auparavant, nous avons essentiellement focalisé nos travaux sur les interactions et outils du système plutôt que sur les aspects techniques de l'interprétation des dessins et leur reconstruction en 3D. Il est évident que la transformation d'un croquis en données compréhensibles pour un système informatique a des impacts sur l'interaction, et c'est pourquoi nous avons tout de même émis des propositions à ce sujet (traitements bas niveau du dessin, propositions pour des traitements plus avancés et interactions associées). Nous n'avons toutefois pas encore évoqué un point de recherche intéressant : l'interprétation et la reconstruction en trois dimensions de *courbes*.

Il va de soi que la conception architecturale ne repose pas uniquement sur des enveloppes polyédriques comme le propose actuellement notre système. Il en est de même pour les autres domaines et pour la modélisation 3D en général. Dès lors, le système que nous proposons se doit de prendre en compte le dessin de courbes, afin de permettre leur reconstruction en 3D.

Du point de vue de l'interaction, l'interprétation de courbes en temps réel a déjà fait l'objet de recherches fructueuses comme par exemple [Sezgin *et al.*, 2001], dont nous avons déjà adapté les résultats pour la segmentation des traits. L'interprétation de trait courbes permet d'en obtenir une approximation par une courbe de Bézier. Pour ce qui est de leur modification, et donc des principes d'itération que nous avons mis en avant dans notre approche, la méthode proposée par Thomas BAUDEL dans [Baudel, 1994; Baudel, 1995] nous parait adaptée. Ces méthodes peuvent rapidement être implémentées et rajoutées dans notre cascade de traitements, grâce à son architecture modulaire.

Si nous évoquons ce point dans cette section sur les perspectives, c'est surtout parce que la reconstruction de courbes continues en 3D nécessite une recherche complète et une évolution majeure du noyau de reconstruction 3D que nous connectons à SVALABARD. En effet, les formalismes géométriques actuels du noyau ne permettent pas la représentation et la reconstruction de courbes continues[(2)], bien qu'il soit tout de même possible d'en construire des approximations par des segments.

Dessin géométral et annotations

Une autre voie d'évolution intéressante consiste à étudier l'introduction de calques de *dessin géométral* sur la feuille augmentée. Ces calques permettraient la création de plans de construction dans le même environnement et avec les mêmes principes que pour le dessin en perspective. Le concepteur pourrait tirer partie de la projection en coupe plane du modèle 3D reconstruit, associée à des algorithmes d'interprétation de dessin 2D tels que ceux d'ESQUISE [Leclercq et Juchmes, 2002]. C'est une piste intéressante pour la réunion des deux approches dans le but de fournir un outil de conception architectural plus complet.

Toujours sur la question des interprétations, une codification et une reconnaissance des annotations tracées sur la feuille augmentée améliorerait les capacités descriptives du système, ainsi que ses possibilités en terme de simulations. En effet, l'utilisation de symboles et de mots usuels (cui-

[(2)]Les travaux présentés dans [Jourdan *et al.*, 2004] sont une voie pour de telles améliorations.

sine salle de bain), toujours dans le même esprit que les interactions d'ESQUISE, est un bon moyen pour l'identification des espaces fonctionnels d'un éventuel calque de dessin plan. Mais aussi, dans le cas général, la reconnaissance d'écriture pour les annotations offrirait des possibilités d'indexation et d'archivage des croquis pour la gestion de projets architecturaux.

5.4.3 Conception « mobile »

Depuis quelques années et la démocratisation de l'informatique mobile (assistants personnels, mais surtout « TabletPC »), la notion de *CAO mobile* a émergé. La puissance croissante de ces plateformes informatiques a permis de prendre en compte un aspect important de la démarche du concepteur, en particulier en architecture ou en construction : la mobilité. Ainsi, des logiciels commerciaux proposent de la CAO sur ces plateformes ; nous pouvons citer par exemple POCKETCAD PRO [Arc Second, Inc, 2004], POWERCAD PRO [GiveMePower Corporation, 2004] et FIELDPROFESSIONAL [fieldDesigner, Inc, 2004] (basés sur AUTOCAD) qui permettent *d'emporter* l'environnement de CAO directement sur le site de construction, les deux derniers offrant même des possibilités de capture des données de stations de base et de scanners laser. Toutefois, ces approches basées sur les paradigmes standards de CAO offrent à notre avis encore moins de capacités créatives que leurs versions de bureau (voir figure 5.1). En effet, mis à part l'aspect portable et mobile, rien n'a été mis en œuvre pour améliorer les aspect créatifs de la conception ; au contraire, ce sont même des systèmes encore plus spécifiquement orientés construction que les systèmes de bureau.

(a) Mobilité avec la CAO actuelle.

(b) Mobilité avec les interfaces gestuelles.

FIGURE 5.1 – Conception embarquée. Au delà du clin d'œil de la figure (a), nous avons voulu montrer l'inadaptation des environnements de CAO et de leurs interfaces actuelles à l'informatique mobile, et ce que nous appelons donc la *conception mobile*. (la figure (b) est tirée de [Masry *et al.*, 2004])

A contrario, une application telle que SKETCHBOOK PRO [Alias, 2005a] a été orientée vers la créativité, par une métaphore de carnet de dessins sur un TabletPC (mais sans interpréter ces dessins).

De la même manière, le système de modélisation 3D proposé dans [Masry *et al.*, 2004] est présenté avec un TabletPC. Toutefois, il n'offre pas de réels avantages par rapport à l'utilisation d'une tablette écran, si ce n'est la mobilité. Le stylet est utilisé comme une souris et ses capacités étendues ne sont pas prises en compte (pression, inclinaison). Si l'on écarte des apports indéniables au niveau algorithmique, le système n'apporte pas de réelle évolution sur les aspects interaction par rapport à ses prédécesseurs tels que *Digital Clay*.

SVALABARD nous semble par contre très approprié pour une application à la conception créative mobile. Sa configurabilité permet son adaptation rapide et interactive aux entrées « simplifiées[(3)] » d'un TabletPC, tout en conservant ses principaux atouts tels que les feuilles d'interaction, les calques de dessin et les interactions associées (voir figure 5.2). D'une table à dessin virtuelle, SVALABARD peut alors rapidement devenir un *carnet de croquis 3D*. Mais cette configurabilité du système permettrait aussi de connecter des dispositifs tels qu'une webcam ou un micro pour capturer et insérer des sons et images du site. Dès lors, la combinaison avec un outil de reconstruction 3D à partir d'images et de documents (tel que celui que nous avons proposé dans [Huot, 2000; Hégron *et al.*, 2000]) offrirait des perspectives nouvelles pour les premières phases de la conception de nouveaux bâtiments directement dans l'environnement auquel ils sont destinés, à même le site.

(a) (b)

FIGURE 5.2 – SVALABARD sur TabletPC.

5.4.4 Conception augmentée et collaborative

D'autres perspectives de recherche intéressantes pouvant tirer partie de nos propositions portent sur la *conception en réalité augmentée* et la *conception collaborative*. Par réalité augmentée, nous entendons aussi bien des environnements que l'on pourrait qualifier de « naturels », tels qu'un bureau virtuel comme celui proposé dans [Safin *et al.*, 2005] (voir figure 5.3(a) page suivante) ou plus complexes, faisant intervenir des appareillages et des interactions plus avancées comme dans [Broll *et al.*, 2004] (voir figure 5.3(b)). Ces mêmes environnements et plateformes matérielles ou logicielles interviennent aussi dans le cadre des systèmes informatiques pour le travail collaboratif.

[(3)]Nous entendons simplifié par rapport à l'environnement instrumenté complet proposé pour SVALABARD.

(a) EsQUIsE Virtual Desk. (Photographie ©LUCID Group 2004)

(b) ARTHUR. (Photographie ©Fraunhofer FIT 2004)

FIGURE 5.3 – Conception augmentée et/ou collaborative. La figure (a) présente les derniers travaux menés par le LUCID group sur le bureau virtuel EsQUIsE. La figure (b) est un système de conception en réalité augmentée conçu au FRAUNHOFER INSTITUTE.

Dans une démarche de conception architecturale individuelle comme nous l'avons supposée jusqu'à maintenant, nous avons souligné l'importance que tenait l'environnement et la métaphore qui émerge du système informatique. Pour améliorer l'espace de travail, il serait intéressant d'étendre la métaphore de table à dessin virtuelle que nous avons proposée à un *bureau virtuel* comme celui proposé par le LUCID group avec EsQUIsE. Les techniques de manipulation directe utilisées dans SVALABARD prendraient alors tout leur sens et pourraient être poussées encore plus loin dans la métaphore : feuilles multiples sur le bureau, pliage des calques et déplacements des feuilles directement avec la main, etc.

Mais un tel espace de travail est aussi un atout pour intégrer complètement l'outil informatique dans les étapes exploratoires de la démarche créative. Comme nous l'avons précisé dans le premier chapitre, la recherche et la manipulation de la connaissance sont des points importants de la créativité. Ce principe de bureau virtuel favoriserait, par l'espace disponible, la manipulation de documents hétérogènes pouvant être utilisés dans la démarche créative (plans, images, modèles 3D, sons, etc.) ainsi que la collaboration directe entre différents acteurs de la conception. SVALABARD pourrait alors devenir un composant d'un système de conception collaborative de plus grande envergure, inspiré par exemple des « Surfaces Augmentées » proposées par Jun REKIMOTO [Rekimoto et Saitoh, 1999] : utilisation de feuilles réelles et virtuelles, de documents tangibles ou numériques, d'objets physiques ou virtuels, etc., le tout dans un même environnement (voir figure 5.4 page ci-contre).

Le portage logiciel de SVALABARD vers de telles plateformes matérielles serait facilité par les fonctionnalités avancées de la boîte à outils sur laquelle il repose (pointeurs multiples, interactions avancées, gestion de nombreux dispositifs, etc.). L'accent pourrait alors être mis sur l'adaptation des techniques d'interaction proposées, ainsi que le prototypage de nouvelles fonctionnalités dans le cadre d'une démarche plus globale de conception.

(a) (b) (c)

FIGURE 5.4 – Les *Surfaces augmentées* de Jun REKIMOTO créent un environnement de travail collaboratif mêlant documents virtuels et physiques (figure (b)), et favorisant l'échange (l'« Hyper Drag » de la figure (c), par exemple, permet d'échanger des objets entre les différents espaces physiques)

5.4.5 Application à d'autres domaines

Comme nous l'avons précisé au début de ce mémoire, nous nous sommes orientés vers l'étude et l'application de nos travaux à la conception architecturale. Il s'en est alors suivi une démarche basée sur une étude et des hypothèses de conception liées à ce domaine. Toutefois, les lignes directrices que nous avons proposées se veulent le plus générales possibles, dans le but d'améliorer la conception des outils de modélisation 3D par le croquis dans les premières phases de la conception.

Quelles sont alors les pistes pour une adaptation de nos travaux à d'autres domaines de conception créative, telles que le design ou même l'art ? Les paradigmes d'interaction et la métaphore de SVALABARD évoquent le point commun à ces différents domaines : le dessin à main levée dans une démarche la moins contrainte possible. Nous sommes d'avis que l'adaptation de notre système à d'autres domaines passe en premier lieu par la spécification de traitements du dessin particuliers et adaptés à ce domaine, comme nous l'avons fait dans le cadre de la conception architecturale ; que ce soit à bas niveau (*filtrage* des traits, fusion des points et des segments) ou à plus haut niveau (pistes que nous avons proposées pour la détection des propriétés et leur propositions à l'utilisateur). Ainsi, une démarche identique à celle que nous avons menée, en se basant sur une étude de dessins spécifiques au domaine pourrait permettre le développement de nouveaux traitements du dessin et l'adaptation de SVALABARD. Sur ce point, la modularité et la flexibilité de notre approche de traitement du dessin est un atout majeur pour de telles évolutions, évitant une refonte globale du système. Et il en est de même pour l'utilisation d'éventuels périphériques d'entrées ou techniques d'interaction spécifiques à un domaine d'application.

5.5 Conclusion : substituer l'écran au papier ?

Sans envisager réellement ce changement dans l'état actuel des technologies, nous soulevons la question de l'utilisation et de l'utilité des outils informatiques dans les usages créatifs. Mais surtout, quel sens plus global donner à l'informatique dans les processus créatifs, d'un point de vue plus détaché que les considérations plutôt pratiques que nous avons faites jusqu'à maintenant.

Usages et informatique

Doit-on substituer l'écran au papier pour changer des usages que nous considérons comme dépassés ?
La question n'est pas de savoir si les méthodes actuelles sont archaïques et méritent d'être dépoussiérées. Si l'on reprend le cas du dessin dans la démarche créative, une telle vision reviendrait à ne le considérer que comme une *technique*. Il ne serait alors pas dénué de sens de proposer une évolution de cette technique avec de nouveaux supports et de nouveaux buts. Et c'est ce que la majeure partie des travaux sur les interfaces de dessin ont fait jusqu'à présent : puisque les concepteurs « dessinent bien », proposons leur un stylet et la possibilité de tracer des traits ; mais les traits que l'on veux et que notre système sait traiter. Nous ne nions pas pour autant le fait que ces approches améliorent la prise en main et le rapport intuitif entre l'utilisateur et le système.

Par contre, une telle démarche ignore totalement la qualité de *compagnon* que tient le dessin aux cotés du concepteur. Plus qu'une simple technique, c'est un collaborateur engageant un dialogue comme nous l'avons exposé dans le chapitre 1. C'est cette *« decontextualisation »* systématique de la tâche que nous avons tenté d'éviter dans notre approche de dessin libre, en visant plutôt la *« contextualisation »* du système informatique dans la tâche de l'utilisateur. Dans ce sens, une vision future de l'évolution de SVALABARD serait d'aller jusqu'à la suppression complète de tous ses composants matériels (périphériques, écran, etc.) au profit de feuilles de *papier électronique* capables de numériser les dessins, et de projecteurs holographiques permettant l'affichage *virtuel* des modèles créés. Nous passerions alors d'une métaphore de *table à dessin virtuelle* à une *table à dessin augmentée*, ou le système informatique serait véritablement transparent et ses apports directement intégrés à un *environnement de conception enfoui* (« ubiquitous computing »).

Choix des usages

Si l'on veut substituer l'écran au papier, quels moyens avons-nous pour considérer que les usages ne sont pas adaptés et que nos techniques innovantes sont meilleures ?
Bien que nous ayons essayé d'adopter une vision des plus détachées tout au long de nos travaux, il n'en reste pas moins évident que nous avons souvent évolué autour d'un domaine et d'un raisonnement assez technique et pratique. Et ce, même lors de notre prise de connaissance des processus *humains* de la démarche créative. Comment alors identifier les véritables apports d'un outil informatique dans les premières phases de la conception autrement que par rapport à nos propres références ? Comment vérifier nos propres assertions sur l'intégration des techniques informatiques dans la démarche créative ?

Cette réflexion nous a fait prendre conscience que les utilisateurs de l'outil seront toujours les plus à même de savoir si celui-ci leur convient, et surtout de le modeler en fonction de leurs besoins et usages. Cette notion de *construire* l'outil est elle aussi une activité créative de la conception[(4)], impliquant un rapport à l'outil et à la tâche bénéfique pour le confort de travail et la créativité.

[(4)] C'est pour cette raison que les ergonomes considèrent la créativité comme présente à tous moments d'un processus de conception, lorsque par exemple un opérateur de machine-outil va modifier son appareillage pour permettre de façonner un nouvel objet.

Façonner et adapter l'outil créatif

SVALABARD introduit des notions nouvelles de flexibilité, de configurabilité et d'adaptabilité dans le domaine des systèmes informatiques de support à la créativité, ouvrant selon nous la voie vers de véritables *outils* informatiques de conception. En effet, en rendant possible la redéfinition complète de ses interactions, de ses comportements mais aussi de ses fonctionnalités, nous envisageons de permettre au concepteur de l'adapter à ses méthodes et ses besoins. Remplacer des interactions, des dispositifs par d'autres, déconnecter des fonctionnalités, en connecter d'autres, tout cela de manière interactive et visuelle, est une des particularités de SVALABARD, rendue possible par la boîte à outils MAGGLITE sur laquelle il repose (et que nous développerons dans la deuxième partie).

Ce principe, pouvant être assimilé à de la programmation visuelle, est proche de systèmes très utilisés dans le domaine de la MAO[(5)], avec par exemple MAX/MSP [Puckette, 1991] et les différents systèmes plus complets qui en découlent : JITTER permet à l'utilisateur de construire sa propre application de traitement et de mélange de sons, de vidéos ou de graphismes à partir des greffons disponibles dans la bibliothèque [Cycling'74, 2004]. L'utilisateur construit sa propre chaîne de traitement des données, adaptée à ses besoins et au(x) but(s) qu'il veut atteindre.

Une autre perspective serait d'approfondir nos travaux pour étudier de telles pratiques dans les domaines de la conception créative. Il est d'ores et déjà possible de composer des traitements du dessin, mais surtout de définir les interactions et comportements de SVALABARD (ce qui est relativement nouveau par rapport aux outils de MAO que nous avons évoqués). L'ajout de nouveaux filtres, mais aussi de fonctionnalités nouvelles reposant sur le dessin (peinture, bande dessinée, dessin technique, etc.) sous forme de greffons ferait de SVALABARD une plateforme de composition d'applications *créatives* de dessin, permettant à l'utilisateur de définir lui-même l'application appropriée à ses besoins, de façonner son *outil*.

C'est dans ce sens que dans le chapitre 2 de ce mémoire nous avions émis des réserves quant à la notion d'usage *transgressif*, selon le terme employé par Daniel ESTEVEZ dans [Estevez, 2001] (détourner un système informatique de son but premier, ou n'en utiliser que les fonctionnalités en adéquation avec la démarche créative). Car en effet, notre position sur ce point tend plutôt à fournir à l'utilisateur les moyens d'intégrer à son cheminement les différentes fonctionnalités dont il a besoin, lui permettant alors de créer son environnement privilégié. Et c'est l'une des perspectives selon nous les plus prometteuses de notre approche, de par la démarche de conception que nous avons suivie, mais aussi grâce à l'architecture logicielle que nous proposons.

[(5)] Musique Assistée par Ordinateur

Deuxième partie

MAGGLITE, une boîte à outil post-WIMP avancée.

Chapitre 6

Une évolution nécessaire

« J'ai inventé une lampe de poche qui fonctionne à l'énergie solaire, elle n'a qu'un dernier défaut, elle ne marche qu'en plein soleil »

A. FRANQUIN.

Sommaire

6.1 Introduction

Lorsque nous avons débuté le développement de SVALABARD, nous nous sommes rapidement heurté à des problèmes d'ordre technique. Ces problèmes d'implémentation ne remettaient pas en cause notre approche et ses principes, mais les rendaient difficilement réalisables avec les outils logiciels actuels pour trois raisons majeures :

1. Tout d'abord, l'environnement graphique de SVALABARD repose sur des techniques avancées, mêlant des effets visuels tels que transparences, déformations, animations, etc.
2. Ensuite, la métaphore de table à dessin naît en partie de l'usage de dispositifs d'entrée avancés liés à de multiples techniques d'interaction non-standard (reposant en majeure partie sur le dessin).
3. Enfin, il nous parait essentiel pour une telle approche novatrice de s'appuyer sur une architecture flexible facilitant le prototypage (afin de pouvoir affiner et remettre en question nos choix lors de la conception du système), l'adaptabilité et l'accessibilité (lors de l'utilisation) et l'évolution.

Ces trois points (modèle graphique, dispositifs d'entrée/techniques avancées et adaptabilité) ont souvent été abordés dans le domaine des boîtes à outils d'interaction et de nombreuses solutions y ont été apportées. Pourtant, ils n'ont que rarement été considérés sous un angle commun permettant d'unifier leurs solutions. Dès lors, il n'existe pas d'outil regroupant les solutions proposées dans un même modèle d'architecture logicielle et à plus forte raison dans une boîte à outils unique.

Nous aurions pu, en nous appuyant sur les outils logiciels actuels, apporter une solution particulière et dépendante de nos besoins pour l'implémentation de SVALABARD. Cette approche monolithique aurait demandé un gros effort de programmation. Nous avons jugé plus important, essentiellement pour les perspectives que cela offrait, d'unifier ces solutions dans un modèle d'architecture logicielle cohérent et de le réaliser au niveau d'une boîte à outils, indépendante des applications. Ce choix ne garantissait pas moins d'efforts et un développement plus rapides, la réalisation de la boîte à outils MAGGLITE ayant nécessité un lourd travail de développement. Par contre, l'implémentation de SVALABARD en a été considérablement réduite, nous permettant alors de concentrer nos efforts sur sa conception. Cette approche de développement parallèle nous a de plus permis d'avoir un exemple concret d'utilisation de la boîte à outils, mais aussi d'en raffiner les mécanismes pour des besoins applicatifs réels.

Dans ce court chapitre introduisant la deuxième partie de nos travaux, nous aborderons la situation actuelle vis à vis des trois points cités plus haut au travers d'un rapide survol des boîtes à outils les plus représentatives (pour un état de l'art plus complet, nous renvoyons le lecteur au chapitre 2 de la thèse de doctorat de Pierre DRAGICEVIC [Dragicevic, 2004b]).

6.2 Supporter l'interaction avancée

Nous considérons trois catégories d'outils de développement d'applications interactives :

1. les boîtes à outils *WIMP* ;
2. les boîtes à outils *WIMP avancées* ;
3. les boîtes à outils *post-WIMP spécialisées*.

Tous s'attachent à offrir un modèle graphique bien défini et générique, relativement complet et évolutif. Il n'en est toutefois pas de même pour la gestion des entrées et de l'interaction, chacune de ces approches raffinant ou proposant des modèles pour des objectifs différents.

6.2.1 Architectures graphiques

Du point de vue graphique, de grands progrès ont été faits dans les boîtes à outils d'interaction. La grande majorité d'entre elles proposent des modèles clairs, basés sur des abstractions simplifiant la gestion de l'affichage et permettant des effets visuels toujours plus sophistiqués. L'impact sur l'interaction est alors évident, ces architectures permettant de réaliser plus simplement des techniques de manipulation directe ou d'innover dans l'interaction grâce à des effets visuels avancés (transparences, déformations, animations, etc.) [Roussel, 2002].

Nous verrons en particulier, dans la section 7.2 page 159 du chapitre suivant, l'intérêt des *graphes de scène*, modèle issu de la 3D et de plus en plus utilisé pour les boîtes à outils 2D. Mais bien que modulaires et flexibles pour l'utilisation ou la création d'objets graphiques, la réalisation de ce type d'architecture dans les boîtes à outils actuelles lie encore trop fortement la description de la partie graphique et de l'interaction, par encapsulation dans des widgets et dans une architecture à événements. Il est alors difficile de découpler ces deux aspects, ce qui ne favorise pas la flexibilité et l'évolution de la boîte à outils sur les deux tableaux.

Ainsi, pour introduire de nouvelles techniques d'interaction avancées ou en utiliser plusieurs dans une même application (comme SVALABARD), il est encore nécessaire de modifier plus ou moins en profondeur les abstractions et objets graphiques de la boîte à outils.

6.2.2 Périphériques d'entrée et interactions non-standard

Les boîtes à outil WIMP

Les boîtes à outils WIMP, de **X Toolkit** [McCormack et Asente, 1988] à **Java Swing** [Eckstein et Loy, 2002], sont les plus standards et probablement les plus utilisées actuellement. Mais elle ne permettent en l'état que la réalisation d'applications WIMP, postulant dès lors que les dispositifs d'entrée utilisés seront une souris et un clavier. Elles n'offrent qu'un support dit *par compatibilité* des dispositifs d'entrée avancés, reposant sur la gestion qu'en offre le système d'exploitation, sans tirer partie de leur capacités (ils sont vus comme une souris).

Basées sur une architecture à événements fortement liée aux données reçues des dispositifs standard, elles sont alors très difficile à étendre pour le support de périphériques et de techniques d'interaction avancées, de par l'encapsulation de l'interaction dans les widgets.

Les boîtes à outil WIMP avancées

Les boîtes à outils WIMP avancées, comme **Subarctic** [Hudson et Smith, 1996] (évolution de **Artkit**) et **Garnet/Amulet** [Myers, 1990; Myers *et al.*, 1997], par exemple, ont pour objectif de faciliter la construction d'interfaces WIMP (ou à manipulation directe) et d'améliorer l'insertion de techniques moins conventionnelles dans ce type d'interfaces. L'effort majeur porte sur l'architecture,

dans le but de fournir l'extensibilité faisant défaut aux boîtes à outils standards. Les mécanismes proposés ont favorisé quelque peu l'ajout de nouvelles techniques et l'extension des boites à outils : *interaction gestuelle*, *champs de gravité* et *lentilles sémantiques* dans *Subarctic* [Tyson R. *et al.*, 1990; Hudson *et al.*, 1997] ainsi que les entrées ambiguës (association *gestes/parole*) [Mankoff *et al.*, 2000] ; *Amulet* améliore entre autre *Garnet* par l'ajout d'interaction gestuelle [Myers *et al.*, 1997], etc.

Ces approches ont permis d'étendre et d'améliorer notablement l'architecture standard basée sur les événements, en décrivant à plus haut niveau leur lien avec les widgets (les interacteurs de *Garnet*) ou en explicitant mieux les mécanismes pour leur aiguillage (les *Dispatch policies* et *Dispatch agents* de *Subarctic*). Mais elles restent encore relativement fermées aux techniques d'interaction avancées des interfaces post-WIMP, nécessitant encore un travail de fond et des changements importants pour insérer de nouveaux paradigmes d'interaction dans leurs architectures. Pour la gestion des entrées, elles étendent les mécanismes de *AWT* (pour *Subarctic*) ou de *X* (pour *Garnet*) et ignorent donc de fait les particularités des dispositifs d'entrée avancés ou leur multiplicité (pointeurs multiples, interaction bimanuelle, etc.).

Les boîtes à outil post-WIMP

Les boîtes à outils post-WIMP sont conçues pour prendre en compte et décrire des paradigmes d'interaction non-conventionnels. Mais bien qu'elles spécifient et implémentent de manière claire des modèles et des architectures assez génériques, elles restent *spécialisées* dans un paradigme d'interaction. Ainsi, **Satin** [Hong et Landay, 2000] se focalise sur l'interaction gestuelle, **MMM** [Bier et Freeman, 1991], **Whizz** [Esteban, 1997; Chatty, 1994] et **MID** [Hourcade et Bederson, 1999] sur les pointeurs multiples (travail collaboratif et interaction bimanuelle), les **Phidgets** [Greenberg et Fitchett, 2001] et **WidgetTap** [Greenberg et Boyle, 2002], ou **iStuff** [Ballagas *et al.*, 2003] pour les interfaces tangibles, etc.

Un cas particulier est celui de **CPN2000/CPN Tools** [Beaudouin-Lafon et Lassen, 2000], une application complète d'édition et de simulation de réseaux de Pétri colorés qui combine manipulation directe, *Marking Menus*, palettes d'outils semi-transparentes, etc. Bien que n'étant pas une boîte à outils, *CPN2000* démontre la possibilité de faire cohabiter ces différentes techniques dans une architecture claire en utilisant un modèle d'interaction adapté (l'*interaction instrumentale* [Beaudouin-Lafon, 2000]).

À l'exception de certaines boîtes à outils 3D et leurs modèles différents de gestion des entrées (*Virtools* [Virtools SA, 2001], par exemple, utilise un modèle à canaux pour la connexion statique ou dynamique de nombreux dispositifs d'entrée), les boîtes à outils restent limitées dans l'exploitation des entrées enrichies, reposant souvent sur des architectures à événements traditionnelles. Même lorsqu'elles sont spécialisées dans un paradigme fortement lié aux entrées (comme les pointeurs multiples), les dispositifs sont exploités de façon *ad hoc*.

Synthèse

Dans un souci de généralité et de portabilité, les boîtes à outils standards ont rendu rigide et stéréotypé le développement d'applications interactives. Ainsi, bien que des boîtes à outils WIMP avancées aient introduit des architectures et des modèles plus souples, elles n'en restent pas moins relativement inefficaces pour supporter les nouveaux paradigmes d'interaction, ainsi que les entrées

non-standard. Leurs extensions dans ce sens se font souvent au prix d'un remaniement plus ou moins profond de leur architecture et de leur implémentation. Cette tendance s'observe d'ailleurs par le nombre élevé de publications portant sur l'ajout d'un nouveau style d'interaction dans une boîte à outils existante (des gestes [Henry *et al.*, 1990] à l'interaction multimodale [Mankoff *et al.*, 2000] dans *Subarctic*, par exemple). Bien que cette approche apporte des solutions aux problèmes de généricité des modèles d'implémentation actuels ou permette d'en formaliser certains aspects, elle n'est pas à notre avis une solution aux problèmes de la flexibilité et de la modularité nécessaires à la cohabitation et au prototypage de nouvelles techniques. Elle se concentre plus sur la conception des interfaces que celle des interactions [Beaudouin-Lafon, 2004].

Du point de vue des outils pour le développement d'application post-WIMP, les solutions proposées ont pris le chemin contraire, en proposant des solutions adaptées à un cadre, un paradigme précis. Dès lors, construire une application qui tire partie de différentes techniques d'interaction avancées, tel que nous le voulions avec SVALABARD, nécessite de réaliser des assemblages plus ou moins cohérents de ces différents modèles (en écrivant une application basée sur plusieurs boîtes à outils) ou de tout redévelopper autour d'un modèle d'interaction particulier à l'application (comme dans *CPN2000/CPN Tools*).

Cette hétérogénéité n'est pas pour autant contestable, du fait de la complexité évidente à faire cohabiter des paradigmes parfois contradictoires. Toutefois, il nous semble que c'est surtout les modèles d'architecture logicielle proposés qui, bien qu'ils soient souvent très avancés et approfondis, sont figés par l'utilisation d'une architecture standard à événements. Cette association limite souvent l'emploi de dispositifs non-standard et nécessite des extensions trop complexes pour prendre en compte les données ou flux propres aux interactions avancées (commandes, nombreux degrés de libertés, dispositifs multiples, etc.).

L'utilisation d'un modèle concret d'implémentation suffisamment flexible et générique pour faire cohabiter différentes techniques et dispositifs dans un même environnement de développement permettrait de reporter ce contrôle de la cohérence lors de la création des applications. Le problème ne serait alors plus d'ordre technique mais d'utilisabilité de l'interface, aspect primordial de la conception des IHM. Il est alors nécessaire de proposer une approche plus flexible qui rend graphismes, interactions et dispositifs d'entrée réellement indépendants tout en supportant les outils et mécanismes permettant l'évolution et le prototypage.

6.3 Prototypage et configuration visuels

Que ce soit pour simplifier le travail des programmeurs, fournir à des concepteurs d'interfaces non programmeurs (graphistes, etc.) des outils adaptés à leurs compétences ou même permettre à l'utilisateur final de paramétrer et de personnaliser son application, la communauté de recherche en IHM a proposé beaucoup d'outils graphiques pour la conception d'interfaces.

Cette préoccupation était aussi l'une des nôtres lors de la réalisation de SVALABARD : pouvoir prototyper rapidement des interactions avancées et novatrices, mais aussi, du fait de l'emploi de techniques non-standard, permettre à l'utilisateur de pouvoir forger son propre outil pour une transition plus « douce » et personnelle (adapter le système aux dispositifs, changer les techniques d'interaction, personnaliser les comportements, etc.).

Pourtant, si de nombreux outils permettent de décrire l'apparence des interfaces de manière gra-

phique (comme par exemple **Visual Basic** pour les interfaces WIMP ou des boîtes à outils utilisant des documents graphiques issus de logiciels de dessin pour les interfaces avancées [Chatty *et al.*, 2004]), ils nécessitent toujours d'avoir recours à la programmation pour décrire les comportements de l'application. Des approches plus évoluées, notamment avec *Garnet/Amulet* [Myers, 1990] et ses nombreux constructeurs d'interfaces basés sur la programmation par démonstration (**Lapidary** [Vander Zanden et Myers, 1995], par exemple), ont permis d'aller plus loin dans la description graphique des comportements, mais leur champ d'application reste relativement limité aux interfaces WIMP dans des domaines bien particuliers.

En plus de leur peu d'ouverture aux techniques d'interaction avancées, il est évident que de telles approches n'étaient pas envisageables pour notre application. Car même si elles avaient facilité son développement, elles n'auraient pas permis le haut niveau de description et de configurabilité désiré pour les interactions et les filtres de traitement du dessin.

Pierre DRAGICEVIC a montré dans sa thèse que les systèmes basés sur un modèle à flot de données offraient non seulement plus de flexibilité au niveau de l'architecture, mais aussi une aptitude certaine à la description graphique des entrées et des interactions. En approfondissant les approches existantes basées sur ce principe (éditeurs de comportements pour la 3D, l'éditeur **Whizz'Ed** [Esteban, 1997]), il a proposé la boîte à outils **ICON** comme solution pour une partie des problèmes que nous avons évoqués [Dragicevic et Fekete, 2001; Dragicevic, 2004b].

6.4 ICON : une solution pour l'adaptabilité en entrée

ICON est une boîte à outils, programmée en Java, complétée par un éditeur interactif permettant de créer des applications interactives hautement configurables, c'est à dire des applications contrôlables avec un grand nombre de périphériques d'entrée et de techniques d'interaction non-standard. ICON introduit une architecture réactive en flot de données qui permet de décrire des configurations d'entrée à partir de modules interconnectés en cascade. ICOM, le modèle d'ICON, est basé sur les *dispositifs* (figure 6.1), généralisation des dispositifs d'entrée : ils peuvent produire des valeurs de sortie, mais peuvent aussi en recevoir.

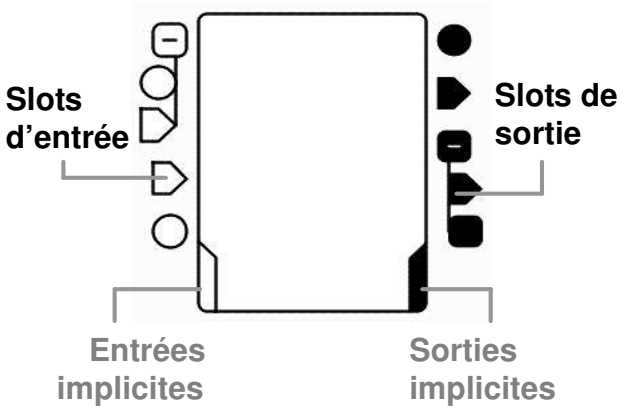


FIGURE 6.1 – Un dispositif ICON avec ses slots d'entrées (à gauche) et ses slots de sortie (à droite).

6.4.1 Dispositifs et slots

Un dispositif est une « boîte noire » qui interagit avec son environnement par l'intermédiaire de canaux typés, appelés *slots* (d'entrée ou de sortie). Un dispositif peut aussi communiquer avec l'environnement extérieur (autre que des dispositifs) par des *entrée/sorties implicites*, traduisant la réception ou l'émission de données par des flux d'informations autres que les slots d'entrée ou de sortie. Le dispositif est alors considéré comme *actif* (ou asynchrone).
Chaque dispositif peut également déclarer un ensemble de *paramètres* permettant de spécialiser son comportement. Dans le langage graphique du configurateur, chaque type de slot possède une représentation distincte (cercles pour les Booléens, triangles pour les entiers, etc.) et peuvent être groupés hiérarchiquement pour composer des types structurés.

Le modèle ICOM définit trois catégories de dispositifs : *les dispositifs système*, qui décrivent des ressources système comme les périphériques d'entrée (souris, claviers, tablettes, reconnaissance vocale, etc.) ; *les dispositifs utilitaires*, qui sont des dispositifs indépendants du système allant de simples opérateurs booléens jusqu'à des feedbacks ou interactions avancées tels que des curseurs, « toolglass » et interpréteurs de gestes ; *les dispositifs d'application*, qui sont les dispositifs qui contrôlent les objets de l'application (du domaine). Les dispositifs système et utilitaires sont fournis par la boîte à outils, alors que les dispositifs d'application doivent être conçus par les programmeurs de l'application.

6.4.2 Configurations d'entrée et exécution réactive

Un slot de sortie d'un dispositif peut être relié à un ou plusieurs slots d'entrée compatibles d'autres dispositifs par des connexions représentées par des lignes (voir figure 6.2 page suivante). Un ensemble de dispositifs connectés définis alors une *configuration d'entrée* exécutable par le noyau réactif d'ICON. Cette exécution se déroule en deux étapes :

1. le *lancement*, qui correspond à la préparation de la configuration d'entrée. Celle-ci est décomposée et les dispositifs sont ensuite *ouverts* afin d'être initialisés. Cette initialisation permet au noyau d'exécution de récupérer le *processeur* de chaque dispositif : son comportement en exécution (traitement et mise à jour des slots). Ces processeurs sont enfin triés topologiquement en fonction du graphe de dépendances des connexions.
2. l'exécution en pas atomiques ou *ticks*. Cet algorithme mets à jour les dispositifs dont les valeurs ont changées et propage ces changements dans les dispositifs qui en dépendent (connectés) en une seule passe grâce au tri topologique des dispositifs.

Ce modèle d'exécution, inspiré des langages réactifs impératifs, est basé sur le principe du synchronisme parfait [Berry, 2000], modèle dans lequel les processus sont capables au niveau conceptuel de s'exécuter et d'échanger des informations en un temps nul. Ce paradigme quelque peu déroutant pour les programmeurs « classique » est un standard pour les théoriciens du contrôle ou les concepteurs de circuits électroniques. Son adaptation à la gestion des entrées a ouvert la voie d'une gestion *réactive* de l'interaction, à un niveau où le standard des files d'évènements est peu adapté, garantissant ainsi une réaction immédiate aux actions de l'utilisateur. Mais plus que la simple connexion de périphériques d'entrée, la description d'interactions a aussi été largement explorée par l'auteur d'ICON dans le cadre de la spécification d'interactions avancées [Dragicevic, 2004b; Dragicevic et Fekete, 2004] ou de l'association de son système avec une approche orientée *contrôle* [Dragicevic *et al.*, 2004].

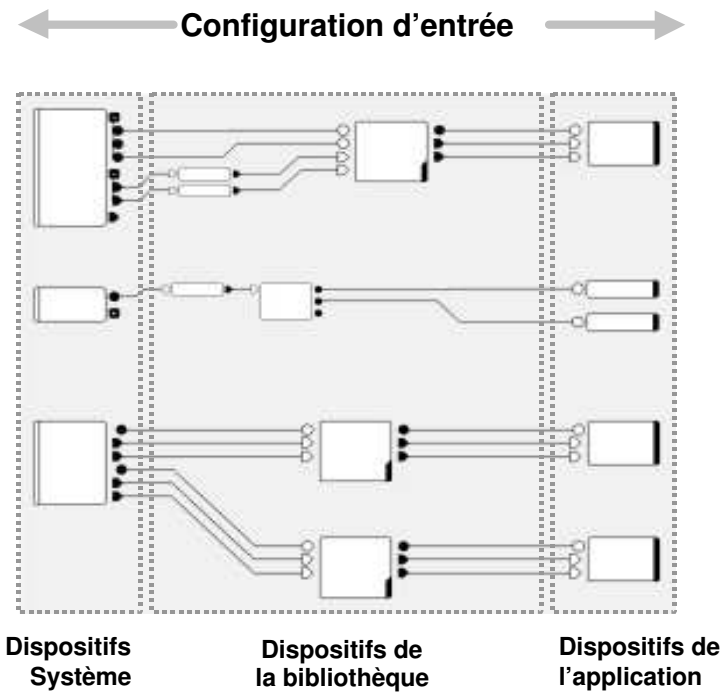


FIGURE 6.2 – Une configuration d'entrée décrit une manière de relier des dispositifs système à des dispositifs d'application.

La description de l'interaction avec ce modèle devient à la fois simple et flexible. Simple, parce qu'elle permet de représenter l'interaction de manière graphique et dans une logique linéaire, moins compacte que la propagation d'événements traditionnelle. La connexion d'un périphérique aux interactions de manière directe et visuelle à l'exécution du programme permet de se rendre compte rapidement du résultat en « suivant les données du regard ». Flexible, car les dispositifs peuvent êtres remplacés facilement par d'autres, à tous les niveaux (périphériques d'entrée, retours graphiques, techniques d'interaction, comportements). Ce paradigme de flot de données permet une description de l'interaction avec une granularité plus fine que par l'utilisation d'une file d'événements.

En effet un type d'événement spécifique est en général associé à chaque type de périphérique d'entrée. Ces événements sont aussi fortement liés aux interactions et objets de l'implémentation pour que ceux-ci sachent adapter leur comportement. Ainsi, pour introduire la gestion d'un nouveau périphérique (manette de jeu, stylet sensible à la pression, par exemple) ou de nouvelles interactions (gestuelles, vocales ou manipulations directes avancées), il faut :

1. définir un nouveau type d'événement ;
2. modifier et adapter le mécanisme de propagation des événements ;
3. étendre les objets qui doivent réagir à ces événements.

Avec ICON, il suffit de créer un dispositif pour gérer le nouveau périphérique ou d'implémenter une nouvelle interaction sous la forme d'un dispositif. Une fois cela fait, le périphérique ou la technique d'interaction peuvent être réutilisés dans nombre de configurations d'entrée.

L'éditeur graphique (voir figure 6.3) permet de mettre en correspondance les périphériques d'entrée et l'application de manière interactive. Les périphériques d'entrée disponibles sur le système ainsi que tous les dispositifs de la bibliothèque d'ICON sont organisés par *dossiers* et présentés sur un panneau. Il suffit de les placer dans la zone de configuration de l'éditeur pour les utiliser. La mise en correspondance met en œuvre l'insertion et la connexion de dispositifs qui décrivent des techniques d'interaction prédéfinies (par exemple, le dispositif de reconnaissance de geste est conçu pour être inséré entre un dispositif de pointage et un dispositif recevant du texte en entrée) aussi bien que la description de nouvelles techniques par la combinaison de dispositifs de traitement plus simples.

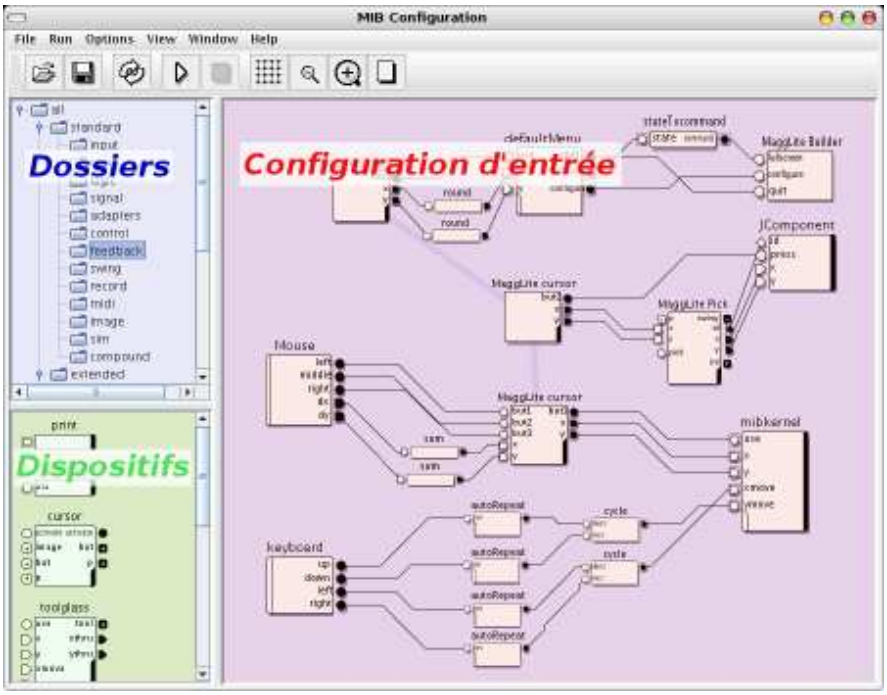


FIGURE 6.3 – *Input Configurator*. Le configurateur graphique d'ICON.

6.4.3 Niveau de contrôle des applications avec ICON

Pierre DRAGCEVIC a définit les différents niveaux de contrôle possible des applications avec le modèle ICOM :

1. **Le contrôle par défaut**, niveau de contrôle par défaut d'une application « hors ICON », en conservant la gestion des entrées effectuée implicitement au niveau système et boîte à outil, et en l'étendant ou non.
2. **Le contrôle générique de surface**, stratégie de contrôle positionnel des widgets de boîte à outils, à son niveau le plus haut (sélection de widgets, focus, clic par exemple).
3. **Le contrôle générique de modèle**, stratégie de contrôle spécifique à un type de widget, décrivant les interactions avec une classe de widget existante (pouvant donc court-circuiter les méthodes d'entrée de la boîte à outils).
4. **Le contrôle dédié**, la stratégie la plus puissante décrivant le contrôle complet d'une application par des *dispositif d'application* dédiés.

Il a aussi introduit la notion de « *dispositifs de boîte à outil graphique, situés à mi-chemin entre la catégorie des dispositifs utilitaires et celle des dispositifs d'application, (ils) permettent un contrôle générique de ces dernières* » [Dragicevic, 2004b]. Les dispositifs réalisés alors ont permis le *contrôle de surface* et le *contrôle de modèle* avec des boîte à outils comme Swing ou Jazz. Par contre, atteindre le *contrôle dédié* nécessite de développer des applications prévues pour et d'externaliser leurs outils sous la forme de *dispositifs d'application*.

Fort de ce modèle réactif, d'une bibliothèque étendue et extensible de dispositifs, et d'un configurateur graphique, ICON permet de décrire de façon homogène la chaîne de gestion des entrées des périphériques jusqu'à l'application, avec un grand pouvoir d'expression (des interactions les plus standards aux plus avancées). En outre, ce paradigme implique un haut niveau de *configurabilité* des applications, renforcé par sa réalisation dynamique qui permet la définition des configurations d'entrée à l'exécution de l'application. C'est ainsi une première réponse aux problèmes que nous avons soulevés dans ce chapitre : permettre l'utilisation de dispositifs d'entrée non standard et de techniques d'interactions avancées dans un même environnement, tout en offrant des outils de configuration interactifs et dynamiques.

6.5 Conclusion

Afin de répondre efficacement à nos besoins, mais aussi de résoudre une partie des problèmes des outils de développement actuels, nous allons introduire une nouvelle boîte à outils : MAGGLITE. Basée sur un modèle d'architecture logicielle original, et un traitement des entrées en rupture avec les architectures à événements traditionnelles, ce nouvel environnement permet la description et l'intégration dynamique de techniques d'interaction avancées, la gestion de dispositifs non-standard et le prototypage rapide d'interfaces post-WIMP (à l'aide d'outils graphiques ou par programmation simplifiée).

De ICON à MAGGLITE

Le modèle d'architecture logicielle que nous allons présenter dans le chapitre suivant prend ses fondements dans le modèle ICOM. Nous avons, afin de combler nos besoins pour la réalisation de SVALABARD, mais aussi pour explorer plus avant les possibilités d'ICON, associé ce modèle à une architecture graphique basée sur les graphes de scène. Il a alors été nécessaire de faire le lien entre les notions de *configuration d'entrée* et de *graphe de scène*, afin de permettre un contrôle maximal des objets graphiques et des outils de l'application.

C'est dans cette optique que nous proposons une architecture logicielle originale pour les boîtes à outils post-WIMP : les *Graphes Combinés*.

Chapitre 7

Les graphes combinés

« Le hockey sur glace est un savant mélange de glisse acrobatique et de seconde guerre mondiale »

A. HITCHCOCK.

Sommaire

7.1 Introduction

Dans ce chapitre, nous introduisons un modèle d'architecture logicielle original pour la conception des Interfaces Homme-Machine, ainsi que les principes de son implémentation dans la boîte à outils MAGGLITE. Cette architecture d'implémentation, les ***« Graphes Combinés »***, étends et affine l'architecture graphique des graphes de scène en en soustrayant la description des interactions sous la forme de *graphes d'interaction*.

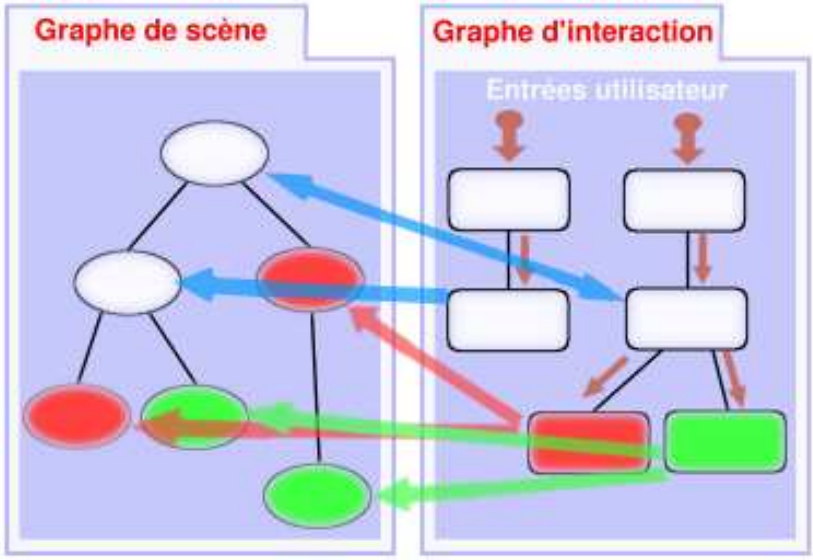


FIGURE 7.1 – Les graphes combinés. Un graphe de scène (à gauche) et un graphe d'interaction (à droite) sont combinés à l'exécution. Certains nœuds du graphe d'interaction communiquent avec des nœuds compatibles du graphe de scène (couleurs équivalentes).

La partie graphique de cette architecture repose sur un ***graphe de scène***. Introduits dans le développement des boîtes à outils 3D et devenus incontournables dans ce domaine, les graphes de scènes sont de plus en plus utilisés pour l'affichage en deux dimensions, tout spécialement dans des boîtes à outils ou applications avancées telles que **Jazz/Piccolo** [Bederson *et al.*, 2000; Bederson *et al.*, 2004], **CPN Tools** [Beaudouin-Lafon et Lassen, 2000] ou **UBIT** [Lecolinet, 2003]. Un graphe de scène décompose la structure monolithique des architectures basées sur les « widgets » en utilisant des objets graphiques de granularité plus fine. Ces objets de base peuvent alors être regroupés afin d'en composer des plus complexes. Chaque nœud, ou groupe de nœuds, encapsule alors ses comportements et interactions, interactions reposant le plus souvent sur une architecture à événements.

Le principe des graphes combinés va plus loin en ne décomposant pas seulement la partie graphique, mais aussi les interactions. Celles-ci sont décrites par des ***graphes d'interaction*** qui décomposent la gestion événementielle en graphes de flot de données. Ces graphes sont des configurations d'entrée d'ICON, composées de *dispositifs*. Le graphe d'interaction met à jour la structure et les composants du graphe de scène de manière dynamique, lorsque cela est nécessaire (voir figure 7.1). Ces communications entre les deux graphes peuvent être déclenchées par des entrées de l'utilisateur ou des événements système.

Il découle de ce modèle des *graphes combinés* une séparation claire entre l'apparence et les com-

portements de l'interface, telle qu'elle est décrite dans le modèle MVC [Schmucker, 1986; Krasner et Pope, 1988]. Mais surtout, les graphes d'interaction peuvent être construits graphiquement de nombreuses manières, pendant l'exécution. Il est ainsi possible de recomposer les associations vue/contrôleur pour adapter l'interface ou tester des techniques d'interaction alternatives sans avoir à programmer, ni même à relancer l'application.

Cette approche, moins compacte que la traditionnelle propagation d'événements, permet un haut niveau de flexibilité dans la description des interactions ainsi que des possibilités graphiques avancées. Cela fait de la boîte à outils MAGGLITE une plateforme de développement adaptée au prototypage d'applications post-WIMP ou d'applications standards proposant des interactions avancées. Il est aussi simple pour le développeur d'implémenter rapidement de nouvelles interactions, que pour le concepteur d'interfaces de les assembler et les essayer ; même l'utilisateur peut adapter l'interface et les interactions à ses besoins ou possibilités.

Ce chapitre décrit le modèle des graphes combinés. Nous l'entamerons par une brève description de l'architecture des graphes de scène. Nous présenterons ensuite les graphes d'interaction. Cette description est essentiellement basée sur les *points d'accès aux interactions*, nœuds particuliers chargés de la liaison dynamique avec le graphe de scène. Afin de ne pas rendre cette présentation trop abstraite, mais aussi pour exposer de façon claire ses apports et la « philosophie » qui en découle, nous avons choisi d'y faire figurer les principes et mécanismes de base de la boîte à outils MAGGLITE qui en est issue (sa description complète fera l'objet du chapitre suivant page 179).

Certains des travaux présentés dans ce chapitre ont été publiés dans [Huot *et al.*, 2004b; Huot *et al.*, 2004a].

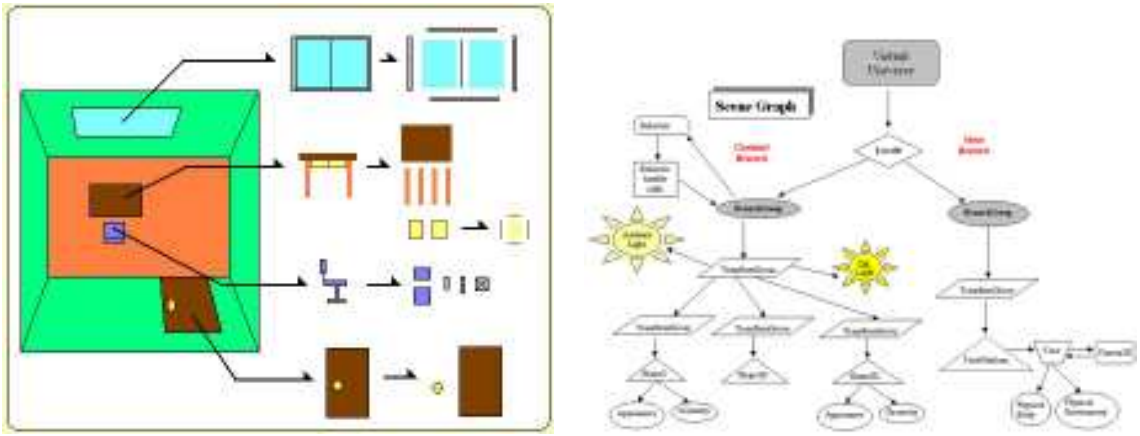
7.2 Graphes de scène

Les graphes de scène sont issus de l'architecture des boîtes à outils 3D du milieu des années 90 telles que *Open Inventor* [Wernecke, 1993; Silicon Graphics, Inc, 2005]. À l'époque où la programmation de scènes 3D demandait de solides notions en programmation avec des librairies graphiques de bas niveau, la solution des graphes de scène a permis la manipulation d'entités de plus haut niveau. Le principe est d'utiliser un graphe acyclique orienté pour organiser hiérarchiquement une scène 3D. Cette structure arborescente est manipulée par sa racine et chacun de ses nœuds est un élément de la scène à représenter. Naturellement, chaque nœud peut lui-même être un graphe de scène, ce qui entraîne une bonne organisation au niveau conceptuel, mais aussi sémantique en offrant plusieurs niveaux de granularité (voir figure 7.2 page suivante).

Parmi les avantages et possibilités de cette architecture, nous pouvons noter :

1. son adéquation avec les paradigmes de la programmation objet et une bonne structuration de la scène ;
2. l'optimisation et la simplification de la gestion de l'ordre d'affichage et du détourage (« clipping ») ;
3. la transmission et le partage aisés du contexte graphique (propriétés) entre nœuds parents et enfants.

Cette représentation structurée et ses avantages techniques qui ont permis de nettes améliorations des performances, ont vite poussé les graphes de scène au rang de standard des architectures de boîtes à outils ou de logiciels pour la 3D. Mais depuis peu, comme nous l'avons déjà évoqué plus haut, cette



(a) Une organisation sémantique.

(b) Une représentation complète.

FIGURE 7.2 – Graphes de scènes. La figure (a) présente un graphe de scène ne représentant que les objets géométriques de la scène. Cette visualisation permet d'appréhender clairement l'organisation de la scène. La figure (b) est une représentation complète d'un graphe de scène, faisant apparaître les transformations et autres nœuds graphiques.

architecture a commencé à être adoptée dans les boîtes à outils 2D, spécialement dans celles dédiées à l'interaction avancée. Nous avons nous aussi choisi cette structure pour ses propriétés de réutilisation et ses avantages du point de vue graphique, mais aussi avec l'idée de fond d'y associer une gestion avancée des interactions.

Un graphe de scène, dans le modèle des graphes combinés, est formé d'un objet racine, le bureau ou *desk*, et de nœuds, les objets graphiques de l'interface (voir figure 7.3 page suivante). Notre implémentation fournit différentes classes d'objets graphiques, la majorité d'entre eux pouvant être instanciés avec des formes diverses et arbitraires. Cette indépendance des objets vis à vis de leur forme permet plus d'expressivité dans la conception d'interfaces du fait que les objets ne sont pas limités à des formes rectangulaires ou ellipsoïdales. Leurs propriétés graphiques, telles que la couleur, les transformations géométriques ou l'opacité sont encapsulées par les objets et propagées aux nœuds fils par un mécanisme de contextes graphiques.

Les nœuds, ou objets graphiques, que nous proposons se décomposent en deux grandes classes :

1. les objets graphiques ***atomiques***, définissant plusieurs états ou propriétés : objets transparents, objets *pliables*, objets calques, etc.
2. les objets graphiques ***composites***, groupements d'objets atomiques (ou composites) dans un graphe de scène, encapsulant éventuellement un comportement prédéfini entre les objets atomiques.

Un exemple d'objet composite est la glissière de la figure 7.4 page ci-contre : les deux objets atomiques (le panneau et la poignée) ont un comportement prédéfini, la poignée ne pouvant être déplacée que le long de la « glissière ». Les « widgets » standard (boutons, glissières, fenêtres, etc.) peuvent être implémentés sous forme d'objets composites.

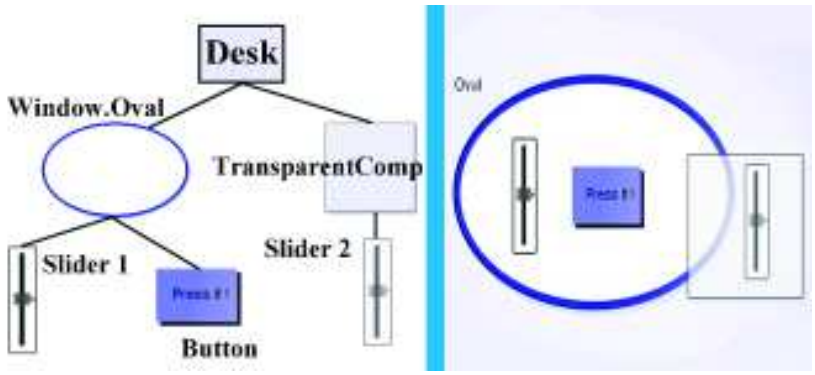


FIGURE 7.3 – Un graphe de scène simple et l'interface correspondante.

Les objets atomiques et composites de base que nous proposons sont extensibles en utilisant le mécanisme d'héritage, et nombre d'objets graphiques plus complexes peuvent être créés à partir des abstractions et interfaces fournies.

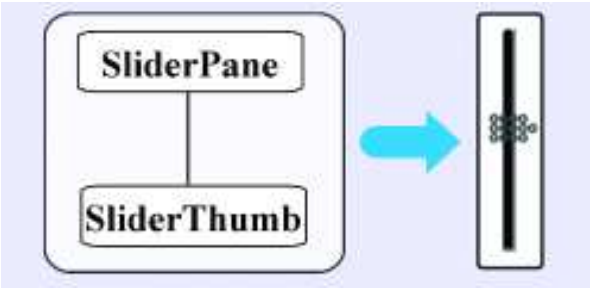
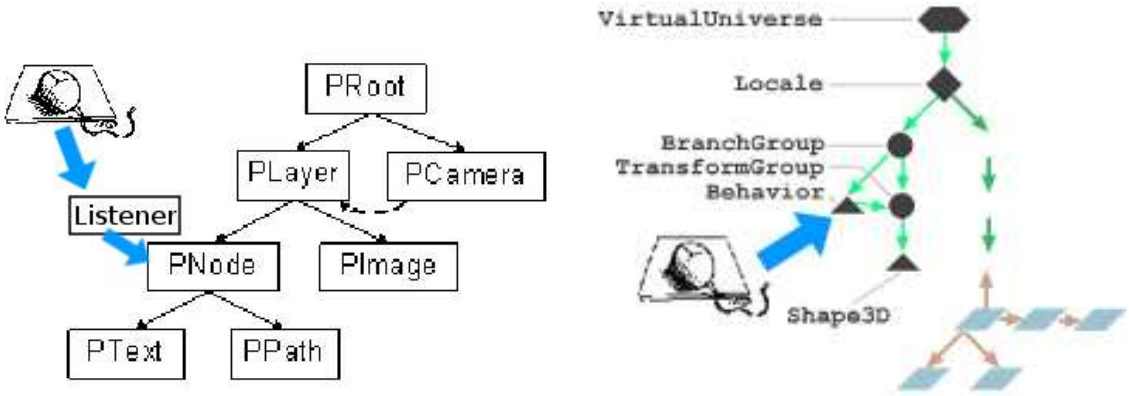


FIGURE 7.4 – Un objet composite décrit par un graphe de scène d'objets atomiques.

Nous passerons dans cette partie les méthodes proposées pour créer un graphe de scène MAGGLITE ; notons juste que, hormis la création de graphes de scène par programmation, nous proposons une méthode interactive basée sur le dessin dans la section 8.4.2 page 199, chapitre 8. Concentrons nous par contre sur les méthodes de description des interactions dans un graphe de scène. Dans les approches actuelles, l'interaction est décrite conjointement à la partie graphique par des nœuds spécifiques du graphe de scène, quand elle n'est pas tout simplement emmurée dans les nœuds graphiques eux-mêmes.

Les boîtes à outils Jazz et son évolution Piccolo développées au HCIL, par exemple, sont basées sur une structure très complète et avancée de graphes de scène [Bederson *et al.*, 2000; Bederson *et al.*, 2004]. Par contre, les interactions sont encapsulées au niveau des nœuds (objets graphiques) et basées sur l'architecture à évènements peu évolutive de la librairie Swing (voir figure 7.5(a) page suivante). Le principe est de créer des écouteurs qui peuvent alors être ajoutés au canevas de l'application (le conteneur principal du graphe de scène) ou directement à un nœud (pour la version .NET). De fait, même si cette boîte à outil innove sur les interfaces zoomables en offrant une grande latitude pour l'introduction de nouveaux composants graphiques, elle reste relativement *fermée* pour ce qui est des interactions avancées.

A contrario, la librairie graphique Java3D [Nadeau *et al.*, 1997; Sun Microsystems, Inc, 2004b] propose une classe spéciale de nœuds permettant de connecter des comportements (animations ou interactions) à des objets du graphe de scène. Cela permet de construire des graphes très complets et complexes, dont les sous-graphes peuvent fournir des comportements différents 7.5(b). Toutefois, ou plutôt encore une fois, bien que proposant une abstraction pour la gestion de périphériques d'entrée avancés, il est encore nécessaire de réaliser la liaison bas-niveau avec les périphériques non-standard (autres que souris et clavier) ainsi que les nœuds qui vont permettre les interactions avec le graphe de scène.



(a) Interactions dans Piccolo.

(b) Interactions dans Java3D.

FIGURE 7.5 – Graphes de scène et interactions. La figure (a) présente un graphe de scène de la boîte à outils Piccolo. Les interactions sont gérées par un mécanisme standard d'événements. Des écouteurs interprètent ses événements et sont connectés (statiquement) à des nœuds du graphe de scène. L'approche choisie pour Java3D (figure (b)) est de créer des nœuds spécifiques pour gérer les interaction et de les insérer aux branches voulues du graphe de scène.

Outre l'aspect statique de ces approches, qui impose de « brancher » et donc de figer les interactions à la création du programme[(1)], se pose aussi le problème de l'extensibilité. Malgré une granularité assez fine des composants permettant de définir les interactions, l'introduction d'interactions avancées, de périphériques nouveaux et leur réutilisation dans d'autres contextes reste laborieuse. Pour offrir plus de flexibilité et d'extensibilité, notre modèle de graphes combinés externalise la description des interactions avec les graphes d'interaction. Ainsi, les graphes de scène ne sont composés que de nœuds représentant des objets graphiques. Et de la même manière que la structure de graphe de scène sépare la partie graphique en composants graphiques primitifs, nous proposons d'utiliser des *graphes d'interaction* pour séparer la structure souvent complexe et monolithique des architectures événementielles.

[(1)]Ces approches permettent tout de même de brancher ces nœuds d'interaction de manière dynamique. Toutefois, cela doit avoir été programmé et prévu pour au moment de la compilation du programme.

7.3 Graphes d'interaction

Les graphes d'interaction reposent sur le modèle ICOM (Input Configuration Model) et la boîte à outils ICON dont nous avons rappelé les principes dans le chapitre précédent. Ils complètent le principe des configurations d'entrée en décomposant la classe des dispositifs d'application de manière à spécifier leur connexion avec un graphe de scène et de faciliter la mise en œuvre d'applications basées sur ce paradigme.

Dans le modèle des graphes combinés, nous définissons donc un graphe d'interaction comme une configuration d'entrée d'ICON qui décrit les comportements du graphe de scène et de ses objets, ainsi que leurs manipulations possibles par un utilisateur. Mais avant de décrire plus avant les principes et les mécanismes que nous avons mis en œuvre pour définir et réaliser cette interopérabilité, décrivons un scénario simple de configuration.

Les figures 7.6, 7.7 et 7.8 présentent la construction d'un graphe d'interaction, qui définit les actions de la souris sur d'autres dispositifs afin de manipuler les objets présents dans un graphe de scène :

1. Figure 7.6 - Le dispositif **souris** est connecté à un dispositif **curseur** par l'intermédiaire d'adaptateurs (dispositifs **sum** permettant de sommer les valeurs reçues, utilisés ici pour transformer les valeurs relatives de la souris en valeurs absolues). Ainsi, chaque déplacement de la souris déplace un pointeur à l'écran.

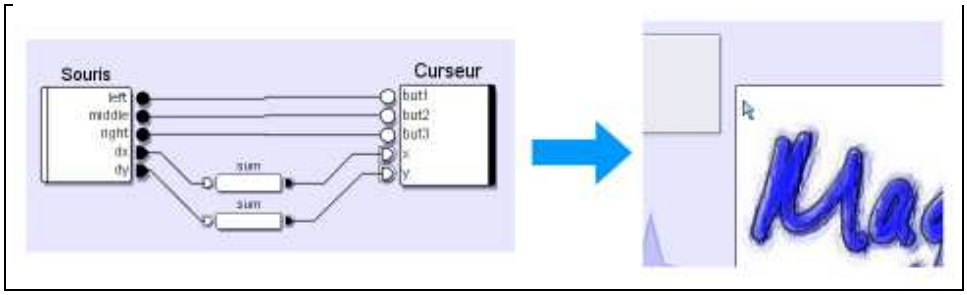


FIGURE 7.6 – Connexion de la souris à un curseur.

2. Figure 7.7 - Le dispositif **curseur** est connecté ensuite à un dispositif **pick** (sélection) qui permet de sélectionner les objets de l'interface (et donc du graphe de scène) qui se trouvent sous le curseur. Ainsi, chaque déplacement de la souris permet de sélectionner des composants de l'interface.

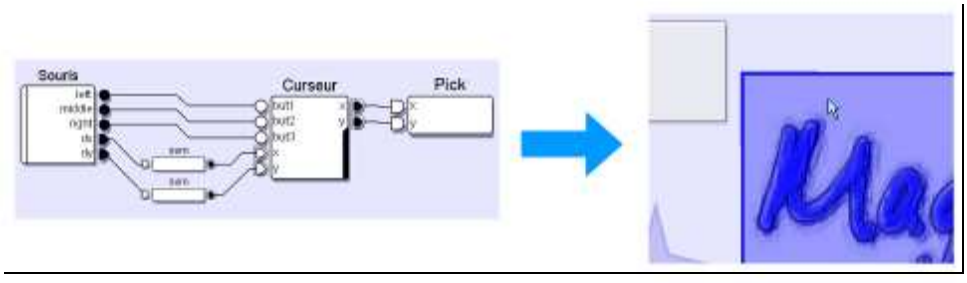


FIGURE 7.7 – Connexion d'un dispositif de sélection (*pick*).

3. Figure 7.8 - Enfin, le dispositif **pick** et un bouton de la souris sont connectés à un dispositif **drag** (Glisser-Déposer) qui permet alors de déplacer les objets sélectionnés, si ceux-ci le permettent.

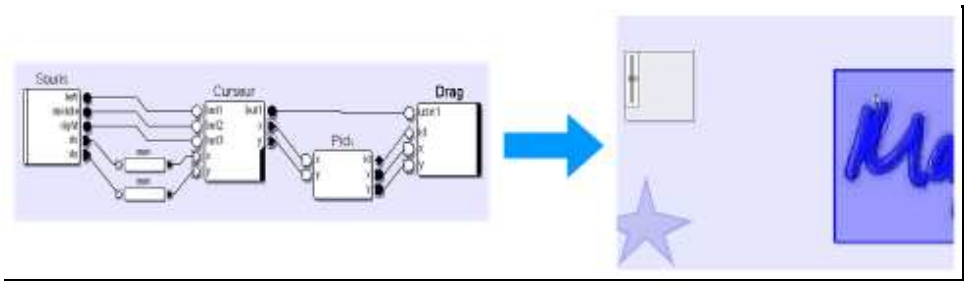


FIGURE 7.8 – Connexion d'un dispositif de manipulation directe (*drag*).

Des graphes plus complexes que cet exemple simple peuvent bien évidemment être construits. Le graphe d'interaction de la figure 7.9 a été conçu pour manipuler le graphe de scène de la figure 7.3 page 161. La souris contrôle un dispositif curseur et un dispositif de sélection. Le dispositif de sélection envoie les objets sélectionnés à deux dispositifs représentant des techniques d'interactions sur les composants graphiques (*MaggLiteHandle* et *MLComponent*, voir la section 7.4.1 page 166). Enfin, un dispositif représentant un périphérique d'entrée (un *Magellan*, périphérique isométrique à six degrés de liberté) est connecté directement à un dispositif de manipulation représentant un objet graphique du graphe de scène.

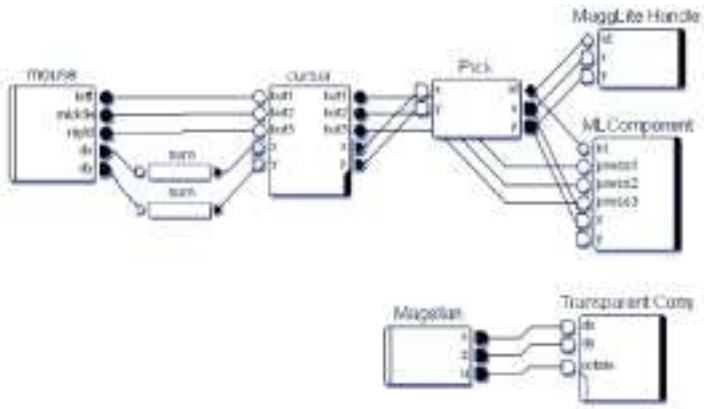


FIGURE 7.9 – Un graphe d'interaction possible pour le graphe de scène de la figure 7.3 page 161.

Ces exemples illustrent l'utilisation des dispositifs définis dans le modèle ICOM : les dispositifs système (souris, Magellan) et les dispositifs de la bibliothèque (curseur, adaptateurs). Mais comment définir les autres dispositifs utilisés dans cet exemple, ceux qui permettent de contrôler le graphe de scène et donc une partie de l'application ? (dispositif de sélection, de glisser-déposer ou de contrôle d'un nœud particulier du graphe de scène).
Le modèle ICOM définit pour cela la classe des dispositifs d'application. Ce sont les points d'entrée de l'application, lien entre la configuration d'entrée et les outils de l'application. Les dispositifs contrôlant le graphe de scène sont de cette classe, comme nous l'illustrons dans la figure 7.10 page suivante.

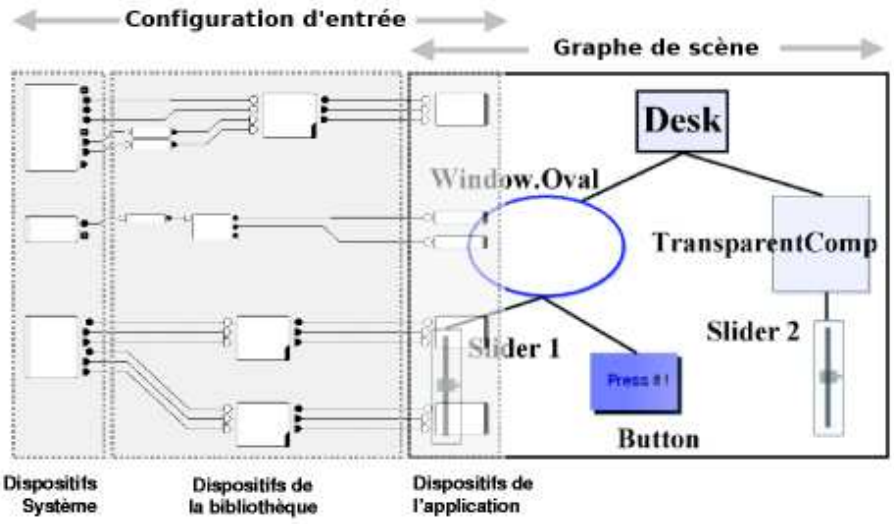


FIGURE 7.10 – Graphe de scène et configuration d'entrée. La liaison entre un graphe de scène et une configuration d'entrée s'opère au niveau des *dispositifs d'application*.

Mais contrairement à des dispositifs spécifiques à une application, ces dispositifs présentent la particularité d'être décrits au niveau du modèle des graphes combinés comme *le lien entre les graphes de scène et d'interaction*. Ils sont donc disponibles pour toutes les applications (au niveau de la boîte à outils). Nous les appellerons les **Points d'accès aux interactions**[(2)].

7.4 Points d'accès aux interactions (*IAPs*)

Les Points d'Accès aux Interactions (nous utiliserons l'acronyme *IAP* pour *« Interaction Access Points »* en anglais) sont les nœuds spécifiques du graphe d'interaction qui établissent des points de connexion dynamiques entre le graphe d'interaction et le graphe de scène. Dans la figure 7.1 page 158, qui illustre le modèle, ce sont les nœuds du graphe d'interaction d'où partent les flèches vers le graphe de scène. Les IAPs peuvent alors être considérés comme la « colle » qui assemble dynamiquement un graphe de scène avec un graphe d'interaction pour composer un graphe combiné. Ils transmettent et reçoivent des données du graphe de scène, modifient sa structure en insérant ou supprimant des nœuds, changent les propriétés ou déclenchent des comportements spécifiques aux nœuds.

Les dispositifs IAP fournissent plusieurs manières de lier des dispositifs ICON (les périphériques d'entrée, en général) aux composants graphiques du graphe de scène. Nous en distinguons dès lors quatre classes, selon le type de liaison qu'ils permettent :

- les **dispositifs d'interaction**, pour manipuler une classe d'objets graphiques ;
- les **manipulateurs**, pour contrôler une instance, un nœud du graphe de scène ;
- les **dispositifs de comportements**, pour spécifier le comportement d'un ou plusieurs objets graphiques ;
- les **outils internes**, pour réaliser une action dans un objet graphique.

[(2)]Nous généraliserons et intégrerons cette notion au modèle ICOM dans l'annexe C page 29.

7.4.1 Dispositifs d'interaction

Les **dispositifs d'interaction** définissent les fonctionnalités d'une technique d'interaction applicable à une **classe** de nœuds du graphe de scène, et donc à une classe particulière de composants de l'interface (voir figure 7.11).

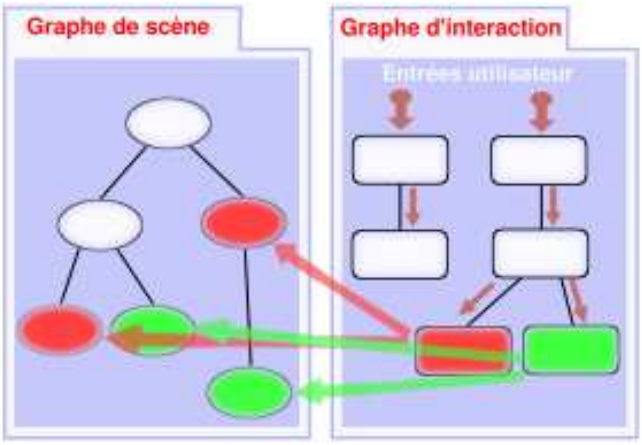


FIGURE 7.11 – Dispositifs d'interaction. Les dispositifs d'interaction communiquent avec une classe de nœuds du graphe de scène.

Selon [Foley *et al.*, 1990], une technique d'interaction décrit *« la manière de se servir d'un dispositif d'entrée »* pour accomplir une tâche sur un ordinateur. Conventionnellement, les techniques d'interactions sont décrites de façon *abstraite* et *générique*, sans les lier à un périphérique d'entrée et à une tâche (ou un objet) spécifiques (c'est le *style d'interaction* dans la figure 7.12). Pourtant, leur implémentation concrète s'avère être fortement dépendante du couple {dispositif, tâche}, de par la rigidité des outils actuels. Il est alors souvent fastidieux d'adapter une technique d'interaction à une autre classe d'objets et à d'autres périphériques d'entrée que ceux pour lesquels elle a déjà été implémentée.

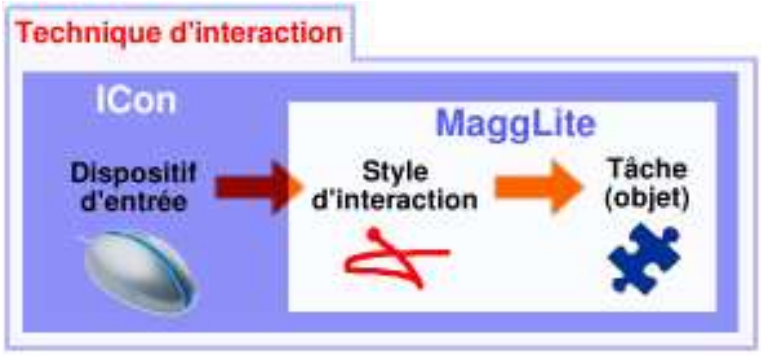


FIGURE 7.12 – Technique d'interaction. Une technique d'interaction est définie par un dispositif d'entrée, un style d'interaction, et une tâche à réaliser. Elle est souvent implémentée d'une façon monolithique qui la contraint à un dispositif et une tâche (contour global). ICON a permis l'adaptabilité en entrée à laquelle MAGGLITE ajoute la généricité en séparant interaction et objet d'intérêt.

ICON a permis de séparer la gestion des entrées de la réalisation des interactions, introduisant la notion d'adaptabilité en entrée, offrant plus de flexibilité. Notre modèle permet cette même adaptabilité et flexibilité au niveau de l'interaction et de son objet d'intérêt en externalisant le style d'interaction avec les dispositifs d'interaction (voir figure 7.12 page ci-contre).

Fonctionnement

Les classes de composants graphiques (nœuds du graphe de scène) déclarent leurs capacités et les dispositifs d'interaction sont spécialisés dans la gestion d'une ou plusieurs capacités particulières. Par exemple, une classe d'objets graphiques pouvant être déplacés signale cette capacité et toutes ses instance seront alors compatibles avec les dispositifs d'interaction permettant le déplacement d'objets. Ces dispositifs représentent donc des implémentations d'*Interactions Génériques*, n'étant pas liées aux objets eux-mêmes mais à leurs caractéristiques et possibilités.

Au niveau de l'implémentation de la boîte à outils, ces caractéristiques des classes de nœuds sont externalisées en implémentant des interfaces Java : l'interface *Moveable*, par exemple, rend compatible les futures instances avec les interactions de déplacement d'objets.

Cette modularité rend les interactions utilisables dans toutes les applications, et adaptables dynamiquement à tous types de périphériques pour composer de nouvelles techniques d'interaction. Elles peuvent aussi être étendues à d'autres classes d'objets graphiques (et donc de tâches), il suffit pour cela que ces classes implémentent les capacités compatibles.

Enfin, la dissociation des dispositifs d'entrée, objets d'intérêt et dispositifs d'interaction implique que ces derniers reçoivent les informations nécessaires à leur application par leurs canaux d'entrée. Pour cela, il présentent en général (voir figure 7.13 page suivante) :

1. un slot de type booléen *Use*, qui lorsqu'il recevra une valeur permettra de déclencher ou d'interrompre l'interaction.
2. un slot de type Objet *id*, qui recevra la liste des objets auxquels l'interaction doit s'appliquer à un instant donné. Ce slot est en général connecté en sortie d'un dispositif de sélection (*Pick*) présenté dans la figure 7.13(a) page suivante. Il permet la sélection dans le graphe de scène des objets qui se trouvent à une position donnée (nous détaillerons le fonctionnement de ce dispositif dans la section 8.3.2 page 190 du chapitre suivant).
3. des slots de données (entier, double, chaîne de caractères, etc.) qui recevrons toutes données utiles à l'interaction. Dans nos dispositifs exemples de la figure 7.13 page suivante, qui sont des techniques de manipulation directe d'objets graphiques, ce sont les slots *x* et *y* qui recevront des valeurs de position.

Utilisation

Les dispositifs d'interaction ont un vaste champ d'application. Ils permettent de factoriser et de réutiliser le *noyau* d'un style d'interaction. Leur réalisation se réduit à définir les propriétés des objets auxquels ils vont pouvoir s'appliquer, et à implémenter le *noyau* du dispositif, l'algorithme de l'interaction proprement dite. Une fois ajouté à la librairie, le dispositif est prêt à être utilisé.

MAGGLITE fournit essentiellement des dispositifs de manipulation directe tels que le dispositif *Drag* de la figure 7.13(b) page suivante (nous détaillons en annexe D, section D.2.1 page 52, les étapes

(a) *Pick*. (b) *Drag*. (c) *Responsive Handle*.

FIGURE 7.13 – Dispositifs d'interaction. La figure (a) présente un dispositif de sélection d'objets (*Pick*). La figure (c) présente un dispositif de Glisser-Déposer (*Drag*). La figure (c) présente un dispositif nommé *Responsive Handle*, une d'interaction originale que nous avons développée.

de la conception de ce dispositif). Mais cette architecture permet aussi la réalisation de techniques de visualisation d'informations, de méthodes de saisie de texte, etc.

Exemple

Le graphe d'interaction de la figure 7.8 page 164 permet de décrire la technique d'interaction de *Glisser-Déposer* à partir du dispositif *Drag*, et de l'utiliser sur tous les objets compatibles. Dans la configuration proposée, la manipulation sera réalisée en se plaçant sur l'objet, en pressant le bouton gauche de la souris et en la déplaçant pour déplacer l'objet. Le relâchement du bouton de la souris terminera cette interaction. Toutefois, cette technique n'est pas figée, et peut être initiée par tout autre dispositif permettant de la déclencher et de la réaliser (un bouton du clavier ou une commande vocale peuvent être connectés au slot *Use*, des dispositifs absolus, relatifs ou même une webcam et des dispositifs adaptateurs sur les slots *x* et *y*). Enfin, cette technique de Glisser-Déposer peut être remplacée par une autre en utilisant un autre dispositif (un exemple est donné dans la section 8.4.2 page 201 qui décrit la conception d'interfaces avec MAGGLITE).

Nous avions qualifié ces dispositifs d'implémentations d'***Interactions Génériques*** d'un point de vue du programmeur. Du point de vue de leurs utilisateurs (concepteurs d'interfaces ou utilisateurs) nous pouvons les qualifier d'***Interactions Enfichables***.

7.4.2 Manipulateurs

Contrairement aux dispositifs d'interaction, qui peuvent potentiellement agir sur n'importe quel objet du graphe de scène, les **manipulateurs** sont des dispositifs IAP liés à des instances, à des nœuds particuliers du graphes de scène (voir figure 7.14 page suivante). Ces *dispositifs d'instance* déclenchent des commandes que sait exécuter l'unique objet graphique qu'ils contrôlent : se déplacer, se redimensionner, etc.

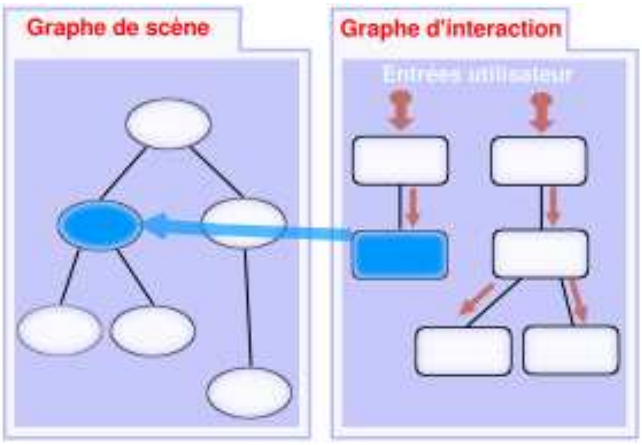


FIGURE 7.14 – Dispositifs manipulateurs. Un manipulateur communique avec un nœud spécifique et unique du graphe de scène.

Fonctionnement

Dès qu'un objet graphique est créé et ajouté au graphe de scène, il apparaît alors sous la forme d'un dispositif dans la bibliothèque du configurateur graphique d'ICON : son manipulateur, qui externalise les points d'entrée et de sortie pour interagir directement avec lui. Pour reprendre l'exemple d'un objet qui peut être déplacé, son manipulateur présentera alors des slots d'entrée *move*, prévus pour recevoir des coordonnées (absolues ou relatives). Les manipulateurs peuvent aussi présenter des slots de sortie, correspondants à des états que l'objet graphique peut transmettre.

Les manipulateurs sont générés dynamiquement et ajoutés à la bibliothèque des dispositifs lorsque des objets graphiques sont instanciés. Ainsi, à chaque capacité spéciale de l'objet correspond un slot d'entrée sur son manipulateur permettant de la déclencher et/ou un slot de sortie permettant d'y accéder (nous détaillerons les mécanismes de génération automatique de ces dispositifs dans l'annexe C consacrée aux extensions que nous avons apportées à ICON).

Exemple

La figure 7.15 page suivante présente la connexion d'un dispositif d'entrée isométrique à six degrés de libertés, le *Magellan*, au manipulateur d'un objet graphique, *Thumbnails*. Les axes x et z du dispositif physique sont connectés aux slots de déplacement relatif de l'objet (dx et dy) et son axe de rotation u au slot de rotation de l'objet (*rotate*). Les actions sur ces axes permettent donc d'agir directement sur l'objet graphique. Les signaux de chaque canaux transitent par les dispositifs de la configuration (le graphe d'interaction) et sont transmis à l'objet (le graphe de scène) par son manipulateur (flèches bleues de la figure 7.15 page suivante). Par contre, du point de vue de l'utilisateur, ce mécanisme est transparent et le périphérique physique est directement couplé, associé à l'objet graphique (flèche rouge pointillée sur la figure). Ce principe d'association directe est une manière simple et efficace de faire de la manipulation directe, en renforçant la notion de contrôle dans l'approche en flot de données.

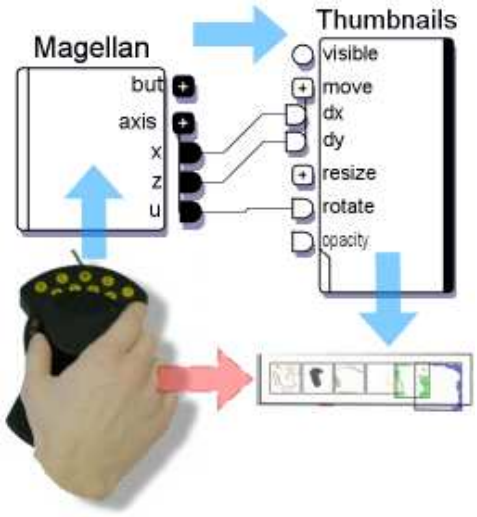


FIGURE 7.15 – Manipulateur. Le dispositif du périphérique d'entrée *Magellan* (à gauche) est connecté aux slots de déplacement du dispositif manipulateur de l'objet graphique *thumbnails* (à droite). Ainsi, il est possible de déplacer directement l'objet en agissant sur le dispositif d'entrée.

7.4.3 Dispositifs de comportements

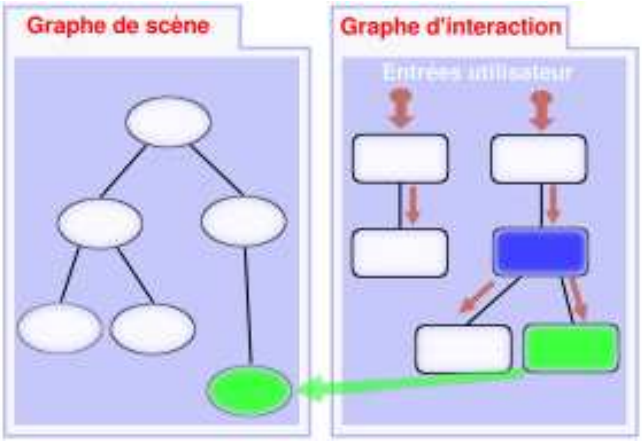


FIGURE 7.16 – Dispositifs de comportement. Un dispositif de comportement est connecté en aval d'un manipulateur afin de communiquer avec le nœud correspondant du graphe de scène.

Les **dispositifs de comportement** décrivent des comportements ou des états génériques (pulsation, surlignage, « collage », etc.) qui peuvent être appliqués à n'importe quel composant ou groupe de composants de la boîte à outils. Leur place et leur fonctionnement dans le modèle des graphes combinés sont très liés aux manipulateurs. En effet, les dispositifs de comportement doivent être connectés aux objets graphique par l'intermédiaire de leurs manipulateurs (voir figure 7.16). Ainsi, il est possible de spécifier lors de l'exécution de l'application quel va (vont) être le(s) comportement(s) graphique(s) d'un objet de l'interface.

Ces dispositifs présentent le double avantage d'externaliser les comportements graphiques des objets de la boîte à outil, afin de pouvoir les réutiliser simplement et dynamiquement sur tout type de composants, et de rendre leur déclenchement complètement configurable. Ainsi, en opposition aux modèles d'architectures logicielles standards qui proposent différentes classes d'objets pour différents comportements, notre proposition permet des associations dynamiques objet-comportement(s) à un niveau de granularité fin et sans avoir recours à de lourdes modifications logicielles (implémentation de nouvelles classes d'objets).

Fonctionnement

Les dispositifs de comportement ont deux slots d'entrée par défaut (voir figure 7.17(a) page suivante) :

1. un slot *magglite*, permettant de connecter un ou plusieurs manipulateurs ;
2. un slot booléen *enable*, qui active ou désactive le comportement.

À la réception d'une valeur *vrai* par son slot d'activation, le dispositif de comportement va alors appliquer le comportement pour lequel il a été programmé à l'objet graphique connecté à son slot *magglite*. Sinon, il se désactive jusqu'à une prochaine activation.
Il est évident que d'autres slots peuvent être présents dans le cas de comportements plus complexes que ceux que nous proposons à titre d'exemple.

Utilisation et exemples

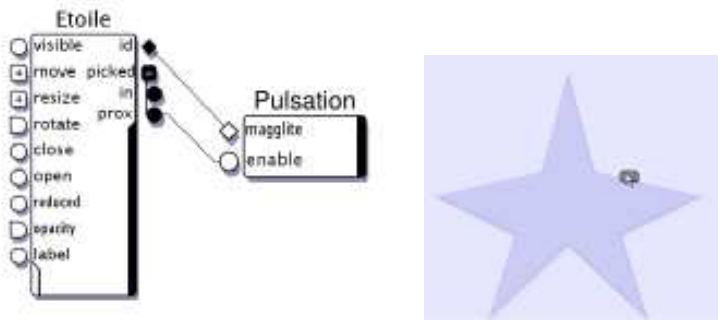
Nous définissons deux modes d'utilisation des dispositifs de comportement dans des graphes d'interaction, selon la méthode choisie pour leur déclenchement :

1. le mode **direct**, lorsque l'objet graphique associé ou une action sur cet objet graphique contrôle le déclenchement du comportement ;
2. le mode **indirect**, lorsque le déclenchement du comportement est contrôlé par un évènement non directement lié à l'objet associé.

Un objet graphique peut être connecté à plusieurs comportements avec différents modes d'activation et nous proposons aussi une notion de comportement **de groupe** pour connecter plusieurs objets à un même comportement.

Comportement direct
Nous parlons de **comportement direct** lorsque le déclenchement du comportement est contrôlé par le composant graphique qui lui est associé. La figure 7.17 page suivante présente une telle connexion. Le manipulateur de l'objet graphique *étoile* et le dispositif de comportement *pulsation* sont connectés par leurs slots *id* (sortie du manipulateur) et *magglite* (entrée du comportement). Ainsi, le comportement pulsation est associé de manière unique à l'objet graphique étoile. Le slot de déclenchement est connecté au slot de sortie du manipulateur *picked.prox*. Cette connexion permet de déclencher la pulsation de l'objet graphique dès que le pointeur est dans sa zone de proximité[(3)].

[(3)]Le slot *id* est présent sur tous les manipulateurs d'objets. Il envoie la référence de l'objet graphique auquel le manipulateur est associé. Les slots *picked.in, picked.prox* sont présents sur les manipulateurs des objets pouvant être sélectionnés.

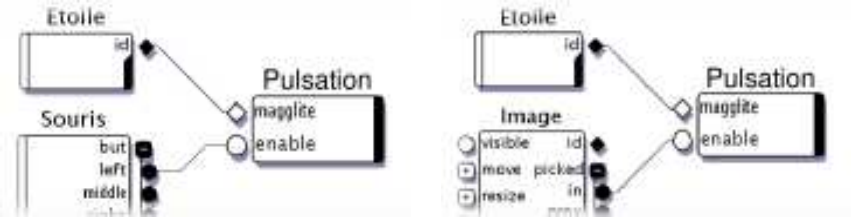


(a) (b)

FIGURE 7.17 – Comportement direct. Figure(a) : le manipulateur d'un composant graphique (*Etoile*) est connecté directement à un dispositif de comportement (*pulsation*). Figure (b) : le comportement est déclenché lorsque le pointeur est dans la proximité du composant associé.

Comportement indirect

La notion de **comportement indirect** est différente du point de vue de l'action qui déclenche le comportement. Celui-ci n'est plus directement connecté à l'objet en question, mais à un autre dispositif. Ainsi, comme le montre la figure 7.18, ce peut-être un périphérique d'entrée (figure 7.18(a)), un manipulateur d'un autre objet graphique (figure 7.18(b)) ou tout autre dispositif compatible.



(a) (b)

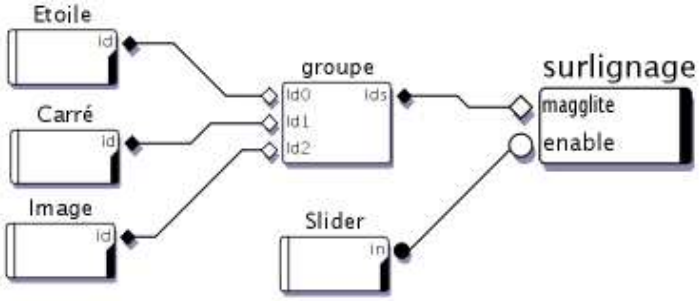
FIGURE 7.18 – Comportement indirect. Figure (a) : le comportement est déclenché par un périphérique d'entrée. Figure (b) : le comportement est déclenché par un autre composant.

Dans le premier cas, la pulsation de l'objet *étoile* est déclenchée lorsque le bouton gauche de la souris est pressé, dans l'autre, lorsque le pointeur est situé sur l'objet *image*.

Comportement de groupe

Nous avons vu jusqu'ici les dispositifs permettant l'association d'un ou plusieurs comportements à un objet unique. Nous proposons un dispositif **groupe** qui permet de connecter plusieurs objets graphiques pour les associer à un même comportement. Nous appelons ce mode de configuration le **comportement de groupe**. Ainsi, que ce soit en déclenchement direct ou indirect, tous les objets se verront attribuer le même comportement, déclenché par la même action. La figure 7.19 page ci-contre illustre ce principe par la connexion de trois objets graphiques à un dispositif de comportement *sur-*

lignage déclenché par la sélection d'un quatrième objet (comportement indirect). Lorsque le pointeur survole cet objet, le comportement est appliqué aux trois autres (voir figure 7.19(c)).



(a)

(b)

(c)

FIGURE 7.19 – Dispositifs de comportement. Comportement de groupe. Dans la figure (a), trois composants sont groupés et connectés à un comportement déclenché par un autre composant. Les autres figures présentent l'état des composants lorsque le comportement est inactif (b) ou actif (c).

Les exemples que nous avons présentés restent relativement simples et illustratifs. Toutefois, les dispositifs de comportement et les modes d'utilisation que nous leurs avons associés offrent de nombreuses possibilités de prototypage et de conception d'interactions avancées, particulièrement avec des objets graphiques plus évolués ou des configurations plus complexes. En plus des avantages de l'association dynamique objet(s)/comportement(s) en terme de réutilisabilité et de configurabilité, ces dispositifs permettent encore une fois de rendre les retours graphiques de l'interface modifiables et adaptables.

7.4.4 Outils internes

Les **Outils Internes** sont la quatrième classe de points d'accès aux interactions. Ces dispositifs représentent des outils conçus pour remplir une tâche dans un composant graphique spécifique. Leur action sur le graphe de scène peut donc être vue comme celle des manipulateurs, n'envoyant des messages qu'à un seul nœud du graphe de scène. Toutefois, leur sémantique n'est pas la même au niveau

de la conception des interactions.
Un manipulateur est une représentation d'un objet, et surtout des commandes d'interaction qu'il sait exécuter. Il agit donc *sur* l'objet graphique concerné (déplacement, rotation, redimensionnement, activation d'états ou déclenchement de comportements). Les outils internes sont des actions applicables *dans* l'objet graphique, conformément aux actions que permet l'objet concerné et aux activités qu'il permet de réaliser (le dessin, par exemple).

Fonctionnement

Les outils internes doivent être assignés à un nœud compatible du graphe de scène. Dès lors, ils sont visibles dans la librairie des dispositifs du configurateur graphique. Ils sont activés lors de leur connexion dans un graphe d'interaction et n'agissent qu'à l'intérieur des limites graphiques du composant auquel ils sont attaché. Ils procurent un retour visuel de type curseur qu'ils se chargent d'installer dans le graphe de scène (voir figure 7.20), en amont du composant graphique auquel il sont attachés. Cela permet l'affichage de ce retour de l'outil au dessus du composant. Ensuite, lorsque le signal d'utilisation est déclenché sur l'outil (slot *Use*), la méthode d'action est appelée pour accomplir la tâche ou l'interaction programmée.

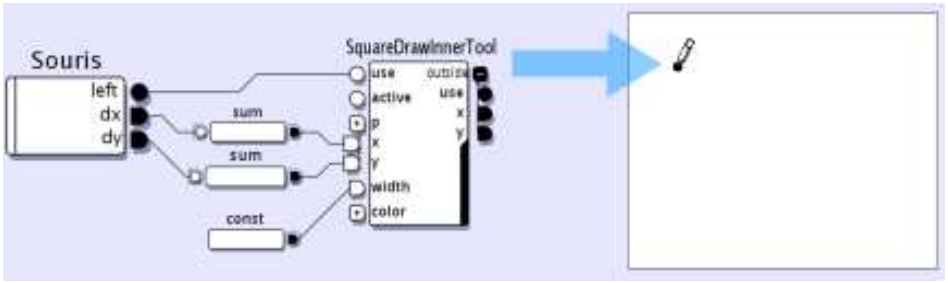


FIGURE 7.20 – Un outil interne est attaché au composant graphique carré. Connecté dans le graphe de configuration (à gauche), son curseur est visible dans le composant (à droite).

Utilisation et exemples

Les outils internes ont été conçus pour remplir des tâches *positionnelles*. Ils doivent donc recevoir en entrée des valeurs de position (en coordonnées écran). Dès lors, deux méthodes de traitements sont possibles, caractérisant deux modes : le **mode local** et le **mode global**. Nous prendrons pour exemple de description de ces modes un outil interne de dessin (*DrawInnerTool*), permettant de tracer des traits sur le composant auquel il est ajouté.

Mode local
Les valeurs reçues par les slots d'entrées de coordonnées sont converties dans le système de coordonnées local de l'objet graphique auquel est attaché l'outil. Ce mode permet une projection complète de la portée du périphérique d'entrée connecté à l'outil sur la surface visible du composant. Dans la figure 7.21 page suivante, l'outil interne de dessin est connecté à une tablette graphique en mode *mode local*, mode approprié aux périphériques d'entrée absolus.

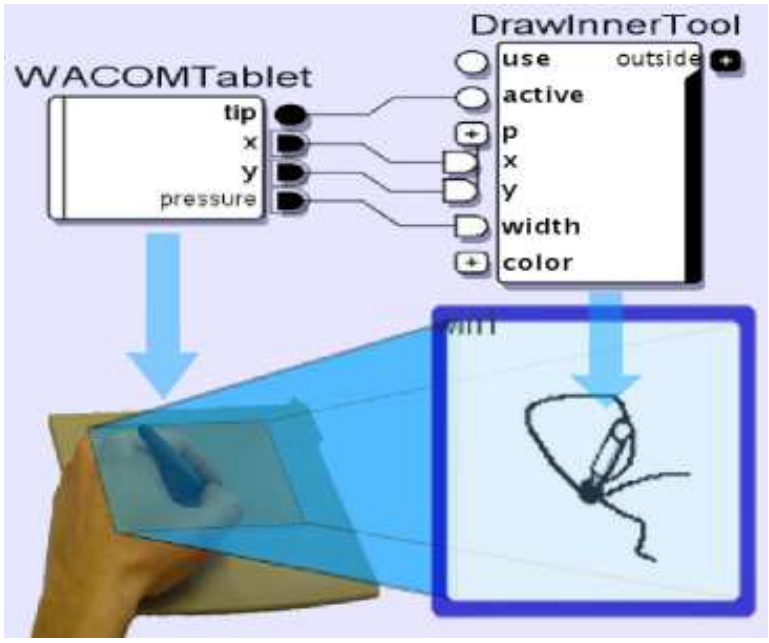


FIGURE 7.21 – Outil Interne en *mode local*.

Mode global
Dans ce mode, au contraire du *mode local*, l'outil n'opère aucune conversion sur les coordonnées et les interprète donc en coordonnées écran. Ainsi, la portée du dispositif d'entrée est projetée sur l'écran. L'outil n'est alors actif que lorsque les coordonnées reçues sont dans les limites de l'objet graphique auquel il est attaché. Dans le cas contraire, l'outil est inactif et les coordonnées sont transmises par ses slots de sortie, sans que son action soit déclenchée. Dans la figure 7.22 page suivante, deux outils internes sont connectés l'un à l'autre en mode global.

Le premier dans la chaîne, nommé *GestureZone*, est attaché au composant à bord bleu et son action est de reconnaître et interpréter des gestes. Ses slots d'entrée sont connectés à la sortie d'un dispositif tablette graphique. Le second outil est un outil interne de dessin, attaché au bureau de l'application. Ses slots d'entrée sont connectés aux slots de relais de l'outil de geste (*outside.use*, *outside.x* et *outside.y*).

Lorsque les coordonnées écran envoyées par le dispositif tablette sont dans les frontières du composant de reconnaissance de geste, l'outil de geste attaché est activé et ne transmet aucune valeur à l'outil de dessin du bureau. Lorsque les coordonnées sont en dehors des frontière de la zone de gestes, l'outil de gestes est désactivé et il transmet les coordonnées à l'outil de dessin du bureau. L'utilisateur peut donc dessiner n'importe où sur le bureau, excepté dans la zone de gestes où ses traits sont interprétés. Cette projection de zones de l'écran sur les dispositifs d'entrée est une manière efficace de construire des interactions post-WIMP, tout spécialement avec des dispositifs tels que des tablettes, tablettes écran ou écran tactiles.

La boîte à outils propose trois exemples d'outils internes : l'outil interne de dessin, l'outil interne de gomme et un outil interne générique que nous détaillerons dans le chapitre suivant. Les différents mécanismes (modes de gestion des coordonnées, retour graphique et installation de l'outil) sont implémentés dans la boîte à outil et sont donc hérités lors de la création d'un nouveau type d'outil

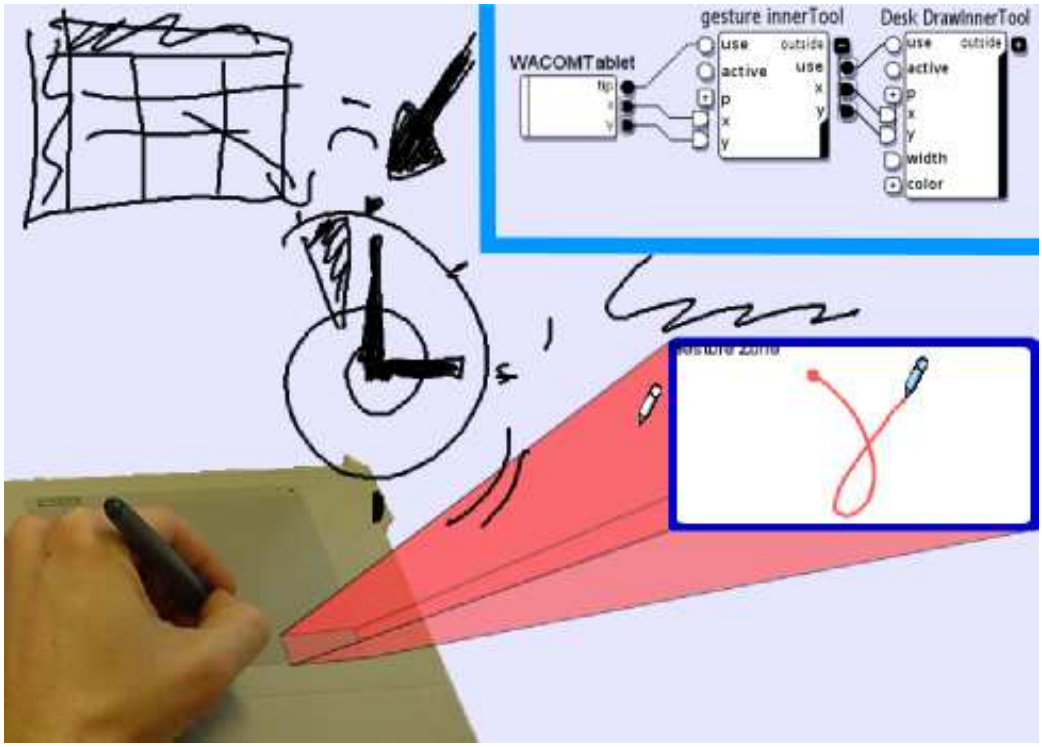


FIGURE 7.22 – Outil Interne en *mode global*.

interne. Seule *l'action* proprement dite de l'outil doit être implémentée (l'annexe D page 45 offre plus de détails techniques sur la création d'un outil interne).

7.4.5 Dispositif de l'application

Une dernière notion peut être assimilée à un point d'accès à l'interaction, bien que le dispositif en résultant ne réalise pas directement une connexion entre le graphe d'interaction et le graphe de scène : le **dispositif de l'application**.

Les dispositifs présentés précédemment permettent essentiellement de contrôler les outils, de spécifier des interactions et comportements pour l'interface au niveau des activités que permet l'application (dessin dans des composants, déplacement d'objets, affichage de données, etc.). Le dispositif de l'application ne permet pas le contrôle d'objets graphiques mais des fonctionnalités globales, c'est à dire *au dessus* du graphe de scène. Ainsi, il peut proposer des fonctionnalités telles que le déplacement ou le redimensionnement de l'environnement graphique global de l'application (fenêtre système), l'ouverture du configurateur graphique ou la fermeture de l'application. La figure 7.23 page ci-contre présente un dispositif d'application.

Les fonctionnalités de base qu'il propose sous forme de slots d'entrée sont communes à toutes les applications construites avec la boîte à outils : plein écran, configuration, quitter, etc. La création de ce dispositif est dynamique, lors du lancement de l'application et il est bien sur possible de lui rajouter d'autres fonctionnalités (par programmation).

FIGURE 7.23 – Dispositif de l'application. Ce dispositif, créé au démarrage de toute application MAGGLITE, permet le contrôle des fonctionnalités globales de l'application.

Ce dispositif fournit donc, au niveau de l'application, les mêmes capacités, principes de conception et d'utilisation que ceux proposés au niveau des outils de l'application. Cela permet de proposer des interactions homogènes et d'éviter pour l'utilisateur une rupture entre les outils proposés et le contrôle global de l'application. En effet, il serait incohérent de proposer une barre de menus standard d'application WIMP (*fichier*, *éditer*, etc.) dans une application avancée basée sur le dessin et la reconnaissance de gestes, comme SVALABARD, par exemple. Le dispositif de l'application permet ce contrôle avec les mêmes interactions, et autorise de fait la suppression de tous les contrôles et widgets standards si ceux-ci ne sont pas adaptés.

7.5 Conclusion

Nous avons décrit dans ce chapitre un modèle d'architecture logicielle pour la réalisation d'applications interactives : les *graphes combinés*. Ce modèle réunis les *graphes de scènes* pour la description de la partie graphique, et les *graphes d'interaction* pour la description des comportements, dans un même paradigme flexible et clair. Cette association conjugue donc capacités d'expression et performances graphiques avec une description fine et dynamique de l'interaction.

Les graphes combinés reposent essentiellement sur la mise en relation de ces deux architectures par l'intermédiaire de modules permettant la combinaison dynamique des deux graphes : les *Points d'Accès à l'Interaction* (ou *IAPs*).

Les IAPs : le cœur des graphes combinés

Les *IAPs* permettent donc d'accéder au graphe de scène depuis le graphe d'interaction afin d'en définir les comportements. Dans cette optique, les quatre classes de *IAPs* que nous avons proposées permettent :

1. de définir des actions génériques sur une classe de composants graphiques (les *dispositifs d'interaction*) ;
2. de définir des actions particulières à un composant graphique (les *manipulateurs*) ;
3. de définir des comportements graphiques (les *dispositifs de comportement*) ;

4. de définir des outils propres à un composant graphique (les *outils internes*).

Ces quatre abstractions permettent de concrétiser un grand nombre de techniques d'interaction, comme nous le verrons dans le chapitre suivant. Bien qu'ils décrivent différents modes d'accès possibles à un graphe de scène (par classes ou instances de nœud), il serait toutefois ambitieux de notre part d'affirmer que les IAPs couvrent tous les besoins en terme de description des interactions. Ils offrent avant tout un cadre extensible pour l'exploration, permettant la réunion de nombreux styles d'interaction dans un même modèle.

Dans le chapitre suivant, nous décrivons la boîte à outils MAGGLITE, développée selon ce modèle des graphes combinés. Nous proposerons des exemples de son utilisation pour la réalisation d'application interactives, mais aussi pour le prototypage de nouvelles techniques d'interaction.

Chapitre 8

La boîte à outils MAGGLITE

« Nous estimons les ecchymoses des Esquimaux aux mots exquis »

M. DUCHAMP.

Sommaire

8.1 Introduction

Dans ce chapitre, nous décrivons les détails de l'implémentation de l'architecture des graphes combinés dans la boîte à outils MAGGLITE, ainsi que les composants et mécanismes qu'elle mets à disposition pour la conception d'interfaces avancées.

Les objectifs de MAGGLITE sont à la fois de proposer une réalisation concrète du modèle des graphes combinés pour permettre plus de flexibilité dans la conception des applications interactives, mais aussi d'intégrer dans une même boîte à outils des techniques jusqu'alors disséminées dans différentes implémentations plus ou moins complexes à réunir.

MAGGLITE est implémentée en *Java*, langage utilisé auparavant pour le développement d'ICON.

Après avoir décrit brièvement comment nous avons utilisé et étendu ICON pour son intégration dans notre architecture, nous ferons le tour des composants de base que nous avons développés dans MAGGLITE, qu'ils soient des objets graphiques ou des techniques d'interaction. Ces exemples couvrent, selon nous, un bon nombre de problèmes soulevés par la conception d'interfaces et d'interactions avancées. Mais il permettent aussi de comprendre d'un point de vue plus pratique la « philosophie » de notre boîte à outils. Enfin, nous proposons une dernière section traitant de son utilisation. Nous évoquerons les possibilités d'extension de la boîte à outils puis nous présenterons ensuite un outil réalisé avec MAGGLITE, permettant de prototyper interactivement des applications sans avoir recours à la programmation. Enfin, nous évoquerons le développement d'applications plus complexes, ayant parfois même recours à l'extension de la librairie. Nous détaillerons en particulier la réalisation de notre application de dessin en perspective pour la modélisation 3D : SVALABARD.

8.2 Extension et utilisation d'ICON

MAGGLITE se positionne comme une sur-couche à ICON (voir figure 8.1 page suivante), lui ajoutant une architecture graphique avancée (les graphes de scène) et surtout des dispositifs spécifiques pour interagir avec cette architecture graphique. Mais plus que cela, MAGGLITE a aussi permis de faire évoluer ICON vers un système de description complet des interactions, au delà du système de gestion d'entrées multiples qu'il était. Pierre DRAGICEVIC a étudié certaines possibilités d'utilisation d'ICON pour spécifier des interactions avancées dans [Dragicevic et Fekete, 2004] et montré des exemples des possibilités de son système dans cette optique [Dragicevic, 2004b]. Nos propositions, le modèle des graphes combinés et sa réalisation dans MAGGLITE, ont permis de conforter ces pistes, en particulier pour les interfaces post-WIMP.

8.2.1 Extension d'ICON

Afin de permettre la composition de nombreux graphes d'interaction, la bibliothèque des dispositifs de MAGGLITE contient tous ceux disponibles dans ICON (dispositifs système pour les périphériques d'entrés et dispositifs de la bibliothèque pour adapter les entrées) ainsi que des dispositifs IAP, les dispositifs de plus haut niveau de MAGGLITE.

Dans un premier temps, nous avons étendu le modèle ICOM sous-jacent à ICON. Nous proposons une formalisation de la notion des *Points d'accès à l'interaction* dans ce modèle afin d'assurer la reproductibilité de notre approche.

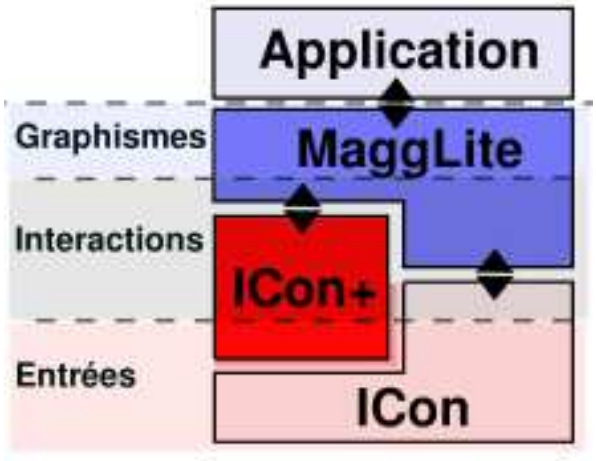


FIGURE 8.1 – Architecture concrète MAGGLITE/ICON. Entre MAGGLITE et ICON, nous avons introduit une extension nommée ICON+.

Dans un second temps, nous avons introduit de nouveaux dispositifs dans la librairie afin de la rendre encore plus polyvalente pour la description d'interactions de haut-niveau (unification du support des périphériques d'entrée, dispositifs *multimédia*, reconnaissance de gestes, etc.).
Enfin, nous avons introduit le principe et les mécanismes de la génération de dispositifs à l'exécution afin de permettre la création dynamique des *manipulateurs* de MAGGLITE.

Ces ajouts, l'extension ICON+ (voir figure 8.1), sont détaillés dans l'annexe C.

8.2.2 ICON à travers MAGGLITE

Bien que nous ne considérions pas la programmation avec la boîte à outil ICON comme particulièrement difficile, celle-ci demande tout de même une certaine expertise. Il était donc nécessaire, afin de ne pas avoir à assimiler les techniques de programmation de deux boîtes à outils, d'abstraire l'utilisation d'ICON dans MAGGLITE au plus haut niveau possible. Ainsi, il est tout à fait possible de développer des applications, de nouveaux composants graphiques et même de nouveaux dispositifs sans pour autant avoir une connaissance pointue d'ICON. Il suffit d'avoir une vision globale de ses principes de fonctionnement.

Encapsulation d'ICON

Nous avons tout d'abord « encapsulé » les environnements d'exécution d'ICON et de MAGGLITE, ainsi que leur initialisation dans une abstraction d'application de haut niveau. Ainsi, créer une application revient seulement à implémenter deux méthodes [(1)] : créer les objets graphiques et instancier des dispositifs si besoin.

Nous avons suivi le même principe pour la programmation de nouveaux points d'accès à l'inter-

[(1)]Nous proposons aussi une méthode interactive de création d'applications ne nécessitant pas de programmation en Java dans la section 8.4.2 page 199

action. Excepté pour les manipulateurs, qui sont générés automatiquement, nous avons inclus dans MAGGLITE trois abstractions permettant de développer rapidement de nouveaux dispositifs : pour les dispositifs d'interaction, les comportements et les outils internes. Il suffit alors de réaliser quelques méthodes de haut niveau (comme par exemple la méthode `toolAction(double x, double y)` , pour les outils internes qui représente l'action à effectuer par l'outil si il est actif). Nous ne détaillerons pas plus avant ces abstractions dans ce chapitre, des exemples complets étant fournis dans l'annexe D page 45.

Le configurateur d'entrées dans MAGGLITE

Enfin nous proposons une organisation de la librairie des dispositifs dans le configurateur graphique identique pour chaque application. *Input Configurator*, comme nous l'avons déjà rappelé avec la figure 6.3 page 155, propose une organisation arborescente des prototypes de dispositifs disponibles. Nous n'avons pas modifié cette organisation, mais simplement uniformisé la partie de l'arborescence qui concerne les applications MAGGLITE, comme le montre la figure 8.2.

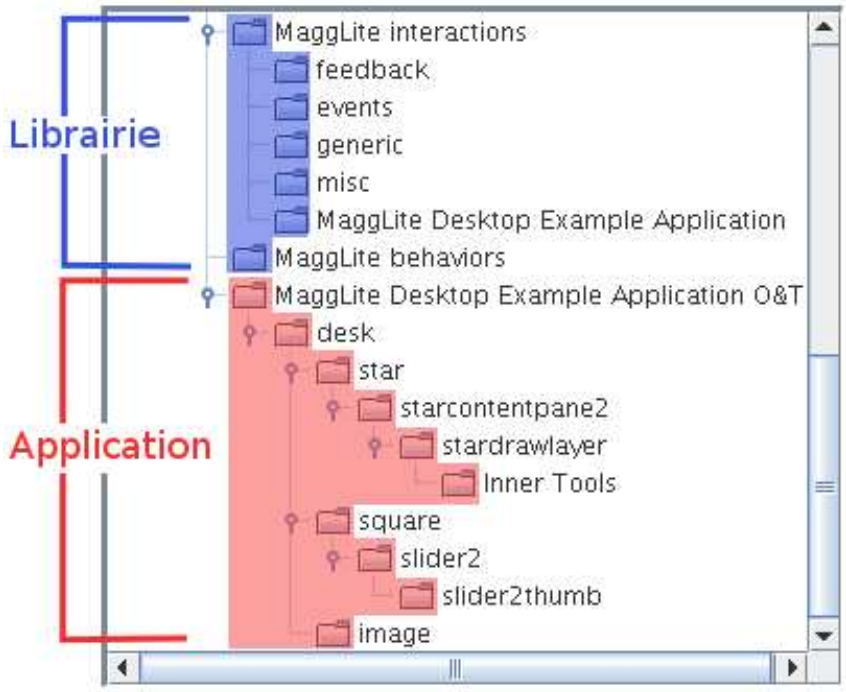


FIGURE 8.2 – Organisation des dispositifs de la librairie. Le configurateur d'entrée organise les dispositifs dans des dossiers (voir la figure 6.3 page 155, dans la section 6.4). Nous les avons étendus de la manière suivante dans MAGGLITE : en bleu, les dispositifs génériques de la librairie (dispositifs d'interaction, comportements) ; en rouge, les dossiers donnent une vue hiérarchique du graphe de scène de l'application, avec un dossier par objet graphique (manipulateurs et outils internes).

Les dispositifs spécifiques de notre boîte à outils se trouvent dans la partie application de l'arborescence. Ce dossier, qui porte le nom donné à l'application, est alors composé de deux parties principales :

1. Les dispositifs génériques de la boîte à outils (en bleu sur la figure 8.2), regroupés dans les sous dossiers *MaggLite interactions* et *MaggLite behaviors*.
2. Les dispositifs liés aux objets de l'application (en rouge sur la figure). Ceux-ci sont regroupé selon l'arborescence du graphe de scène, chaque dossier prenant le nom du composant qu'il représente. Cela donne un aperçu de la structure du graphe de scène, et permet de retrouver rapidement les dispositifs. Chaque dossier correspondant à un composant graphique contient son manipulateur (si celui-ci est rendu visible) et un éventuel sous-dossier *InnerTools* si l'objet s'est vu affecter un ou plusieurs outils internes.

8.3 MAGGLITE

Dans cette section, nous présentons les composants proposés dans la boîte à outils, ainsi que quelques exemples de leur utilisation. Nous commencerons par les *composants graphiques*, nœuds de graphe de scène, avant de décrire les *interactions* disponibles, regroupant différentes réalisation des points d'accès à l'interaction décrits dans le modèle des graphes combinés.

8.3.1 Composants graphiques

Les composants graphiques sont les nœuds qui vont permettre de composer des graphes de scène. Nous en distinguons deux classes, les objets atomiques et les objets composites, comme nous l'avons défini dans le modèle des graphes combinés. Nous décrirons à part des composants particuliers, les *calques*.

Les objets atomiques

Magglite et Magglite Container
Magglite et **Magglite Container** sont les deux classes abstraites de base de la librairie, permettant la définition des composants graphiques de plus haut niveau (avec une forme, une proximité, des attributs graphiques plus évolués, etc.).
Elles définissent les mécanismes et les opérations de l'architecture de graphes de scène, et tout spécialement les *traverses*. Les traverses (ou « traversal », en anglais), sont les opérations de parcours des nœuds d'un graphe de scène, pour effectuer les opérations d'affichage par exemple. Ces deux classes définissent aussi les *contextes* et attributs graphiques des objets, données qui vont être manipulées et propagées par les opérations de traverse[(2)], ainsi que les opérations abstraites qui seront appelées par ces traverses. L'opération d'affichage, par exemple, exécute une séquence précise d'opérations concrètes (affichage des ombres, affichages des fils et propagation du contexte) et d'opérations abstraites qui devront être implémentées par les classes de plus haut niveau (affichage avant les fils, affichage après les fils, etc.).

[(2)]Nous ne détaillerons pas plus ces notions de traverse et de propagation de contexte très commune dans de telles architecture. Nous attirerons juste l'attention sur le fait que notre implémentation d'une architecture de graphe de scène reste un prototype, et ne fournit pas encore d'opérations avancées et optimisées, comme peuvent le proposer d'autres boîtes à outils plus abouties.

L'objet *MaggLite Container*, descendant de l'objet *Magglite*, permet en plus l'agrégation d'autres objets : il représente donc les nœuds du graphe de scène, les objets *Magglite* pouvant être vus comme des feuilles.

Bureau
Le **Bureau** est le conteneur de base de toute application MAGGLITE, le nœud racine de tout graphe de scène d'application. Il étend le *MaggLite Container* et fournit tous les mécanismes qui initient les parcours du graphe de scène, en particulier pour l'affichage. C'est un composant que l'on peut qualifier d'hybride, car il intègre des fonctionnalités de la librairie Swing, utilisée pour obtenir les ressources nécessaires à l'affichage (fenêtre du système d'exploitation).
Outre les parcours pour affichage, ce composant initie aussi les parcours lors de l'initialisation de l'application pour l'accès aux dispositifs particuliers de chaque nœud (manipulateurs et outils internes), mais aussi pendant l'exécution, pour la sélection de nœuds (*pick*).

Composant
L'objet **Composant** concrétise la notion d'objet graphique en définissant une forme pouvant donc être affichée. Notons toutefois que la géométrie de ces objets reste abstraite dans son implémentation, celle-ci étant fournie lors de l'instanciation. Cette abstraction de la forme permet la création de composants graphiques de géométrie quelconque, ce qui est indispensable dans le contexte de développement des interfaces avancées (qui ne repose pas uniquement sur des objets rectangulaires ou elliptiques).
Définissant une forme, ces objets définissent aussi une *proximité*. La proximité d'un objet graphique est une zone de forme identique et de taille paramétrable autour de lui. Invisible par défaut, cette zone peut être rendue visible sous forme d'un halo, comme dans la figure 8.3. Nous reviendrons sur ce principe des proximités dans la section 8.3.2 page 190, celles-ci étant utilisées dans MAGGLITE pour les *interactions de proximité*.

FIGURE 8.3 – Proximité. La proximité forme un halo autour des objets graphiques (il peut être rendu invisible).

Enfin, cette classe est étendue par trois autres, offrant des capacités supplémentaires (voir figure 8.4 page suivante) :

1. un **Composant Transparent**, qui a la propriété de pouvoir varier d'opacité.
2. un **Composant Image**, qui étend le *Composant Transparent* et permet d'afficher une image.
3. un **Composant Pliable**, qui étend le *Composant Transparent* et offre en plus la capacité d'être

plié par l'intermédiaire de son manipulateur ou par une interaction générique (voir la section 8.3.2 page 192).

FIGURE 8.4 – Composant. Trois composants, un standard (en étoile), un composant image et un composant transparent et pliable.

Label
Enfin, MAGGLITE contient aussi un objet atomique permettant l'affichage simple de texte sur une seule ligne.

Les calques

Les **Calques** étendent la classe des objets pliables et ont la propriété de prendre la forme et la taille du container auquel ils sont ajoutés. Un objet calque par défaut n'a pas de contenu graphique, il n'affiche donc rien sur son composant parent. Ils proposent toutefois les fonctionnalités par défaut des objets dont ils héritent, comme la possibilité d'être rendus transparents, semi-transparents ou complètement opaques, ainsi que la capacité à être pliés. Ce sont de plus des conteneurs et peuvent de fait contenir d'autres objets graphiques.

Les calques représentent donc une abstraction pouvant être simplement étendue pour spécifier des propriétés et capacités particulières qui vont être ajoutées sous forme de couches graphiques à un autre objet. Les différents calques que nous proposons, déclinant le calque de base, permettent aussi l'ajout de couches d'interaction lorsqu'ils sont associés à des outils internes adaptés. C'est une réalisation efficace et modulaire (bien que moins complète) du modèle multi-couches proposé dans [Fekete et Beaudouin-Lafon, 1996; Fekete, 1996].

Calque de dessin
Les **Calques de dessin** permettent de dessiner sur un composant graphique (voir figure 8.5 page suivante). Il faut pour cela que le calque ait bien évidemment été ajouté à un composant et qu'un outil de dessin approprié lui soit affecté (outil interne de dessin).

Calque 3D
Les **Calques 3D** permettent l'affichage d'une scène 3D, en utilisant les capacité d'accélération matérielle des cartes graphique OpenGL. Nous avons pour cela utilisé la librairie graphique Jogl [Petersen

(a) (b)

FIGURE 8.5 – Calque de dessin. Un calque de dessin et son outil interne de dessin ont été ajoutés dans le composant graphique *Image*. Il est ainsi possible de dessiner sur le composant graphique (a). Les calques sont aussi des composants pouvant être pliés (b).

et Russel, 2004] pour la partie rendu 3D et la librairie Xith3D [Xith3D Project Group, 2003] afin de permettre la construction des scène 3D avec des graphes de scène. Ainsi, le calque 3D est un conteneur de notre graphe de scène qui affiche lui-même un graphe de scène 3D[(3)]. Il est ainsi possible de développer des applications de visualisation ou des environnements 3D très avancés et somme toute assez performants. Nous avons pour cela utilisé les méthodes fournies par la librairie Xith3D, mais aussi ajouté certaines méthodes de rendu 3D plus évoluées (notamment le rendu *non photoréaliste* de la figure 8.6 page ci-contre, pour lequel nous avons implémenté la méthode proposée dans [Gooch *et al.*, 1998]).

Ces calques sont une approche flexible d'inclusion de scènes 3D dans une boîte à outils 2D avancée. Ils permetent la conception d'interfaces très innovantes, mêlant 2D et « vraie » 3D[(4)] dans un modèle d'architecture logicielle unifié. Il reste toutefois à clarifier les possibilités d'interaction avec cette scène 3D. Actuellement, seule une interaction directe sur le point de vue de la scène 3D est possible, par l'intermédiaire du manipulateur du calque 3D (voir figure 8.7(c) page suivante). Cela permet tout de même la connexion directe de périphériques d'entrée standards ou avancés à cette manipulation de la vue, mais aussi de techniques d'interaction originales (manipulation de la vue 3D par reconnaissance de gestes ou par commandes vocales). Aller plus loin dans les interactions avec la scène 3D (sélection et déplacement d'objets, par exemple) implique à notre avis l'unification des graphes de scènes 2D et 3D dans l'architecture des graphes combinés. Bien qu'il soit aussi possible

[(3)]Il est vrai que cette approche brise quelque peut la cohérence de notre architecture, c'est pourquoi nous envisageons d'unifier ces deux modèles distincts de graphe de scène en un seul pour la gestion de l'affichage 2D et 3D avec OpenGL. Notons toutefois que cette séparation des modèles ne pose pas de réel problème pour le dévelloppeur étant donné que nous fournissons une abstraction de haut niveau rendant transparente les mécanismes d'affichage bas niveau (utilisation des *pbuffer* pour afficher la scène 3D dans un contexte graphique hors-écran, ou « lightweight »). L'unification apporterait certes une amélioration des performances.

[(4)]Nous entendons par « vraie 3D » de la visualisation 3D en temps réel, excluant donc les méthodes de 2D1/2 ou de 3D précalculée.

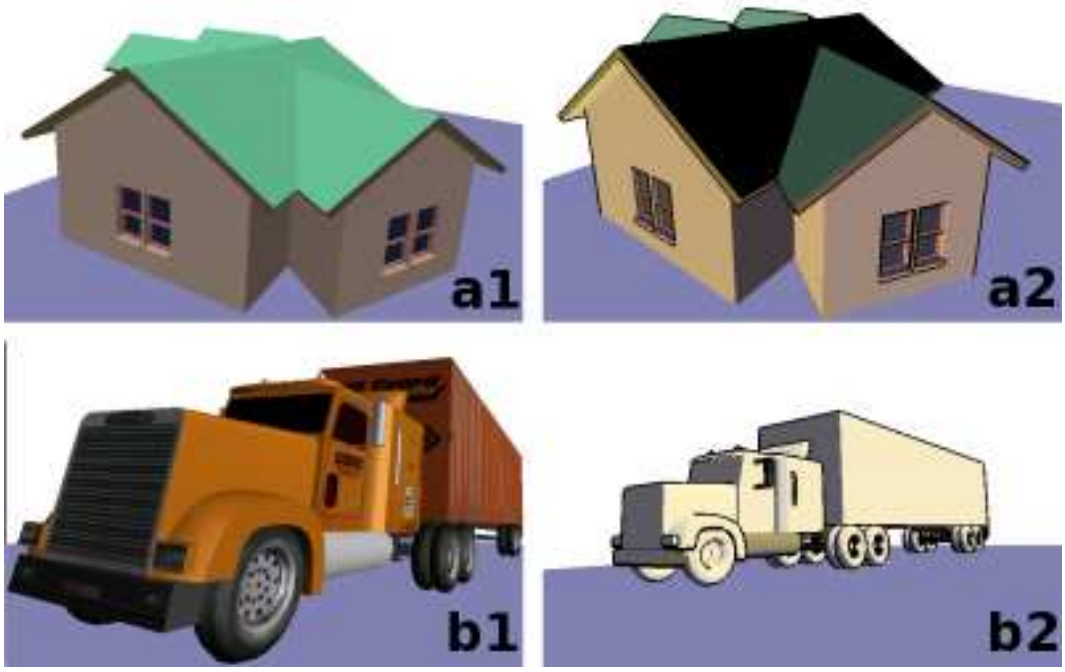
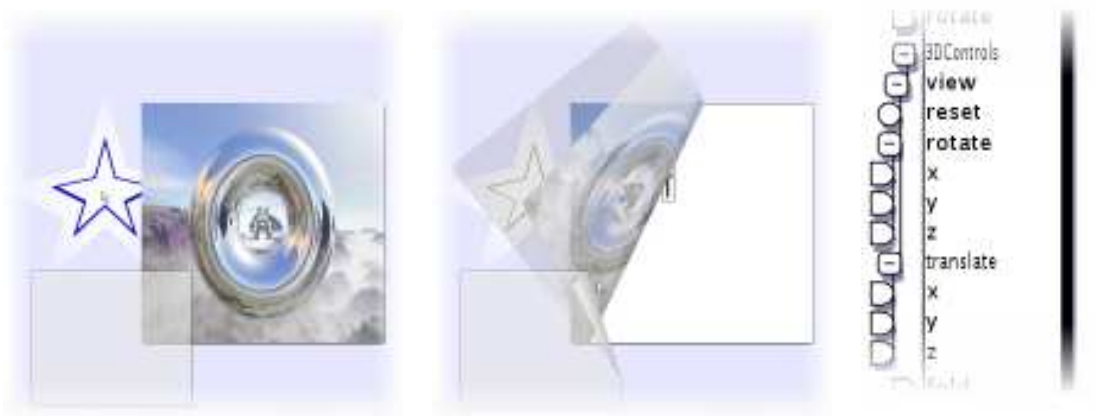


FIGURE 8.6 – Rendus 3D. Le calque 3D permet des rendus avancés : le rendu « classique » de polygones (a1), les rendus texturés (b1), mais aussi un rendu non photoréaliste (a2 et b2).



(a) (b) (c)

FIGURE 8.7 – Calque 3D. Le calque 3D de la figure (a) permet d'afficher une scène 3D dans un composant graphique. Il peut être plié, comme tout autre calque (b). Son manipulateur (c) externalise les contrôles du point de vue.

d'externaliser ces points d'entrée à l'interaction 3D par l'intermédiaire du manipulateur associé au calque 3D, cela serait beaucoup moins flexible qu'un modèle de graphe combinés étendu à la 3D (un seul graphe de scène 2D/3D associé à un graphe d'interaction).

Gestionnaire de calques
Un troisième composant est lié aux calques : un **Gestionnaire de calques**, permettant de contenir et manipuler une *pile* de calques. Les composants graphiques de la boîte à outils que nous avons vus jusqu'alors peuvent contenir un nombre indéterminé de calques, mais ne fournissent aucun moyen de les manipuler, les réordonner, établir une stratégie de sélection des calques. Ce composant de gestion de calques le permet en proposant les méthodes nécessaires à l'ajout dynamique de calques de différents types (dessin, 3D ou tout autre type de calque qui sera ajouté à la boîte à outils), mais aussi les mécanismes permettant de sélectionner un calque actif du point de vue de l'interaction, de les réordonner dans la pile, d'en supprimer, etc. De plus, il propose une vue miniature de la pile de calques qu'il contient, ainsi que les notions de *calque d'arrière plan* et *calque de premier plan*, deux calques qui ne peuvent être réordonnés (permettant ainsi l'ajout d'objets graphiques en fond ou premier plan). Son manipulateur donne accès à ces fonctions de gestion de la pile de calques.

Ce composant, associé à sa vue miniature, a principalement été conçu pour la réalisation de SVALABARD. Nous l'avons toutefois aussi utilisé pour une application exemple de la boîte à outils, application plus basique de dessin bitmap multi-calques.

Les objets composites

La boîte à outils fournit aussi un ensemble d'objets composites, comme nous les avons définis dans le chapitre précédent. Ces objets sont des nœuds du graphe de scène encapsulant un sous-graphe de scène prédéfini statiquement : une association de composants éventuellement liée à un modèle interne (comportements et interactions). Construits à partir des objets atomiques (ou extensions de ceux-ci), les quelques objets que nous présentons sont principalement des widgets reprenant les principes des boîtes à outils standard (voir figure 8.8). Par contre, même si leur modèle interne est figé, ils ne sont ni liés à un dispositif d'entrée, ni à des évènements spécifiques, leur comportement global dans l'interface pouvant être défini dynamiquement.

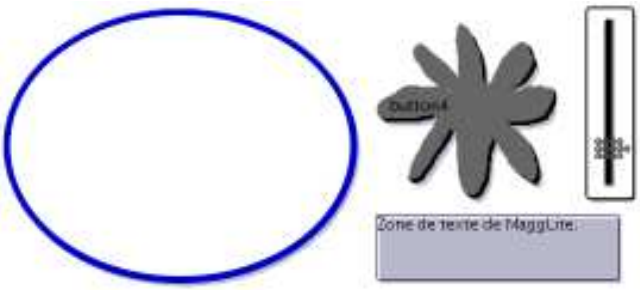


FIGURE 8.8 – Ces objets graphiques composites implémentent des modèles de widgets standards.

Fenêtre
Le composant **Fenêtre** reprend le principe des fenêtres des boîtes à outils standards, avec en plus les possibilités de graphismes et d'interactions avancées de MAGGLITE. Son graphe de scène est composé

d'un nœud racine sans aspect graphique, auquel sont ajoutés deux fils : un bord et un panneau de contenu. Le bord sert à redimensionner la fenêtre, le panneau interne est le conteneur dans lequel peuvent être ajoutés d'autres objets graphiques. Le composant racine est le modèle de l'objet fenêtre, il implémente les comportements et interactions entre lui et ses fils (gestion du redimensionnement global lors de la manipulation du bord, ajout d'autres fils dans le panneau de contenu, etc.).

Bouton
L'objet composite **Bouton**, comme son nom l'indique, reprend le principe du widget bien connu des interfaces standards, présentant tout de même la particularité de ne pas être limité à une forme géométrique rectangulaire ou elliptique.

Slider
Le **Slider** est un objet composite qui implémente le widget standard représentant une glissière. Son graphe de scène associe deux objets, une glissière et une poignée, le comportement de l'un étant lié à celui de l'autre (les mouvements de la poignée sont contraints sur la glissière). Son modèle permet de maintenir une valeur entière dépendant de la position de la poignée.

Zone de texte
La **Zone de texte** agrège un composant graphique rectangulaire et plusieurs labels. Il définit un comportement permettant de créer plusieurs labels et d'adapter leur nombre afin d'afficher le texte voulu dans les limites graphiques du composant.

Mise à part les possibilités graphiques offertes par notre architecture pour la réalisation de ces objets, le principal avantage est qu'ils ne sont pas « câblés » de manière rigide et exclusive comme dans les boîtes à outils usuelles, que ce soit en entrée ou en sortie. Il peuvent alors être manipulés avec des paradigmes d'interaction différents et leurs comportements communs peuvent être définis dynamiquement par leurs manipulateurs qui externalisent l'accès à leur modèle. Par exemple, les comportements des trois widgets de la figure 8.9 page suivante sont configurés graphiquement comme suit avec leurs manipulateurs : le slot de sortie *value* (valeur) de la glissière au centre est connecté au slot d'entrée de la zone de texte à droite par l'intermédiaire d'un adaptateur (conversion de type entier → chaîne de caractères). Lorsque la valeur courante de la glissière change, par manipulation du widget ou par réception d'une nouvelle valeur au slot d'entrée *value* de son manipulateur, la zone de texte affiche cette valeur. Le slot de sortie *action* du manipulateur du bouton (à gauche) est connecté à un dispositif *pass* qui envoie alors la valeur constante du dispositif *Init Value* au slot d'entrée *value* du manipulateur de la glissière. Ainsi, actionner le bouton permet de réinitialiser la glissière à une valeur par défaut.

Cet exemple simple dénote des possibilités d'expression et de prototypage rapide que permettent les graphes d'interaction avec les manipulateurs d'objets graphiques. Mais nous y voyons aussi des possibilités évidentes pour des domaines tels que l'enseignement de l'IHM, une telle approche visuelle et interactive permettant de concentrer le discours sur les fondements plutôt que sur les techniques de programmation. Enfin, cette possibilité de connexion dynamique des entrées-sorties des états des objets d'intérêt de l'interface peut aussi permettre à l'utilisateur final de composer sa propre interface, de décider de quels objets vont devenir des instruments pour en contrôler d'autres.

D'un point de vue plus général sur les objets composites, il serait intéressant de spécifier leur modèle interne par un graphe d'interaction, à l'aide des manipulateurs des objets qui les composent, plutôt que par programmation comme nous l'avons fait jusqu'alors. Par exemple, les contraintes de déplacement portant sur la poignée du *Slider* et sa valeur courante pourraient être spécifiées de la

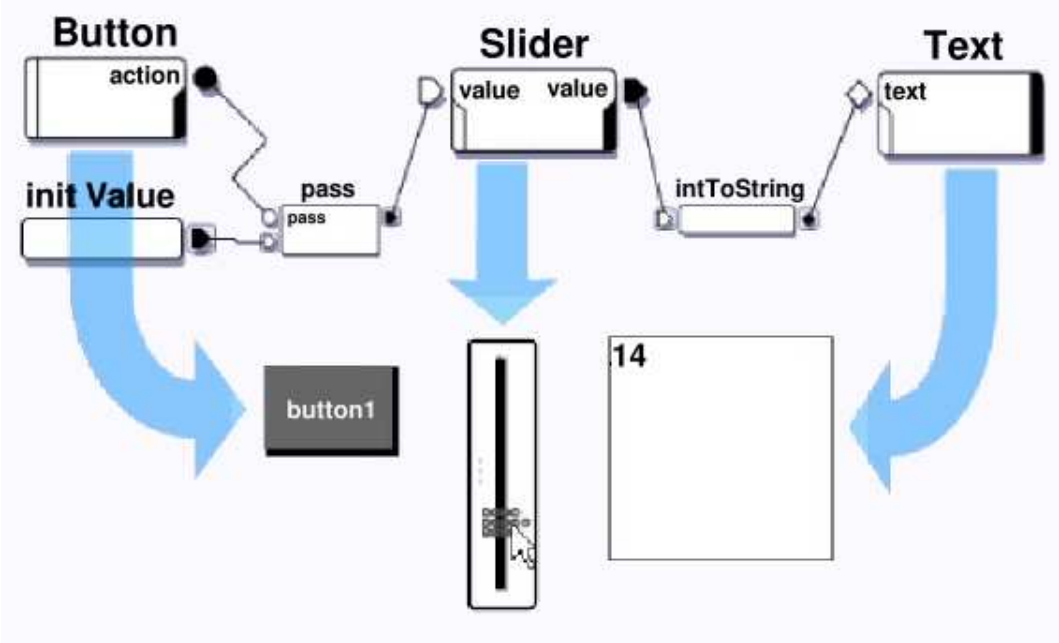


FIGURE 8.9 – Connexion entre manipulateurs. Une configuration permettant de décrire des interactions entre objets à l'aide de leurs manipulateurs.

sorte. Cette approche, encore plus flexible, permettrait le prototypage rapide d'objets composites et donc d'objets complexes réutilisables. Toutefois, cela soulève le problème à la fois conceptuel et technique d'attacher un graphe d'interaction particulier à un nœud du graphe de scène. En effet, bien qu'ICON permette la création de dispositifs composites, encapsulant une configuration d'entrée, il nous parait à l'heure actuelle complexe d'inclure dans notre modèle d'architecture logicielle des manipulateurs d'objets graphiques basés sur ces principes.

8.3.2 Les interactions

MAGGLITE propose un ensemble d'interactions *clés en main*. Réalisées sous la forme de dispositifs d'interaction, de comportements ou d'outils internes (voir le chapitre 7 page 157), elles peuvent être utilisées dans toutes les applications en les insérant dans des graphes d'interaction. Combinées entre elles, avec des manipulateurs d'objets et des dispositifs d'entrée, elle permettent la construction et le prototypage de nombreuses méthodes d'entrée.

Dispositif de sélection

Le **Dispositif de sélection** (*pick*) permet d'obtenir les objets graphiques situés à une position précise dans l'espace des coordonnées écran. Il communique avec le graphe de scène de manière à envoyer en sortie les références des objets situés sous la dernière position reçue par ses slots d'entrée (voir figure 8.10 page suivante). Ces références peuvent être transmises à des dispositifs permettant d'interagir avec les objets, comme nous le verrons par la suite.

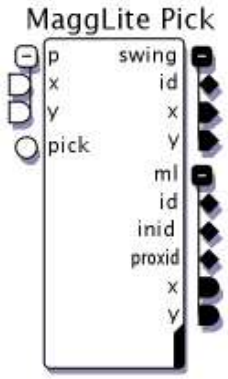


FIGURE 8.10 – Dispositif de sélection (*pick*). Le dispositif de sélection d'objets émet en sortie les références des objets graphiques situés sous les coordonnées reçues en entrée. Ces références sont émises par les slots *ml.inid* lorsque la position reçue est sur l'objet et *ml.proxid* lorsqu'elle est dans la proximité de l'objet (le slot *ml.id* ne fait pas de distinction entre intérieur et proximité). Ce dispositif permet aussi la sélection de widgets Swing (émis par la sortie *swing.id*).

Pour une sélection standard, *sous le curseur*, nous proposons plusieurs stratégies :

1. ne renvoyer que l'objet le plus « haut » ;
2. ne renvoyer que l'objet le plus « bas » ;
3. renvoyer tous les objets sous la position (triés du plus haut au plus bas, ou inversement).

L'objet, ou la liste d'objets, sélectionné(s) est alors envoyé(e) par le slot de sortie *ml.inid*.

Nous avons étendu cette sélection par une méthode de *sélection par proximité*. Le dispositif de sélection permet de déterminer si la position reçue est dans la proximité d'un objet. De la même manière que précédemment, l'objet, ou la liste d'objets, sélectionné(s) par proximité est alors envoyé(e) par le slot de sortie *ml.proxid*.

Nous proposons une dernière méthode de sélection assimilable à la notion de *focus*. Le dispositif de sélection peut être paramétré pour renvoyer la référence de l'objet le plus proche de la position reçue. Il y a ainsi toujours un objet sélectionné, quelle que soit la position reçue.
Enfin, ce dispositif permet aussi la sélection de composants de la boîte à outils Swing, ceux-ci pouvant être ajoutés dans des graphes de scène MAGGLITE. Les références de ces objets sont émises par le slot *swing.id*.

Dispositifs de manipulation

Ces dispositifs sont des dispositifs d'interaction pour la manipulation directe d'objets graphiques, style d'interaction privilégié des interfaces post-WIMP.

Drag (Glisser-Déposer)
Le dispositif de **Drag** permet la manipulation d'objets graphiques par Glisser-Déposer. Ce dispositif générique n'implémente que l'action de déplacement des objets, sans fournir une sémantique particulière telle que le déplacement d'objets dans des conteneurs compatibles ou le déclenchement d'actions par *Drag and Drop*. Toutefois, il est envisageable d'étendre cette technique de base en décomposant

justement ces actions de plus haut niveau sous forme de dispositifs et en les connectant dynamiquement (création d'un dispositif *Drop*, à connecter à la suite du dispositif *Drag*).

Paper sheet
Le dispositif **Paper Sheet** réalise une interaction de manipulation directe d'objets proposée dans [Beaudouin-Lafon, 2001]. Par analogie au déplacement d'une feuille de papier, cette technique permet d'effectuer simultanément une translation et une rotation sur un objet graphique (voir figure 8.11). Initialement conçue pour effectuer des rotations sur des fenêtres empilées, cette technique peut s'avérer applicable dans d'autres contextes d'utilisation. Sa réalisation sous une forme générique applicable à tous les objets graphiques de MAGGLITE facilite son emploi (nous verrons plus loin un exemple d'utilisation de cette technique).

Pliage
Issue aussi de [Beaudouin-Lafon, 2001], et associée à une *interaction de franchissement* (Crossing-based interaction) dans [Dragicevic, 2004a], la technique du **Pliage** permet de replier un objet graphique, de la même manière qu'une feuille de papier (voir figure 8.11). Ce dispositif permet une gestion globale du pliage des objets graphiques compatibles de la librairie : les objets implémentent la capacité de pliage (géométrie et affichage) et le dispositif permet de paramétrer son déclenchement par l'entrée dans sa proximité ou le franchissement de sa frontière (il est aussi possible de *plier* directement les objets par l'intermédiaire de leur manipulateur).

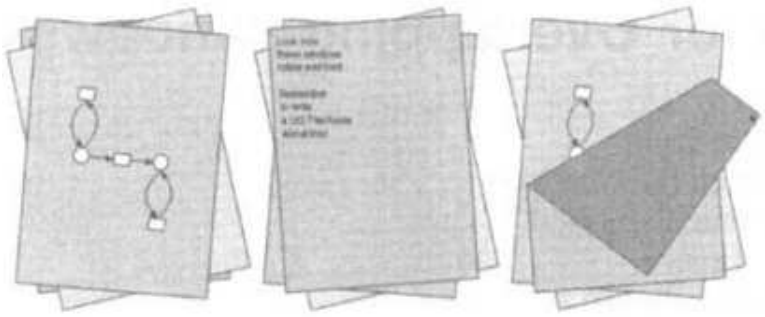


FIGURE 8.11 – Dans [Beaudouin-Lafon, 2001], Michel BEAUDOUIN-LAFON a proposé des techniques de manipulation directe de fenêtres inspirées de la manière dont nous interagissons avec des feuilles de papier. Nous avons inclus ces techniques dans MAGGLITE, les généralisant alors à la manipulation d'objets graphiques.

Responsive Handles
Nous proposons dans MAGGLITE une technique originale pour la manipulation d'objets graphiques : les **Responsive handles**[(5)]. Ces *poignées réactives* étendent le principe des poignées de manipulation, courantes dans les logiciels de dessin 2D ou de modélisation 3D, par exemple. Les poignées sont des instruments, dont la représentation dépend de la fonction, visibles sur le contour de l'objet d'intérêt lorsque celui-ci est sélectionné et permettant de le transformer. Dans la figure 8.12 page ci-contre, l'on peut voir les deux types de poignées les plus courantes : pour redimensionner ou tourner l'objet. Leur répartition autour de l'objet permet d'accéder facilement et rapidement à l'interaction. Toutefois, il est nécessaire de sélectionner l'outil (redimensionnement ou rotation, par exemple), puis l'objet d'intérêt pour avoir accès aux poignées.

[(5)]Nous pourrions donner à cette technique le nom français de *Poignées réactives*. Nous avons tout de même conservé le nom anglais, le terme *Responsive* impliquant aussi la notion de « réagit bien ».

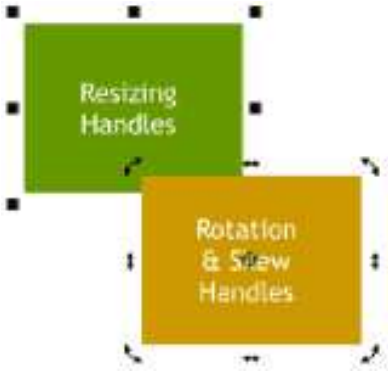


FIGURE 8.12 – Poignées de manipulation.

La technique que nous proposons regroupe ces différentes poignées sur un même instrument réactif qui va apparaître et suivre automatiquement le pointeur lorsque celui-ci entre et se déplace dans la proximité d'un objet (voir figure 8.13). Ainsi, l'interaction devient accessible sans sélection explicite de l'objet et sans que l'instrument n'occupe de la place inutilement. De plus, les Responsive Handles permettent plusieurs manipulations avec un seul instrument : déplacement, redimensionnement ou rotation. Il n'est alors plus nécessaire d'opérer à une sélection préalable de la fonction désirée : celle-ci sera déterminée par la manière de manipuler la poignée, dépendant de la connexion de son dispositif aux périphériques d'entrée. Par exemple, les différents modes peuvent être connectés sur les trois boutons d'une souris. Dès lors, le bouton utilisé pour manipuler la poignée par Glisser-Déposer déterminera la transformation appliquée à l'objet correspondant. Enfin, dans le cas d'objets proches (dont les proximités se croisent), ces poignées permettent d'effectuer la sélection et la manipulation de plusieurs objets en un seul geste.

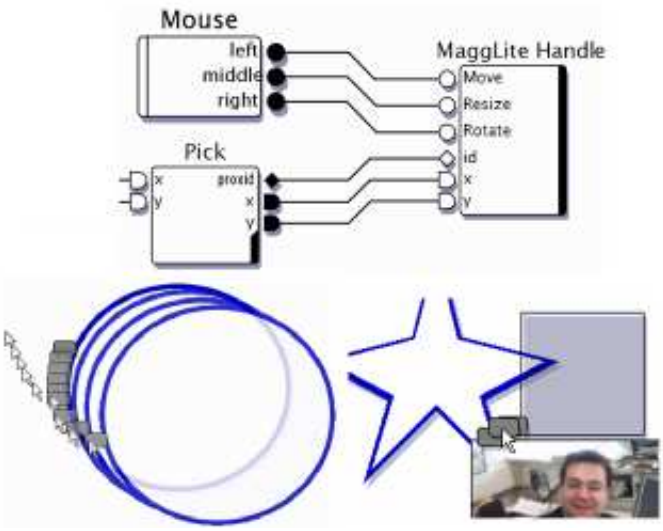


FIGURE 8.13 – Responsive Handles. En haut, une configuration du dispositif avec une souris. La poignée apparaît pour permettre le déplacement de l'objet. Plusieurs poignées peuvent être saisies simultanément, permettant alors la sélection et la manipulation de plusieurs objets en un seul geste.

L'évaluation de cette nouvelle technique pourra nous permettre de confirmer les apports que nous lui supposons. Elle est en plus une piste vers d'autres types d'interactions basées sur le même paradigme d'*interactions de proximité*.

Mais elle est aussi un exemple concret des apports de MAGGLITE pour le prototypage et l'utilisation de nouvelles techniques :

- réalisation rapide : l'implémentation de cette technique n'a en effet pas nécessité un lourd et long travail de programmation comme cela aurait été le cas dans d'autres environnements (nous détaillons cette réalisation dans l'annexe D, section D.2.2 page 55).
- réutilisabilité et interchangeabilité : une fois réalisée sous la forme d'un dispositif d'interaction, cette technique peut être utilisée dans différentes applications, et donc différents contextes d'utilisation, à la place d'autres techniques. Elle n'a pas été exclusivement *câblée* pour une démonstration ou une étude comme c'est souvent le cas lors de la présentation de techniques innovantes. Cela permet de passer rapidement du prototypage à l'utilisation concrète, favorisant alors l'exploration.

Transformations

Nous avons aussi tiré parti de l'architecture modulaire de MAGGLITE pour développer des techniques avancées basées sur des transformations géométriques, souvent utilisées en visualisation d'informations par exemple. L'intérêt est, encore une fois, la relative rapidité avec laquelle ces techniques ont été implémentées sous la forme de dispositifs d'interaction, et la possibilité de les utiliser dans des applications et contextes variées.

La boîte à outil fournit les abstractions nécessaires à la réalisation de ces transformation sous la forme d'interfaces et de classes abstraites Java. Le principe d'implémentation est, pour chaque nouvelle transformation, de réaliser un objet graphique calque qui va être le conteneur dans lequel les objets graphiques seront transformés, la transformation en elle-même et le dispositif d'interaction contrôlant la transformation. Pour construire une application permettant une transformation, il suffit d'utiliser le calque correspondant comme conteneur (au niveau de n'importe quel nœud du graphe de scène). Il sera alors possible, par connexion du dispositif approprié, de déclencher et contrôler la transformation.

Fisheyes

La première transformation que nous proposons comme exemple est une déformation de type **Fisheyes** [Furnas, 1981; Furnas, 1986]. Cette technique, largement utilisée en visualisation d'informations [Fekete, 2004] mais aussi pour l'interaction avec les *fisheye menus* [Bederson, 2000] par exemple, applique le principe de déformation des objectifs photographiques du même nom à la visualisation d'informations.

Le principe est que les objets proches du point d'intérêt (le point focal) sont nets et visibles, et plus ils s'en éloigne, plus le niveau de détail diminue, selon une déformation sphérique (voir figure 8.14 page suivante). Cela permet d'avoir une vue globale sur un grand nombre d'information (la périphérie), tout en conservant la précision et un haut niveau de détails (autour du point focal), cela de manière continue.

Nous avons donc réalisé cette transformation et l'interaction associée (changement du point focal), en réutilisant l'implémentation présente dans la boîte à outils *InfoVis* [Fekete, 2004]. Ainsi, cette

technique de visualisation peut-être connectée selon les besoins et utilisée dans MAGGLITE. Dans la figure 8.14, elle est utilisée pour une application de visualisation d'informations (partie gauche de la figure). Mais nous voyons aussi qu'elle peut être utilisée facilement dans une application standard, avec des *widgets* (partie droite de la figure). Cette utilisation ne présente pas un grand intérêt du point de vue interaction et utilisabilité, mais elle montre que la généricité et la flexibilité de MAGGLITE encourage l'utilisation de techniques dans des contextes différents, permettant l'exploration et la découverte de nouveaux paradigmes.

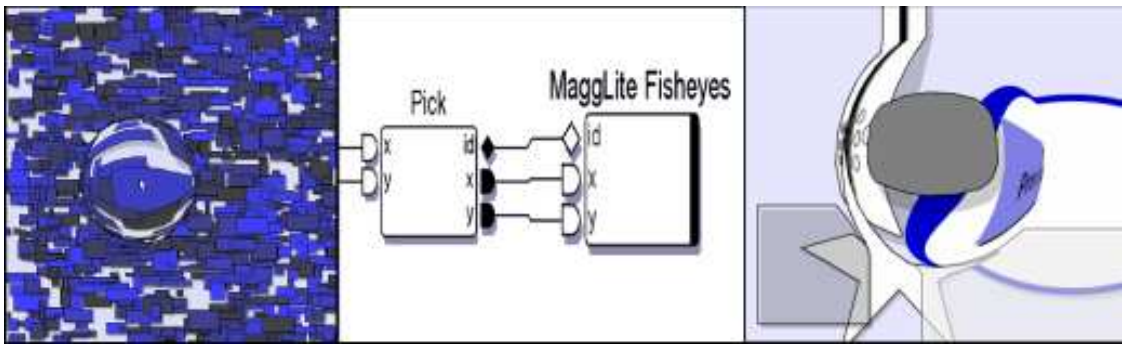


FIGURE 8.14 – Utilisation d'une déformation *fisheyes*. À gauche, pour une application de visualisation d'information. Au centre, la connexion du dispositif. À droite, avec des *widgets*.

Zoom adaptatif à la densité
Nous avons utilisé cette architecture pour prototyper une autre transformation, dont le principe est proche du *fisheyes*. Celle-ci consiste à appliquer un agrandissement proportionnel à la quantité d'objets présents autour du point d'intérêt, d'où son appellation *adaptatif à la densité*. Ainsi, plus il y a d'objets dans le voisinage, plus l'agrandissement est important, offrant alors une meilleure vue. Cette visualisation peut être renforcée par un effet visuel de *flou* (voir figure 8.15).

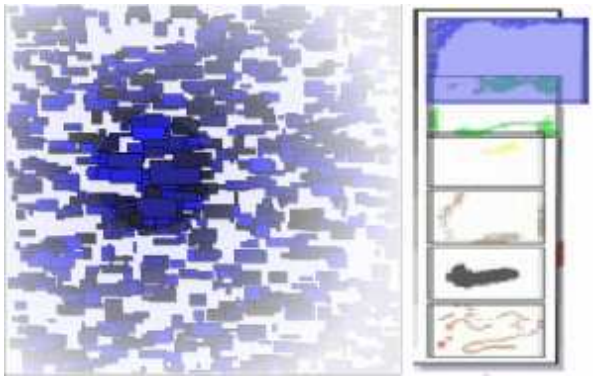


FIGURE 8.15 – Utilisation d'une déformation de *zoom adaptatif*. À gauche, pour une application de visualisation d'information. À droite, avec un composant graphique.

Outre son application à la visualisation d'informations, cette méthode montre aussi un intérêt pour d'autres applications, comme le composant de visualisation des calques de dessin utilisé dans SVALABARD (voir figure 8.15), même si dans ce cas, l'aspect adaptatif n'est pas pris en compte (les composants à agrandir sont répartis de manière uniforme).

Comportements

Les dispositifs de comportement par défaut sont au nombre de trois dans MAGGLITE :

1. la **Pulsation**, basée sur l'opacité des objets graphiques et qui permet de faire un effet cyclique d'apparition et de disparition plus ou moins rapide.
2. le **Collage**, effet graphique principalement utilisé pour distinguer les objets inactifs. Cet effet permet de désactiver l'ombre projetée par un composant graphique et d'augmenter sa transparence, comme si celui-ci était *collé* à son conteneur.
3. le **Surlignage**, permettant d'insister sur les contours et la forme d'un objet afin de le mettre en avant par rapport aux autres. C'est une technique bien adaptée pour distinguer un objet d'intérêt, en particulier lorsqu'elle est associée à une opération de sélection.

Dans l'exemple de la figure 8.16, la poignée de la glissière est connectée à un dispositif de comportement *Surlignage* déclenché lorsque celle-ci est sélectionnée. Les deux autres objets graphiques sont connectés à des dispositifs de *collage*, actifs lorsque les objets ne sont pas sélectionnés.

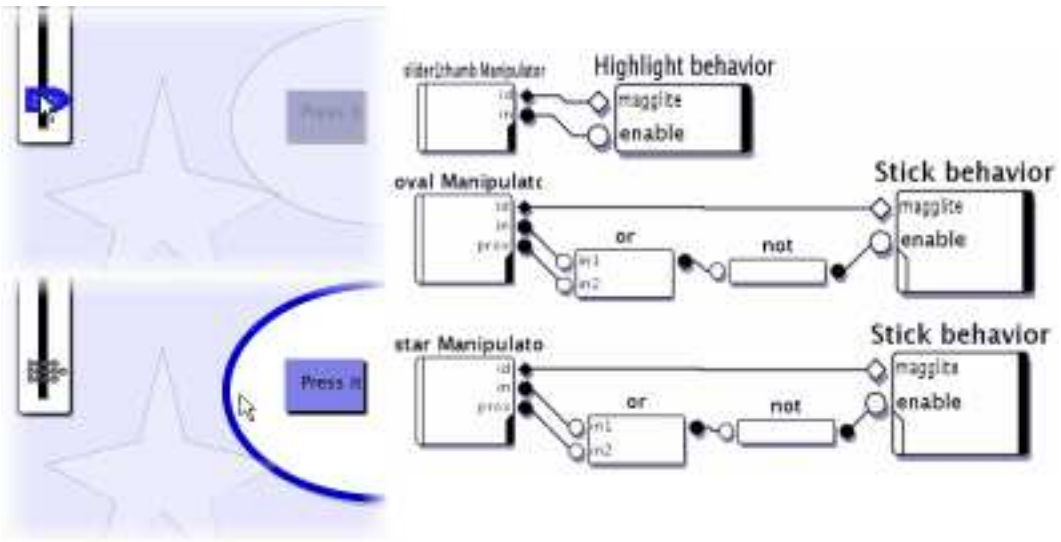


FIGURE 8.16 – Comportements. En haut à gauche, la poignée de la glissière est surlignée, et les autres objets graphiques sont *collés*. En bas à gauche, la fenêtre ovale n'est plus collée. La configuration des dispositifs est représentée dans la partie droite de la figure.

Outils Internes

Nous avons réalisé trois outils internes : un outil de dessin, un de gomme et un outil interne *générique*. Les deux premiers permettent de tracer simplement des traits et de les effacer sur des objets graphiques compatibles. Le troisième permet de restreindre une technique d'interaction déjà définie (sous la forme d'un dispositif d'interaction) à l'intérieur des limites graphiques d'un composant graphique.

Dessin et gomme
Les **outils internes de dessin et de gomme** implémentent une action de dessin ou d'effacement.

Ils peuvent être utilisés avec les *calques de dessin* que nous avons présentés précédemment, ainsi qu'avec tout autre objet graphique prévu pour (en implémentant l'interface Java correspondante). Ainsi, comme le montre la figure 8.5 page 186, il est très simple de réaliser des interfaces de dessin, que ce soit sur l'espace de travail ou à l'intérieur de composants graphique. De plus, cette approche permet de contrôler plusieurs outils simultanément, à l'aide de dispositifs différents dans le même objet graphique ou non (pointeurs multiples). Cela promets des possibilités intéressantes pour des interfaces collaboratives, par exemple.

Outil interne générique

L'**outil interne générique** présente la particularité de ne pas réaliser d'action spécifique sur le composant graphique auquel il a été ajouté (si l'on exclus évidemment son retour graphique). Il dispose en effet de trois slots de sortie supplémentaires (*inside.use*, *inside.x* et *inside.y*) qui propagent les valeurs reçues en entrée uniquement lorsque celles-ci sont dans les limites du composant graphique associé. Lors de la présentation des outils internes du chapitre précédent, nous avions évoqué un outil interne nommé *gesture InnerTool*. Cet outil est en fait une composition de dispositifs comme illustré dans la figure 8.17.

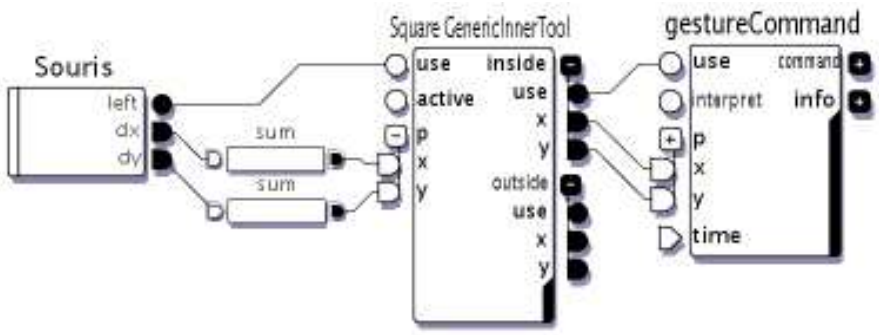


FIGURE 8.17 – Outil Interne générique. L'outil interne générique, associé à un composant graphique nommé *square*, ne transmets ses valeurs d'entrée que lorsqu'elles sont dans les limites graphiques du dit composant. Le dispositif de reconnaissance gestuelle est donc activé dans ce cas.

Un outil interne générique est associé à un composant graphique nommé *square*. Les slots de sortie *« inside »* de cet outil interne sont connectés aux slots d'entrée d'un dispositif de reconnaissance de gestes. En mode global, lorsque les valeurs reçues en entrée de l'outil interne générique sont dans les limites du composant graphique, le dispositif de reconnaissance de gestes est activé. Sinon, les valeurs sont transmises par les slots de sortie *« outside »*, comme pour tout outil interne en mode global[(6)]. C'est la technique que nous avons utilisée pour réaliser la zone de reconnaissance de gestes de SVALABARD.

Cet outil interne dit *générique* permet donc de restreindre une technique d'interaction, ou tout autre dispositif positionnel d'action, à l'intérieur des bornes d'un objet graphique. Il peut être vu comme un *filtre* de coordonnées, bien que proposant plus de fonctionnalités telles que le retour graphique paramétrable (curseur de l'outil interne), ou les différents modes de fonctionnement (local ou global).

[(6)]En mode local, le dispositif connecté à l'outil interne générique serait toujours actif.

8.4 Extension, prototypage et développement d'applications

8.4.1 Extension de la boîte à outils

L'architecture modulaire des graphes combinés employée dans MAGGLITE favorise son extension. Du point de vue graphique, ajouter un nouvel objet consiste en la réalisation d'un nouveau type de nœud de graphe de scène. Du point de vue de l'interaction, il suffit de développer un dispositif approprié. Dans les deux cas, la dérivation de classes abstraites où l'extension des classes de base de la boîte à outils facilite cette démarche.

Objets graphiques

La création de nouveaux objets graphiques peut se faire à partir des classes abstraites *MaggLite* et *MaggLiteContainer*, ou par extension des objets existants, en particulier les calques.

Dans le premier cas (à partir des abstractions), la nouvelle classe d'objets va devoir réaliser plusieurs méthodes abstraites, en particulier les méthodes concernant son affichage. Le développeur va aussi pouvoir utiliser le principe de la génération automatique de slots sur le manipulateur de l'objet (voir l'annexe C, section C.3 page 39).

L'utilisation des calques permet d'étendre les capacités des objets graphiques existant de la boîte à outils, sans pour autant avoir à en développer de nouveaux à partir de rien. En effet, l'ajout d'un calque dans un objet va permettre de lui ajouter des capacités graphiques, mais aussi d'autres techniques d'interaction. Cela facilite l'évolution des objets et leur combinaison, tout en évitant la prolifération de classes dans la librairie. Par exemple, nous avons vu qu'il y avait plusieurs classes de composants graphiques de base (composants, composants transparents, composants image, etc...). La création d'un calque de dessin permet de le composer avec ces différentes classes de composants afin de créer des composants de dessin, des composants transparents de dessin, des composants image de dessin.

Enfin, la déclaration des capacités des objets graphiques pour leur compatibilité avec les interactions génériques de MAGGLITE se fait par l'utilisation d'interfaces : *MaggLiteMoveable* pour les objets que l'on peut déplacer, *MaggLiteDrawable* pour les objets sur lesquels il est possible de dessiner, etc. Il suffit alors de réaliser les interfaces correspondantes aux fonctionnalités désirées, ou d'en créer de nouvelles pour étendre les capacités et interactions associées.

Interactions

De la même manière, l'ajout de nouvelles interactions se fait par l'utilisation des abstractions adaptées (dispositif d'interaction, comportement ou outil interne). Dans chacun des cas, la méthode est relativement proche : spécifier les classes d'objets graphiques auxquels peut s'appliquer l'interaction et implémenter une ou plusieurs méthodes correspondants à l'action à réaliser. Pour les dispositifs d'interactions, ces méthodes correspondent aux action à effectuer lorsque l'objet est sélectionné, lorsqu'une action est réalisée alors qu'il est sélectionné, etc. Dans le cas des comportements, il suffit de spécifier les actions à effectuer sur l'objet graphique à l'activation et à la désactivation du comportement. Enfin, pour les outils internes, les méthodes à réaliser correspondent aux différentes actions de l'outil sur l'objet graphique (activation, désactivation, mouvement ou action).

Nous proposons en annexe D page 45 des exemples concrets de la réalisation d'objets graphiques et de dispositifs d'interaction, de comportement et d'outils internes.

8.4.2 Une plateforme de prototypage : MAGGLITE Interface Builder

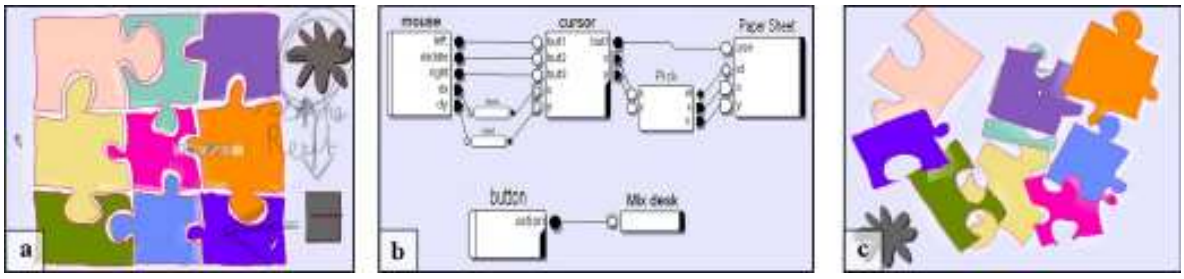


FIGURE 8.18 – Construction d'interfaces avec MAGGLITE Interface Builder. (a) Dessin de l'interface. (b) Configuration des interaction. (c) Utilisation de l'application.

Mais nous proposons aussi avec MAGGLITE une méthode interactive, directe et simple pour prototyper des interfaces post-WIMP. Nous avons utilisé les possibilité de la boîte à outils pour développer un système basé sur le dessin et d'autres interactions avancées. Concevoir une interface suit un alors processus en trois étapes :

1. **Dessiner l'interface** : en utilisant le constructeur d'interfaces basé sur le dessin et les gestes (*MIB* pour MaggLite Interface Builder), un concepteur d'interfaces dessine les composants directement sur l'espace de travail. Ces objets sont immédiatement utilisables et manipulables après leur création. C'est la construction des graphes de scène.
2. **Connecter les interactions** : avec ICON, les interactions sont décrites graphiquement avec à disposition toute la palette de dispositifs d'entrée potentiels et d'interaction avancées proposés. Des maquettes très évoluées peuvent être construites sans écrire une seule ligne de code. C'est la construction des graphes d'interaction.
3. **Utiliser l'interface** : l'interface est entièrement fonctionnelle pendant la création ; elle peut être testée, raffinée et enregistrée. Une application MAGGLITE générique peut alors être utilisée pour recharger le couple *{Graphe de scène, Graphe d'interaction}* qui vient d'être créé. Le graphe d'interaction peut aussi être réutilisé indépendamment comme base d'interaction pour d'autres applications.

Pour illustrer cette démarche, nous proposons d'observer Geoffrey, un concepteur d'interfaces qui va créer une interface de jeu de "puzzle" interactif avec MAGGLITE.

En tant que graphiste et concepteur d'interfaces, Geoffrey a quelques notions de base en programmation. Il préfère toutefois les outils interactifs pour pouvoir exprimer toute sa créativité lorsqu'il conçoit une interface. Il doit de plus fournir des applications fonctionnelles, pas seulement des images de prototypes ou des captures d'écran. C'est une contrainte forte de la conception d'interfaces post-WIMP et d'interactions avancées afin de pouvoir tester et évaluer la faisabilité et l'utilisabilité de paradigmes originaux. Dans le scénario que nous proposons, Geoffrey doit concevoir une interface pour un jeu de puzzle simple et intuitif. Nous allons montrer comment les outils intégrés à MAGGLITE vont l'aider dans sa tâche.

Dessiner l'interface

Pour concevoir la partie graphique de son interface de puzzle, Geoffrey va utiliser les outils de dessin de MAGGLITE Interface Builder (*MIB*). Ce constructeur d'interfaces pour MAGGLITE a la particularité d'avoir été conçu avec MAGGLITE. Ainsi, nous avons pu utiliser la vaste palette d'interactions et de techniques avancées de la boîte à outils. Les différents outils sont sélectionnés à l'aide d'une barre d'outils transparente (*toolglass* [Bier *et al.*, 1993]). Chaque fois que Geoffrey dessine un trait à partir d'une zone de la toolglass, un objet du type correspondant est créé ou une commande est appliquée à l'objet graphique situé en dessous.

1. *Dessin à main levée* : Geoffrey dessine chaque pièce du puzzle en commençant ses traits dans la zone « Composant » de la toolglass (voir figure 8.19). Cet outil est utilisé pour dessiner des composants graphiques de forme libre, ce qui est bien adapté à notre exemple de puzzle.

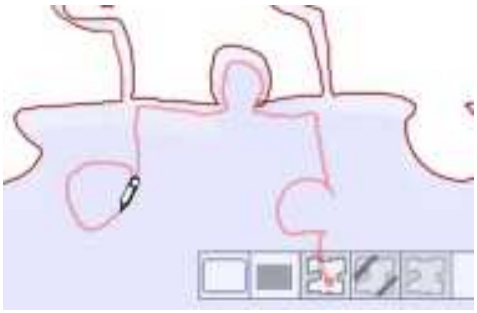


FIGURE 8.19 – Dessin d'un composant avec *MIB*.

2. *Changement de couleur* : Geoffrey peut changer la couleur des pièces du puzzle en plaçant la dernière zone de la toolglass au dessus de l'objet et en cliquant dessus avec son outil de dessin (voir figure 8.20). Les attributs de cet outil de changement de couleur (couleur d'avant plan et d'arrière plan de l'objet) peuvent évidemment être changés interactivement.

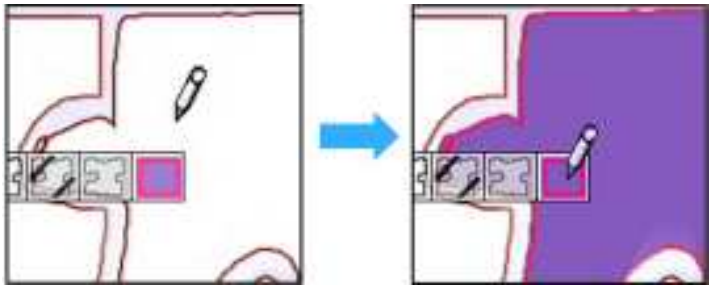


FIGURE 8.20 – Changement de couleur d'un composant avec *MIB*.

3. *Reconnaissance de gestes* : après avoir dessiné les pièces du puzzle, Geoffrey ajoute un bouton pour déclencher une action de mélange automatique des pièces sur la zone du jeu. En réalisant le geste correspondant, il créé un bouton rectangulaire (voir figure 8.21 page ci-contre). Lorsqu'un geste est reconnu, il est interprété et l'objet correspondant est créé ou une action est exécutée (suppression ou déplacement d'un objet, sauvegarde, ouverture du configurateur d'interactions, etc.). Si les gestes ne sont pas reconnus, les traits dessinés sont considérés comme des annota-

tions (voir figure 8.21, vignette (c)). Finalement, Geoffrey souhaite concevoir un bouton moins austère qu'un simple bouton rectangulaire. Il dessine alors un bouton de forme plus extravagante (voir figure 8.18 page 199, vignette (a)) en utilisant l'outil de dessin à main levée avant d'effacer le précédent avec un geste.

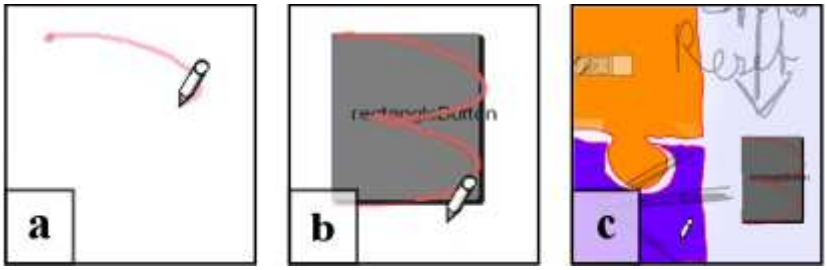


FIGURE 8.21 – Création de composants par interaction gestuelle. (a) Début d'un nouveau geste. (b) Le geste est reconnu, le composant correspondant est créé. (c) Le geste n'est pas reconnu, les traits sont gardés comme annotations.

Ces pièces de puzzle sont une application concrète des possibilités qu'offre MAGGLITE pour la création d'objets graphiques de forme libre. De plus, étant donné que *MIB* a été développé avec MAGGLITE, il peut tirer partie des techniques avancées disponibles : la toolglass, le dessin à main levé et la reconnaissance de gestes sont contrôlés avec une tablette graphique, rendant alors la conception d'interfaces simple et naturelle (comme le proposait déjà SILK [Landay et Myers, 1995; Landay, 1996]). Bien évidemment, *MIB* est entièrement reconfigurable pour l'utilisation d'autres dispositifs et interactions.

Configurer et utiliser l'interface

Lorsque Geoffrey ajoute de nouveaux composants, il va pouvoir construire un graphe d'interaction, comme nous l'avons vu dans la section 7.3 page 163. Il commence par connecter les sorties du dispositif *mouse* aux entrées d'un dispositif *cursor* (étape (1) de la figure 8.22 page suivante). Ensuite, il va connecter les dispositifs d'interaction *pick* et *Drag* afin de pouvoir sélectionner et déplacer les pièces (étapes (1) et (2) de la figure 8.22 page suivante). Au bout de quelques manipulations de pièces du puzzle, Geoffrey réalise que le Glisser-Déposer n'est pas une technique bien adapté à son application car il ne peut effectuer que des translations sur les pièces. Donc il reconfigure l'application et remplace le dispositif *Drag* par le dispositif *Paper Sheet*. Il peut maintenant bouger et tourner les pièces en un seul geste.

Finalement, Geoffrey doit décrire le comportement du bouton de mélange des pièces du puzzle. Il va pour cela utiliser le dispositif manipulateur du bouton qui externalise un slot de sortie *Action* dont la valeur est *vrai* lorsque le bouton est enfoncé. Au cours de la première phase de la conception de l'interface, le bouton n'est connecté à aucune action. En effet, le fait de *mélanger* les pièces du puzzle n'est pas une interaction commune pouvant être décrite simplement par des dispositifs. Cela tient plutôt du noyau fonctionnel de l'application. Geoffrey demande donc à un programmeur de son équipe de lui créer un dispositif *Mix Desk* pour effectuer cette tâche. Une fois ce nouveau dispositif ajouté à la bibliothèque, Geoffrey connecte le slot de sortie du dispositif manipulateur du bouton au slot d'entrée du dispositif *Mix Desk* (voir figure 8.23 page suivante) et l'action peut être effectuée. Cette fonction de mélange des pièces est la seule partie de l'application qui a nécessité d'être programmée. Il en serait de

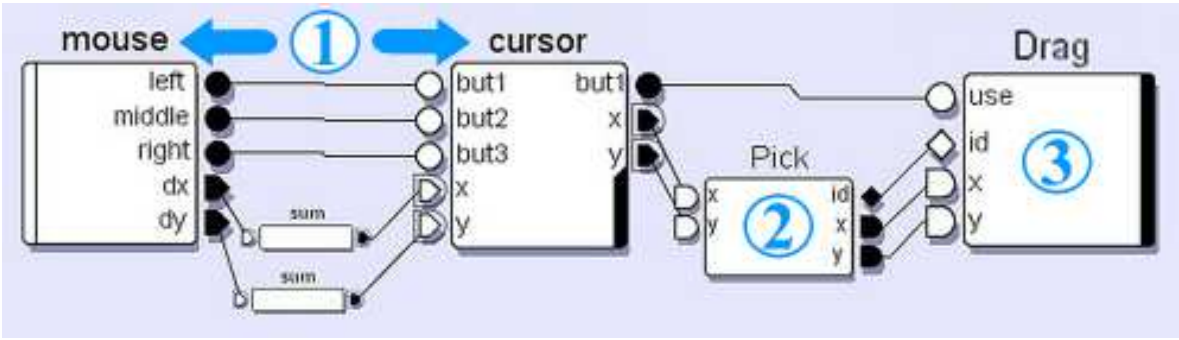


FIGURE 8.22 – Configuration de l'interaction en 3 étapes.

même pour des fonctions très avancées telles qu'une reconstruction automatique du puzzle, un calcul de score ou une fonction d'aide à la reconstruction. Toutefois, une fois implémentées sous forme de dispositifs, les comportements de ces fonctions (déclenchement, affichage des résultats, etc.) seraient spécifiés graphiquement avec le configurateur. De plus, ces nouveaux dispositifs seraient disponibles pour d'autres applications, une fois incorporés à la bibliothèque. Notons aussi que des actions plus simples peuvent être construites graphiquement avec les dispositifs déjà existant, ou en utilisant le dispositif de script inclus dans ICON et qui interprète du code JavaScript.

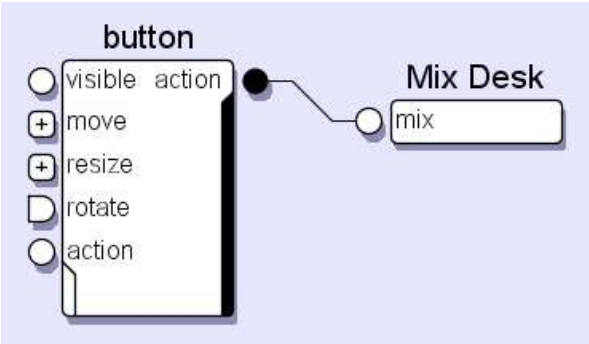


FIGURE 8.23 – Configuration d'un widget.

Utiliser, améliorer et distribuer l'application

Une fois terminé la conception de son application, Geoffrey l'enregistre sous la forme d'un fichier XML qui pourra être rechargée pour des éditions futures, mais aussi utilisé comme une application dans l'environnement d'exécution de MAGGLITE. Bien que sauvegardés, les gestes et annotations de créations ne seront alors plus visibles. Il est évidemment possible pour Geoffrey ou un autre utilisateur de reconfigurer entièrement les interactions de l'application. Par exemple, il peut remplacer la souris par une tablette graphique pour le déplacement des pièces, connecter une commande vocale pour le mélange des pièces. Il est même possible de dupliquer la configuration pour manipuler les pièces du puzzle à plusieurs utilisateurs avec des pointeurs multiples. Toutes ces configurations peuvent être sauvegardées et proposées comme des alternatives dépendant des goûts, capacités ou configurations

matérielles.

Nous avons construit nous-mêmes cette application de puzzle en moins d'une dizaine de minutes. Nous avons mis en scène ce scénario pour montrer comment MAGGLITE et les outils associés peuvent être utilisés pour le prototypage rapide et interactif d'applications post-WIMP interactives. Il est évident que cette approche montre ses limites pour le développement d'applications plus complexes, ce que nous abordons dans la section suivante.

8.4.3 Une plateforme de développement

La conception d'interfaces avec le constructeur graphique ne permet actuellement que l'utilisation d'objets graphiques de la librairie, qui plus est dans une démarche statique (l'application ne peut pas elle-même créer d'objets graphiques). Il est nécessaire, pour des applications plus complexes, d'avoir recours à la programmation en Java. Nous présentons dans la suite deux exemples de développement d'applications : un outil de visualisation d'informations, basée sur la création dynamique d'objets graphiques, et enfin SVALABARD, le système proposé en première partie de cette thèse.

Application à la visualisation d'informations

MAGGLITE n'est pas une boîte à outil spécialisée dans la construction d'applications de visualisation d'informations, et ne prétend pas résoudre les problèmes qu'abordent ces outils dédiés. Toutefois, elle permet le développement et l'exploration rapide de telles techniques. « MaggLiteVisu » est un exemple d'application qui permet la visualisation de données multivariées par une technique étendue de « Scatter Plots » [Chambers *et al.*, 1983]. Les données manipulées peuvent par exemple être un ensemble de données telles que des noms de villes, leur population, superficie, taux de criminalité, etc. Chacune des entrées de l'ensemble, chargé à partir d'un fichier XML, est représenté par un objet graphique rectangulaire. Ses attributs sont alors assignés aux propriétés graphiques de l'objet : sa position, sa hauteur, sa largeur, sa couleur (voir figure 8.24 page suivante).

À partir du squelette d'application de la boîte à outil, la réalisation de cette technique de visualisation bien connue nous a pris environ une dizaine de minutes, pour une trentaine de lignes de code écrites (une grande majorité étant d'ailleurs dédiée à la lecture du fichier XML). C'est relativement peu par rapport à l'implémentation nécessaire pour réaliser une telle application en utilisant les outils de développement habituels, tout particulièrement si l'on considère l'utilisation de techniques avancées telle que le *Fisheyes*.

Réalisation de SVALABARD

L'architecture de MAGGLITE et les différents composants qu'elle contient nous a beaucoup facilité la réalisation de SVALABARD, du point de vue graphique mais surtout pour la définition de ses interactions.

Les feuilles d'interaction
Les trois feuilles d'interaction étendent chacune différents types de calques de la boîte à outils.

– La feuille de dessin étend simplement le gestionnaire de calques, afin de le restreindre à la création de calques de dessin.

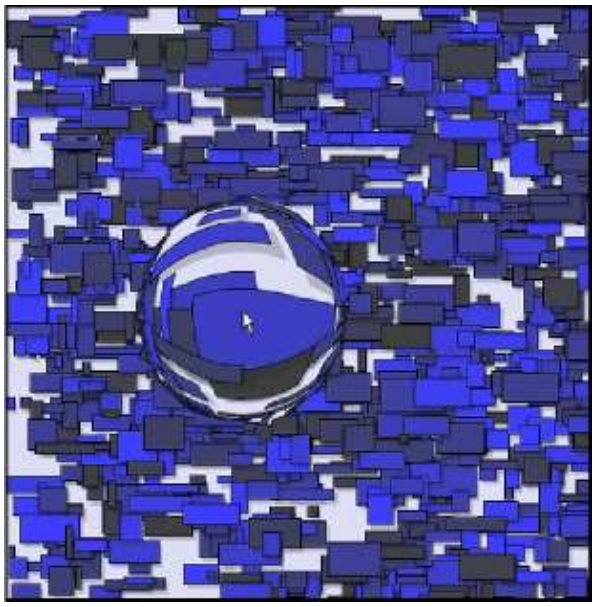


FIGURE 8.24 – Une application de visualisation d'informations construite avec MAGGLITE. Un ensemble de données est affiché, l'utilisateur peut connecter le dispositif de *Fisheyes* générique.

- La feuille augmentée est celle qui a demandé le plus d'adaptation. Elle étend le calque standard pour permettre l'affichage et la manipulation des informations d'interprétation du dessin (points, segments, plans et contraintes).
- La feuille 3D est tout simplement un calque 3D de MAGGLITE, sans modifications particulières.

Les autres objets graphiques, la vue miniature des calques et la zone de gestes, sont des composants « standard » de la boîte à outils.

Configurations d'entrée

Du point de vue de l'interaction, SVALABARD repose évidemment sur les configurations d'entrées d'ICON, associant dispositifs d'entrée et interactions. Il a suffit de connecter les interaction clefs en main de MAGGLITE (interactions, manipulateurs, comportements et outils internes) pour spécifier les comportements et modes d'interactions du système.

Il a par contre été indispensable d'introduire quelques dispositifs spécifiques à l'application :

1. pour le traitement du dessin (les filtres présentés dans la section 4.4 page 110) ;
2. pour la sélection interactive des segments (spécification des contraintes).

Les dispositifs d'interprétation du dessin (cadre bleu de la figure 8.25 page suivante) sont des *adaptateurs*. Ils forment la chaîne de transformation des données émises par les dispositifs systèmes (périphériques d'entrée) pour leur interprétation et utilisation par le noyau mathématique (connecté lui aussi par un dispositif).

Les dispositifs de sélection des segments (cadre rouge de la figure 8.25 page ci-contre) correspondent chacun à un type de propriété à saisir (parallélisme, orthogonalité ou coplanarité). Ils reçoivent simplement une commande et sa zone d'application (provenant de l'interaction choisie comme par exemple un geste, ou une combinaison parole/pointeur), afin de sélectionner et transmettre au noyau mathématique les éventuels segments auxquels appliquer la propriété.

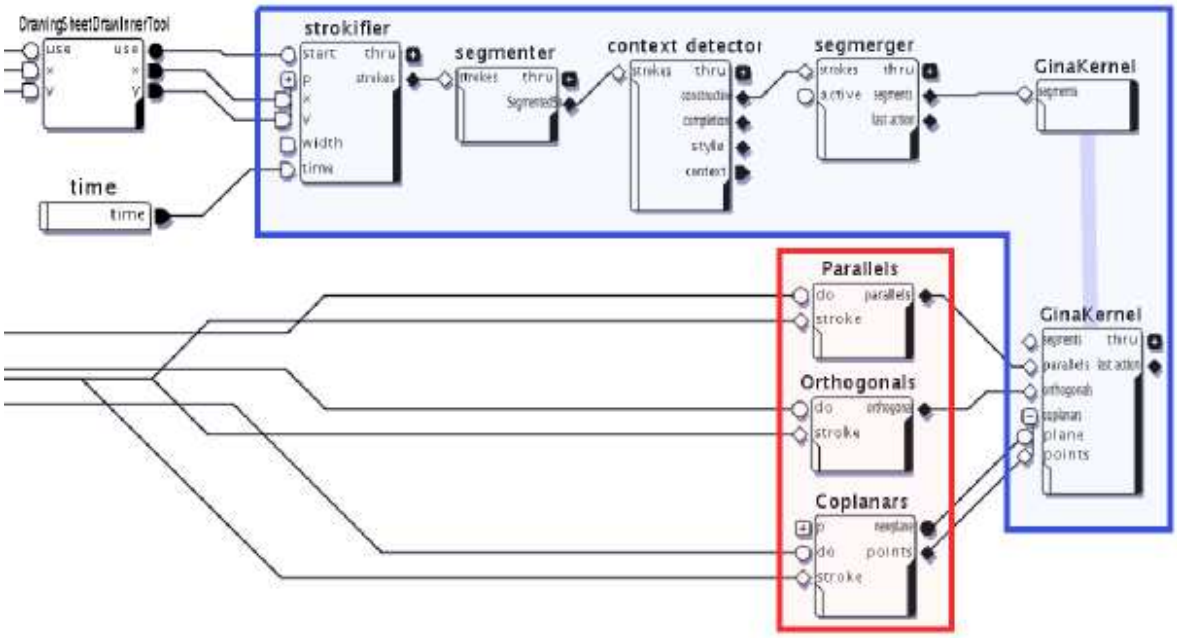
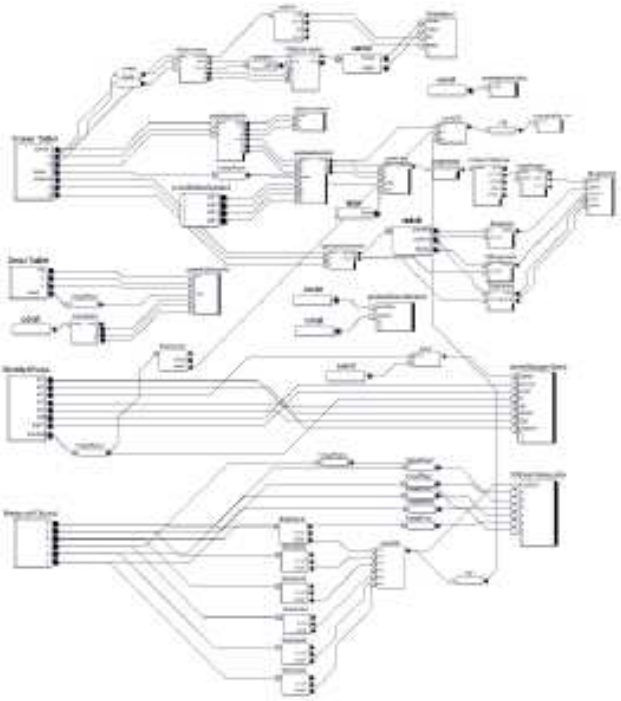


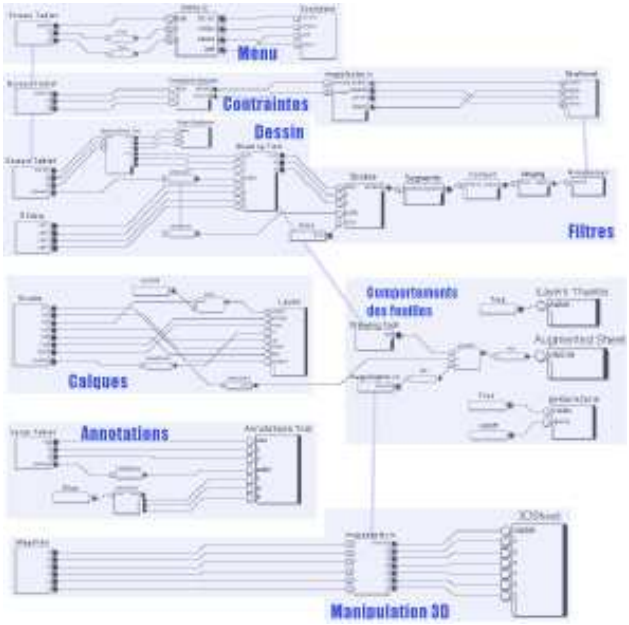
FIGURE 8.25 – Filtres de dessin. Les filtres de dessin sont réalisés sous la forme de dispositifs ICON. Ils peuvent être connectés, déconnectés et configurés graphiquement.

Nous avons définis plusieurs configurations d'entrée pour utiliser SVALABARD, de la plus complète qui spécifie les interactions décrites dans le chapitre 4 page 83, à la plus simple basée uniquement sur l'utilisation d'une souris et d'un clavier, réduisant alors les fonctionnalités et les interactions.
Ces configurations peuvent être utilisées telles quelles, ou modifiées par l'utilisateur à tout moment selon ses envies ou ses besoins. Il est toutefois nécessaire de discuter la lisibilité et la complexité de ces configuration. La complexité de SVALABARD implique des configurations d'une densité importante, comportant un grand nombre de dispositifs et de connexions (voir figure 8.26 page suivante). Nous avons, dans un premier temps, appliqué les recommandations de structuration spatiales et les possibilités avancées du configurateur graphique d'ICON pour améliorer la lisibilité des configurations que nous proposons (utilisation de *liens* pour éviter le croisement des connexions, renommage et dimensionnement des dispositifs, etc).

Mais nous avons aussi utilisé le principe d'encapsulation de sous-configurations pour définir plusieurs niveaux de configurabilité du système. ICON permet de créer des configuration-filles qui seront chargées par un dispositif *composite* dans une configuration globale. La lisibilité est alors améliorée (voir figure 8.26(b) page suivante). Mais cette approche permet en plus une décomposition logique des fonctionnalités et des interactions du système, offrant alors plusieurs niveaux de configurabilité. Pour SVALABARD, nous avons définis plusieurs sous configurations, indépendantes des entrées : définition des marking menus, adaptation des dispositifs, comportements des feuilles, etc. Ces sous-configurations représentent le noyau de l'application. L'utilisateur novice se contentera de changer les dispositifs d'entrée connectés à ces modules. L'utilisateur expert pourra modifier ces sous-configurations pour redéfinir plus profondément les comportements du système.



(a)



(b)

FIGURE 8.26 – Configurations d'entrée de SVALABARD. Ces deux configurations représentent le même graphe d'interaction (une partie de la configuration globale de SVALABARD), seule leur représentation change. Dans la figure (b), l'utilisation des liens et des sous-configurations permettent une meilleure lisibilité (regroupement par fonctionnalités), ainsi que plusieurs niveaux de configurabilité du système.

8.5 Conclusion

Nous avons décrit dans ce chapitre la boîte à outils MAGGLITE, dont l'objectif est l'association et l'intégration de multiples techniques d'interaction avancées avec une architecture graphique complète, dans un environnement logiciel flexible et modulaire. L'ensemble minimal de composants graphiques et d'interactions que nous avons proposé nous a permis de produire des exemples concrets de l'utilisation de MAGGLITE pour la conception d'interfaces post-WIMP basées sur des paradigmes connus ou novateurs. Dans cette optique, la réalisation d'une application complexe telle que SVALABARD nous a permis à la fois de montrer la validité du modèle des graphes combinés, mais aussi d'étendre considérablement les mécanismes de MAGGLITE au fur et à mesure de nos besoins.

Succédant à ICON, reposant largement sur son modèle d'interaction en entrée, la boîte à outils MAGGLITE ouvre des possibilités actuellement hors d'atteinte des autres approches, aussi bien du point de vue de l'extensibilité, que de celui de la flexibilité pour le prototypage et l'étude de nouveaux paradigmes d'interaction ou de la configurabilité et l'adaptabilité des applications. Nous avons, à l'heure actuelle, choisi de distribuer librement l'environnement de développement et d'exécution de MAGGLITE, espérant populariser et améliorer notre approche dans la continuité des travaux entrepris par Pierre DRAGICEVIC (la distribution est disponible sur le site internet de MAGGLITE [Huot, 2004]). Nous envisageons prochainement de diffuser les sources de MAGGLITE dans un contexte de développement de groupe (*SourceForge*, par exemple) afin de réunir suffisamment de collaborateurs pour franchir le pas d'un prototype abouti à un environnement finalisé.

MAGGLITE est une réponse efficace aux problèmes que nous avions soulevés au début de cette partie, problèmes essentiellement apparus lors de la réalisation de SVALABARD et liés à la rigidité et le manque d'extensibilité des outils actuels pour le développement d'interfaces. Mais, bien que plus polyvalente et généraliste que la majorité des boîtes à outils actuelles, notre approche ne prétend pas non plus résoudre tous les problèmes inhérents aux développement d'interfaces et d'interactions post-WIMP. Dans le chapitre suivant, nous proposons de comparer MAGGLITE et son architecture aux approches existantes, tout en identifiant ses faiblesses et les éventuelles voies de recherche qu'elle peut contribuer à explorer.

Chapitre 9

Positionnement et discussions

« Des idées, tout le monde en a. Souvent les mêmes. Ce qu'il faut, c'est savoir s'en servir. »

M. COLUCCI.

Sommaire

9.1 Introduction

Dans ce chapitre, nous comparons MAGGLITE à d'autres approches qui font référence dans le domaine des boîtes à outils d'interaction. Cela concerne aussi bien des outils de programmation et de prototypage que des techniques d'interaction proches de celles utilisées dans MAGGLITE. Cet état de l'art ne se veut pas exhaustif, de nombreux travaux ayant étant réalisés dans ce domaine. Nous avons choisi de présenter les plus significatifs afin de situer nos travaux dans le vaste champ des boîtes à outils d'interaction. Pour un tour d'horizon plus complet, nous invitons le lecteur à lire la thèse de doctorat de Pierre DRAGICEVIC [Dragicevic, 2004b] en ce qui concerne la gestion des périphériques d'entrée et le chapitre du livre « Computer Science Handbook » [Myers, 2004] pour la programmation d'interfaces graphiques.

Nous conclurons alors le dernier chapitre de cette thèse par les perspectives qu'ouvre cette boîte à outils. Mais dans un premier temps, nous proposons une analyse des apports de notre approche vis à vis de la réalisation des modèles sur lesquels elle repose ou avec lesquels elle présente des similitudes.

9.2 MAGGLITE et les modèles

Dans cette section, nous positionnons notre boîte à outils et son modèle concret par rapport aux modèles sur lesquels elle repose (le modèle d'interface *MVC* et le modèle d'interaction de l'*Interaction Instrumentale*) et qu'elle permet de réaliser (*Modèle multi-couches*).

9.2.1 MVC

Le modèle **MVC** (Modèle, Vue, Contrôleur) [Krasner et Pope, 1988] est un modèle à agents interactifs devenu référence dans l'implémentation des applications interactives, par soucis d'adaptabilité et d'adéquation au langage à objets. Les applications interactives sont décomposées en agents *MVC*, chacun présentant :

1. un **modèle**, partie de son noyau fonctionnel pouvant être une donnée ou des objets plus complexes ;
2. une ou plusieurs **vues**, les représentations perceptibles du modèle par l'utilisateur qui sont modifiées selon les changements d'état du modèle ;
3. un ou plusieurs **contrôleurs**, qui reçoivent et traitent les actions de l'utilisateur, les propageant sur le modèle ou sur la vue.

MVC dissocie la gestion des entrées et des sorties dans ses agents, promettant une certaine indépendance entre l'aspect et le comportement des objets graphiques (le modèle **PAC** – *Présentation*, *Abstraction*, *Contrôleur* – [Coutaz, 1987] généralise *MVC*, et regroupe entrées et sorties dans le module de présentation). Toutefois, sa mise en œuvre s'est révélée complexe et une grande majorité des boîtes à outils actuelles fusionnent la vue et le contrôleur dans leurs implémentations. Elles assurent ainsi la flexibilité au niveau modèle, mais gardent une approche monolithique du point de vue de l'interaction (en entrée et en sortie).

Dans la perspective de ce modèle, les configurations d'entrée d'ICON représentent le *contrôleur*, laissant la gestion de la vue et du modèle aux applications. Le modèle des graphes combinés place la *Vue* au niveau des objets graphiques du graphe de scène. Le *modèle* est alors décrit dans l'application.

Ce sont les *Points d'accès à l'Interaction* qui vont permettre d'établir la connexion entre les différents composants de *MVC*. Cette séparation franche des différents composants de *MVC* et leur *connectivité dynamique* est un apport important de notre modèle. Elle offre plus de flexibilité pour le développement d'interfaces et la description d'interactions mais aussi pour l'extensibilité de la boîte à outils que les autres approches basées sur *MVC*.

9.2.2 IAPs et l'interaction instrumentale

Le modèle de **l'interaction instrumentale** est un modèle d'interaction, « *un ensemble de principes de règles et de propriétés qui guident la conception d'une interface* » du point de vue de l'utilisateur [Beaudouin-Lafon, 1997; Beaudouin-Lafon, 2000]. Il étend et généralise le modèle de la **manipulation directe** [Shneiderman, 1983] en s'inspirant de notre façon d'interagir avec les objets du monde réel par l'intermédiaire d'*outils*. Par exemple, le crayon de l'architecte est un *instrument* qui va lui permettre d'agir sur un *objet d'intérêt*, son dessin.

Le modèle de l'interaction instrumentale définit, entre autres principes, les propriétés des **instruments** de l'interface qui vont permettre d'agir et d'interagir sur les **objets du domaine**, les entités du programme informatique. En particulier, il introduit la notion de **réification**, processus qui consiste à transformer un concept en objet (une barre de défilement, par exemple, réifie le concept de défilement d'un document).

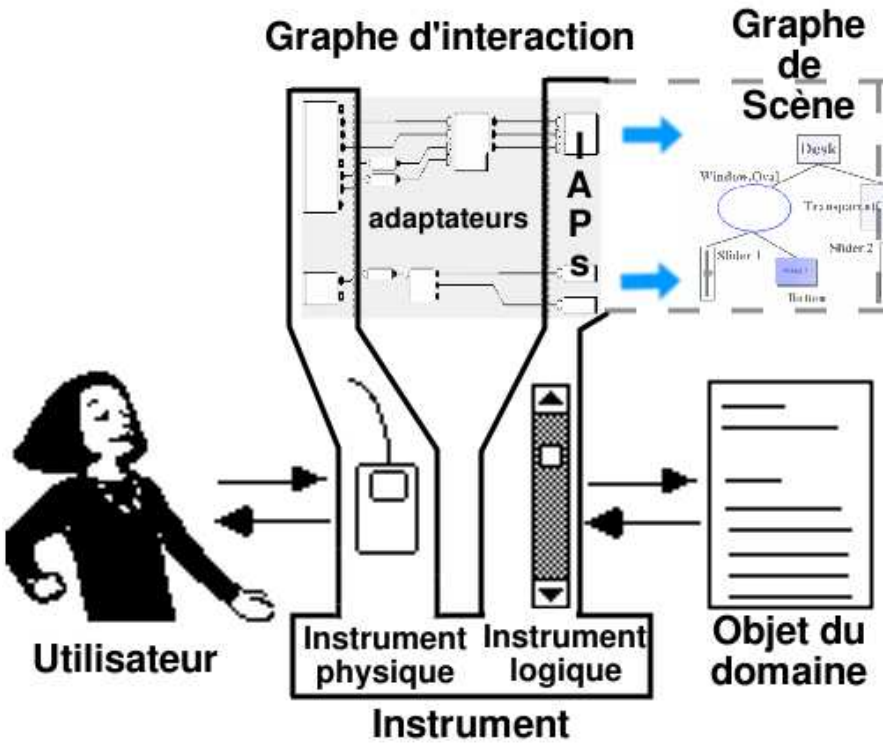


FIGURE 9.1 – IAPs et l'interaction instrumentale. Les IAPs définissent la *partie logique* des instruments, pouvant être connectée à la *partie physique* dans une configuration d'entrée, par l'intermédiaire d'adaptateurs. Les *objets du domaine* sont définis par l'application au niveau du graphe de scène.

Les points d'accès aux interactions, et la *philosophie* de MAGGLITE en général, correspondent bien à ce modèle. Les IAPs s'intègrent dans la description d'*instruments* d'interaction par leur association avec des périphériques d'entrée. Ils définissent, avec certaines parties du graphe de scène (retour graphique ou vue de l'instrument), la composante *logique* des instruments. La figure 9.1 page précédente illustre ce propos en mettant en correspondance le modèle des graphes combinés et l'interaction instrumentale.

Les IAPs suivent le principe de *réification* : ils réifient les commandes qui contrôlent soit l'objet qu'ils représentent (pour les manipulateurs), soit la classe d'objets avec lesquels ils sont compatibles (pour les dispositifs d'interaction et les outils internes). La technique des *Responsive Handles* que nous avons présentée dans la section 8.3.2 page 192, est un exemple d'instrument logique permettant d'interagir avec des objets graphiques. Une fois connecté à un dispositif d'entrée (souris, joystick, magellan, etc.), le dispositif d'interaction et les nœuds du graphes de scène(poignées) composent un instrument qui *réifie* les concepts de déplacement, redimensionnement et rotation d'objets graphiques. De plus il est *polymorphe* [Beaudouin-Lafon, 2000], car il peut être utilisé avec tous les objets pour lesquels il a du sens (les objets déplaçables), mais aussi parce qu'il peut être adapté à d'autres contextes en utilisant diverses méthodes d'entrée (périphériques, reconnaissance vocale, etc.).
Les manipulateurs permettent en plus la manipulation directe des objets avec un *instrument physique* (périphériques d'entrée), poussant alors l'interaction instrumentale jusqu'à sa limite : l'interaction est on ne peut plus directe, physique, et le contrôle est alors plus explicite.

Notre modèle d'implémentation permet donc de séparer explicitement les *instruments physiques* (périphériques d'entrée) des *instruments logiques* (dispositifs d'interaction, outils internes et manipulateurs) et permet leur association dynamique pour créer des *instruments*. Les *objets du domaine* sont décrits au niveau de l'application et représentés dans le graphe de scène. Ce niveau de granularité très fine est un atout pour respecter les principes de conception qui découlent de l'interaction instrumentale (*réification*, *polymorphisme* et *réutilisabilité*), facilitant ainsi le prototypage d'instruments basés sur un grand nombre de techniques d'interaction ainsi que leur intégration dans une même application.

9.2.3 Le modèle multi-couches

Dans [Fekete et Beaudouin-Lafon, 1996; Fekete, 1996], Jean-Daniel FEKETE a proposé une architecture d'implémentation en couches pour, entre autre, séparer la spécification de l'interaction et du graphique dans les applications interactives : le **Modèle multi-couches**. Comme représenté sur la figure 9.2 page ci-contre, cette approche permet une décomposition sémantique des fonctionnalités de l'application graphique en spécialisant les interactions et/ou les visualisations pour chaque couche.

Chaque couche contient des outils qui lui sont propre et qui vont gérer l'interaction. Les événements d'interaction *traversent* les couches de la plus proche de l'utilisateur à la plus éloignée (de haut en bas) et sont *interceptés et bloqués* par les outils capables de les traiter (voir figure 9.2 page suivante). Cette organisation sépare les outils et objets graphiques de nature différente dans des couches séparées. Cela permet une description claire de l'interaction, mais aussi de facilement adapter la visualisation aux actions (masquage ou transformation d'une couche, par exemple).

MAGGLITE ne repose pas exclusivement sur ce modèle multi-couches, mais permet de le réaliser efficacement et d'en tirer partie pour deux objectifs : *réaliser des applications* pour lesquelles ce modèle est adapté (des éditeurs comme **TicTacToon** [Fekete *et al.*, 1995] où notre application **SVALABARD**) ou *étendre les capacités graphiques et interactives de composants*.

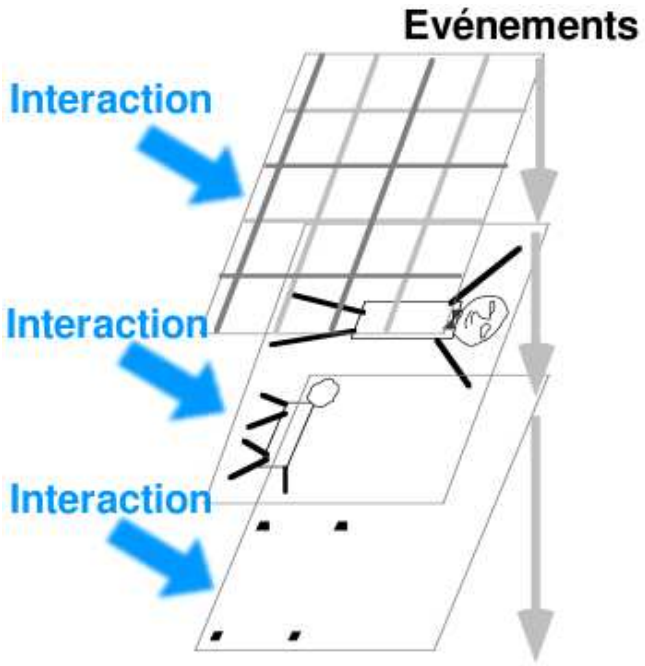


FIGURE 9.2 – Le modèle multi-couches décompose une application en couches graphiques interactives. MAGGLITE permet de réaliser ce modèle, en offrant une approche encore plus flexible de l'interaction (une approche horizontale – flèches de gauche – par rapport à l'approche verticale du modèle).

La réalisation d'applications

Nous proposons d'utiliser ce même paradigme des couches pour le développement d'applications graphiques appropriées avec MAGGLITE. SVALABARD en est un exemple, avec le paradigme des feuilles d'interaction. Notre approche permet la même gestion *verticale* de l'interaction (par traversée des couches), mais autorise aussi une approche *horizontale*, par connexion directe de dispositifs ou d'interactions aux outils et propriétés de la couche (voir figure 9.2). Ainsi, la réalisation de cette architecture est encore plus simple et flexible, permettant même la redéfinition interactive du comportement des couches et de leurs outils.

L'extension des composants

Mais au delà de la construction d'applications, ce principe de couches est aussi l'un des fondements de l'extension des composants de la boîte à outils. Comme nous l'avons montré avec l'exemple des calques de dessin, les capacités graphiques et interactives des composants de la boîte à outils peuvent être augmentées par l'ajout dynamique de calques (ou couches) et de leurs outils associés. Cette approche évite de devoir développer plusieurs classes pour ajouter une même propriété à différents types d'objets, tout en favorisant leur composition dans un but exploratoire.

9.3 Boîtes à outils

Il serait difficile de vouloir comparer MAGGLITE à toutes les boîtes à outils proposées aux développeurs pour l'implémentation d'applications interactives. Nous avons donc choisi de la positionner par rapport aux travaux de recherche les plus significatifs dans ce domaine.

9.3.1 Pour les IHM avancées

Bien que plus utilisées pour développer des interfaces « conventionnelles », des boîtes à outils telles que **Garnet/Amulet** [Myers *et al.*, 1997] et **Subarctic** [Hudson et Smith, 1996] ont introduit des fonctionnalités avancées pour simplifier la programmation des interfaces, comme par exemple la gestion automatique d'arrangement spatial basée sur les contraintes. Ils utilisent aussi des mécanismes plus complets, génériques et extensibles que les boîtes à outils traditionnelles pour décrire les entrées et les événements [Myers, 1990; Henry *et al.*, 1990]. Pourtant, même si elles introduisent quelques techniques post-WIMP, comme de la reconnaissance de gestes, ces boîtes à outils ne supportent qu'un ensemble limité de périphériques d'entrée et nécessitent des modifications importantes pour y inclure de nouveaux paradigmes d'interaction. En opposition, la palette de périphériques d'entrée et de techniques d'interaction que propose MAGGLITE est très étendue, mais aussi facilement extensible grâce à l'architecture des graphes d'interaction.

9.3.2 Post-WIMP

La plupart des boîtes à outils dites post-WIMP sont spécialisées dans un paradigme d'interaction. **Satin** [Hong et Landay, 2000], par exemple, étend *Swing* pour y inclure des techniques d'interaction avancées à base de gestes et de traits. Malheureusement, cette extension est basée sur des événements souris standard et ne permet pas l'utilisation de stylets sensibles à la pression et de traits haute résolution comme le fait MAGGLITE avec la gestion avancée des périphériques que fournit ICON.

Jazz [Bederson *et al.*, 2000] est une boîte à outils Java pour développer des interfaces zoomables. Son architecture graphique est aussi basée sur les graphes de scène, avec ce niveau très fin de granularité par rapport aux widgets monolithiques. Mais les auteurs ont reconsidéré leur approche, proposant comme évolution la boîte à outils **Piccolo** [Bederson *et al.*, 2004]. Toujours basée sur les graphes de scène, cette boîte à outils se veut un compromis entre une approche monolithique classique et les approches décomposées trop finement comme *Jazz* qui favorisent, selon les auteurs, une prolifération des classes pouvant nuire à la prise en main et la compréhension des mécanismes de la boîte à outils pour le programmeur. Ainsi, plutôt que de proposer une classe d'objet graphique pour chaque *capacité* envisagée, *Piccolo* propose une classe maîtresse regroupant toutes les fonctionnalités communes, que les développeurs peuvent instancier ou étendre.
Mais cette vision ne prend en compte que les problèmes dus à la réalisation d'objets graphiques, postulant que l'interaction sera définie dans un paradigme standard (les interactions de *Piccolo* sont décrites avec une architecture à événements standard encapsulée dans les nœuds du graphe de scène). Notre approche et notre vision de ce problème sont différentes. Elle conserve un niveau de granularité fin (pour permettre le contrôle externe de l'interaction), tout en proposant des mécanismes efficaces pour éviter la prolifération des classes d'objets graphiques : les *dispositifs de comportement* et les associations *calques–Outils internes*. Ces outils permettent de définir ou d'ajouter de nouvelles capacités graphiques et interactives aux objets de base sans avoir à programmer de nouvelles classes.

Ubit [Lecolinet, 2003] est aussi basée sur les graphes de scène, mais introduit la métaphore d'architecture « moléculaire » : des objets graphiques de base (atomes) sont assemblés pour construire des widgets (molécules). *Ubit* propose un langage de script, plus léger qu'un langage de programmation. Toutefois, les possibilités de prototypage sont limitées par le manque d'un éditeur graphique. Comme **CPN2000/CPN Tools** [Beaudouin-Lafon et Lassen, 2000] et **MMM** [Bier et Freeman, 1991], *Ubit* permet de concevoir des interfaces à pointeurs multiples, basées sur les périphériques de pointage standard, mais ne gère toutefois pas de périphériques et d'interactions avancées.

Plus généralement, les boîtes à outils graphiques post-WIMP gèrent efficacement un ensemble limité de périphériques d'entrées, mais par compatibilité : soit elles utilisent les mécanismes fournis par le système d'exploitation (qui n'est pas loin de supposer que *« tout est souris ou clavier »*), soit elles proposent une implémentation *ad hoc* pour des dispositifs particuliers limitant alors leur extensibilité (physique et logicielle).
De la même manière, elles se concentrent sur un style d'interaction particulier pour un type de tâches (le dessin, par exemple), un problème précis auquel elles répondent efficacement. Leur utilisation dans d'autres contextes, ou en association avec d'autres paradigmes, n'est pas impossible mais nécessite de gros efforts d'adaptation et de développement.

CPN2000/CPN Tools est un exemple flagrant d'application dont le développement à nécessité de partir de zéro pour supporter un modèle graphique et un modèle d'interaction originaux. En effet, une telle application ne peut reposer que sur une boîte à outils construite explicitement dans ce sens : permettre une intégration cohérente de différents types d'interactions avancées (outils transparents, interaction bimanuelle, menus circulaires, etc., voir figure 9.3) [Beaudouin-Lafon et Lassen, 2000].

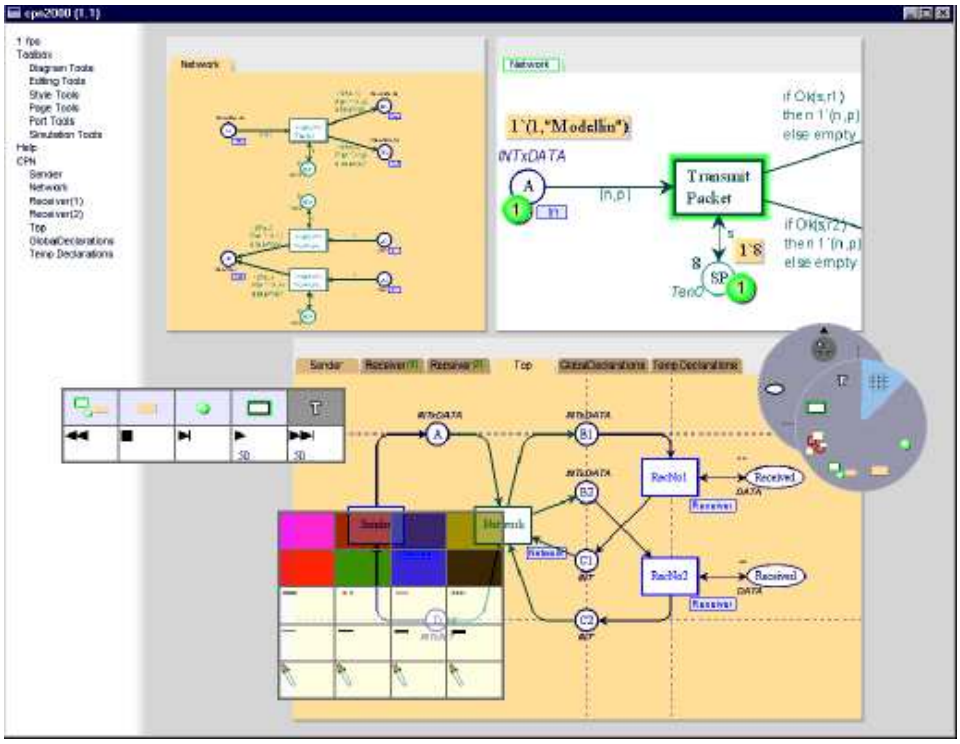


FIGURE 9.3 – *CPN2000/CPN Tools* intègre de nombreuses techniques d'interaction avancées.

Dans ce même article, Michel BAUDOUIN-LAFON considère ce problème sous deux angles :

1. étudier comment ces techniques peuvent êtres combinées entre elles ou avec des techniques standards. Cela requiert, selon lui, de définir de nouveaux **modèles d'interaction** comme l'*interaction instrumentale* utilisé dans ce cas [Beaudouin-Lafon, 2000].
2. proposer les **outils logiciels** basés sur un tel modèle (boîtes à outils) qui vont permettre aux développeur d'introduire des techniques d'interaction avancées « avec autant de flexibilité que de nouveaux widgets » dans une boîte à outils standard.

Il propose alors deux approches pour la réalisation d'une telle boîte à outils.

1. Développer la boîte à outils autour du modèle pour en préserver la généricité. Cette approche soulève alors la question de l'adéquation de ces outils pour la réalisation d'applications complètes et complexes.
2. Construire une application à grande échelle basée sur le modèle pour en isoler les points spécifiques à la réalisation ultérieure d'une boîte à outils.

C'est la deuxième solution qui a été choisie pour la réalisation de *CPN2000* et la boîte à outils n'a, à notre connaissance, pas encore vu le jour.

Bien que SVALABARD, notre application de dessin pour la modélisation 3D *créative*, n'était pas voué comme *CPN2000* à une diffusion à grande échelle, elle reste tout de même une réalisation complexe, plus qu'une simple application *jouet*. Et nous avons sensiblement rencontré les mêmes problèmes d'intégration et de développement. Nous avons par contre proposé une méthode de résolution à la frontière des deux propositions précédentes : le développement conjoint d'une boîte à outils et de l'application, l'une permettant l'évolution continuelle de l'autre[(1)]. Nous avons alors pu conserver la généricité et la flexibilité de la boîte à outils, tout en prouvant son adéquation au développement d'applications complètes.

9.3.3 Entrées multiples

Récemment, beaucoup de travaux ont étés menés sur des boîtes à outils pour les entrées multiples, tout spécialement dans les domaines de l'informatique ubiquitaire et de la réalité augmenté [Salber *et al.*, 1999; Greenberg et Fitchett, 2001; Ballagas *et al.*, 2003]. Ces approches tendent à proposer un accès unifié et flexible à beaucoup de périphériques d'entrée, mais elles reposent toutefois sur des modèles d'interaction minimalistes. La bibliothèque de « **Phidgets/WidgetTaps** » [Greenberg et Fitchett, 2001] permet de relier des « widgets » à leurs représentations physiques (voir figure 9.4 page ci-contre). Le « *Patch-Panel* » de la boîte à outils **iStuff** [Ballagas *et al.*, 2003] permet la même connexion entre périphérique et fonction. Le principe de « connexion dynamique » de ces outils est très similaire à MAGGLITE.

Mais dans ces approches, seule la connexion des périphériques est explicite, reposant sur un configurateur graphique. Pourtant, les interactions mises en œuvre dans la majorité des interfaces utilisateurs avancées ne peuvent pas se réduire à de simples mises en correspondance entre des boutons et des commandes. De fait, il est nécessaire d'utiliser des techniques d'interaction avancées pour pouvoir adapter efficacement n'importe quel périphérique à n'importe quelle tâche, ce que ces systèmes ne permettent que par implémentation dans un paradigme de programmation standard.

[(1)]Cette approche est en partie due au fait que nous avions déjà des moyens logiciels avancés pour la gestion des entrées : ICON.

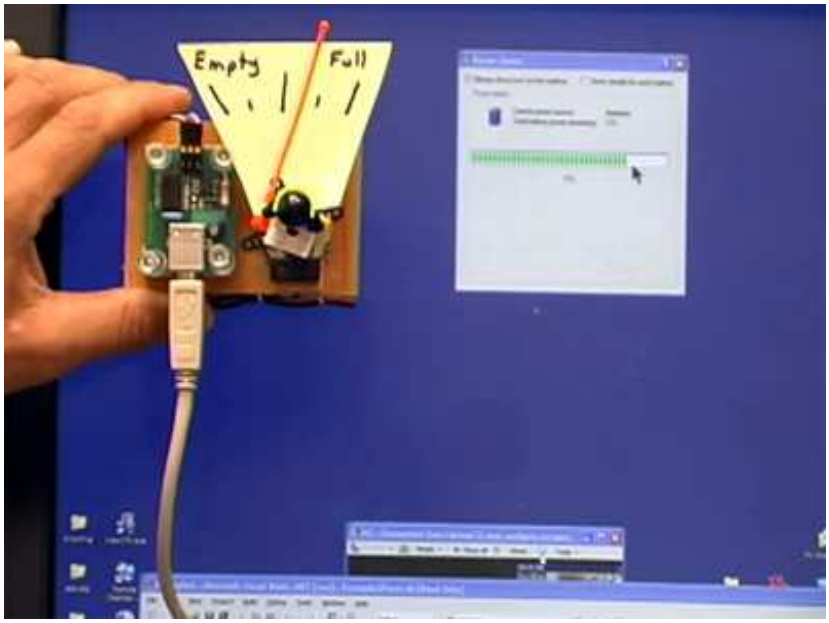


FIGURE 9.4 – Les *Phidgets* [Greenberg et Fitchett, 2001] sont des composants physiques pouvant être mis en relation avec des composants logiques de l'interface.

MAGGLITE va plus loin que les outils existants en offrant plus de pouvoir d'expression et de flexibilité de ce point de vue. Notre approche montre comment introduire plus de flexibilité et de contrôle en décrivant les entrées multiples avec les graphes d'interaction connectés à des graphes de scène à un niveau de granularité plus fin. De plus, le principe des *interactions génériques* permet d'adapter l'interaction aux entrées, mais aussi de proposer plusieurs styles d'interaction dans un même contexte.

9.4 Constructeurs d'interfaces

Les constructeurs d'interfaces courants, tels que **Visual Basic**, ont deux principales lacunes : premièrement, ils sont fortement limités à la construction d'interfaces WIMP. Deuxièmement, alors que l'apparence de l'interface est construite graphiquement, les comportements (incluant les interactions) doivent être programmés, parfois avec des techniques et un code lourds. Des outils plus avancés pour la construction et le prototypage d'interfaces ont étés proposés, tout particulièrement avec la boîte à outils *Garnet/Amulet* [Myers, 1990]. La plupart, comme **Lapidary** [Vander Zanden et Myers, 1995], sont des constructeurs graphiques d'interfaces qui utilisent la programmation par l'exemple pour spécifier les comportements. Bien que puissant et efficace, ce paradigme soulève beaucoup de problèmes qui n'ont pas encore été résolus. C'est surtout le cas lors de la spécification de comportements non triviaux, nécessitant alors de restreindre le paradigme à un domaine d'application précis.

SILK [Landay et Myers, 1995; Landay, 1996] est un outil de prototypage d'interfaces basé sur une interaction gestuelle (voir figure 9.5 page suivante). C'est un environnement de conception d'interfaces complet, avec beaucoup de fonctionnalités telles que l'édition des dessins d'interfaces, l'historique du projet, des outils d'annotation et des scénarios d'explication des interaction/comportements.

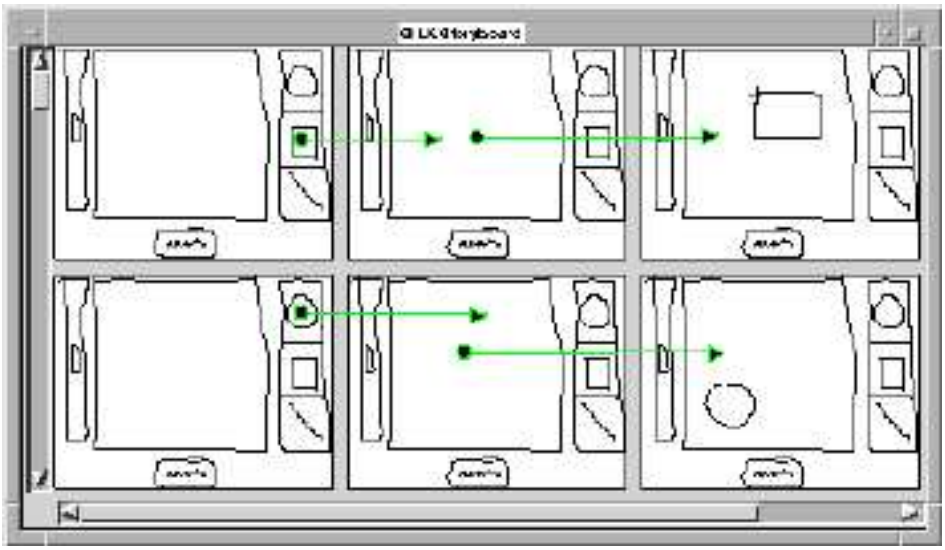


FIGURE 9.5 – *SILK* (Sketching Interfaces Like Krazy) [Landay et Myers, 1995].

Par contre, les composants des interfaces construites sont limités aux composants standards de *Visual Basic* ou *CommonLisp* (pour *Garnet*). De plus, les interfaces et interactions ne sont pas fonctionnelles lors de la conception et ne peuvent donc pas être complètement testées et raffinées dynamiquement lors de cette étape.

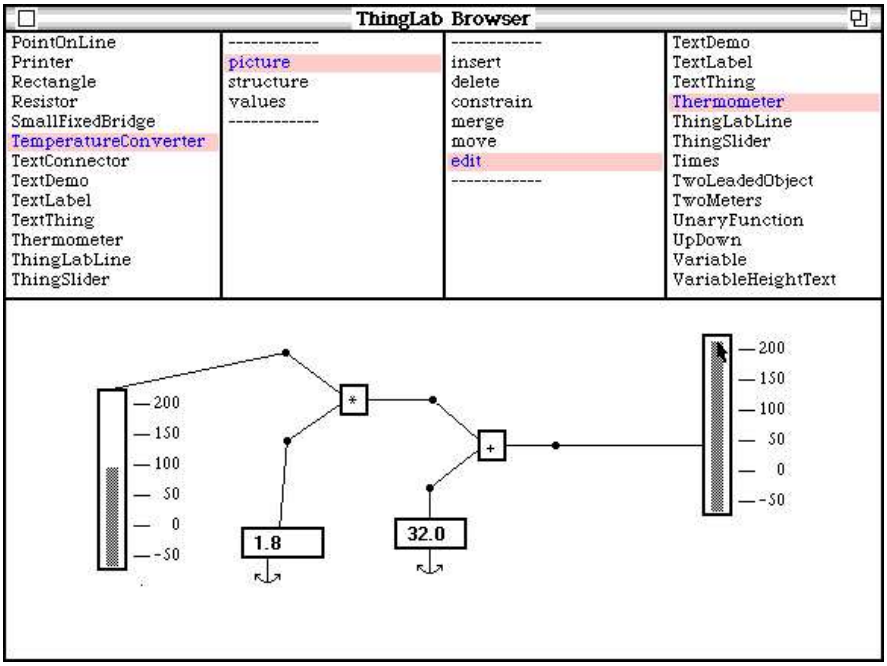
MIB, le constructeur d'interfaces que nous proposons avec MAGGLITE, permet par contre de prototyper rapidement des interfaces graphiques tirant parti de toutes les interactions avancées de la librairie (dessin, gestes, reconnaissance vocale, etc.), et même d'en prototyper de nouvelles. Il n'est pas limité à la construction d'interfaces WIMP et permet d'utiliser l'interface pendant sa création.

9.5 Éditeurs graphiques de comportements

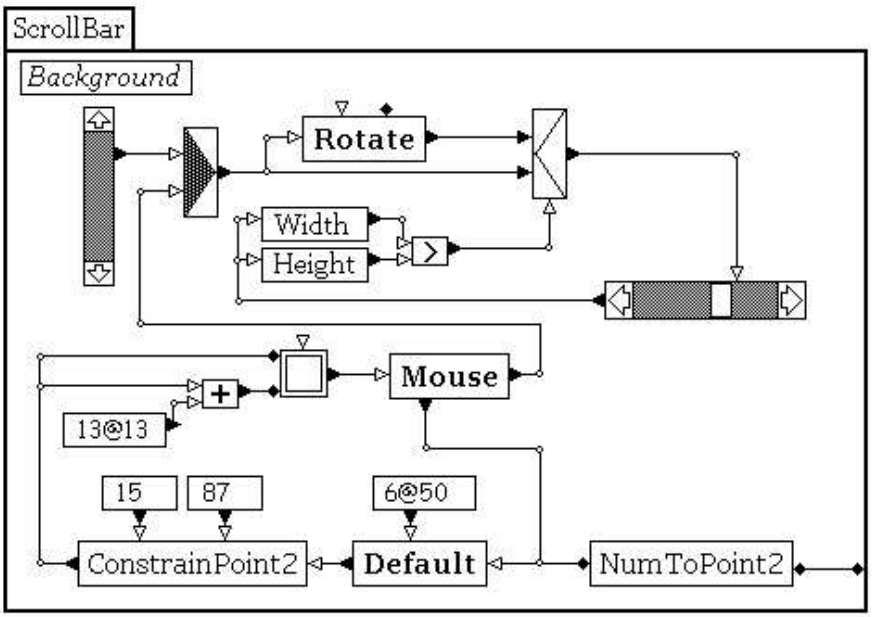
Des éditeurs basés sur la résolution de contraintes, tels que **Thinglab** [Borning, 1979] (pour des simulations, voir figure 9.6(a) page ci-contre) ou **Fabrik** [Ingalls *et al.*, 1988] (voir figure 9.6(b)) permettent de spécifier graphiquement certains comportements des applications interactives, en particulier le comportement des agencements spatiaux automatiques. Des approches basées sur le flot de contrôle, telles que **ICO/PetShop** [Bastide *et al.*, 2002], utilisent les *Réseaux de Petri* ou les *Diagrammes État-Transition* pour décrire les parties orientées contrôle et hautement modales des applications interactives.

Des éditeurs graphiques de flot de données sont utilisés dans divers domaines. Par exemple, **Max/MSP** [Puckette, 1991] est un environnement de programmation graphique très utilisé pour les applications dans le domaine de la musique. Toutefois, bien que très adapté au traitement en temps réel des données MIDI, de l'audio ou des images avec des widgets simples et somme toute standards, il n'a pas été conçu pour décrire des interactions avancées.

L'utilisation d'éditeurs de flot de donnés pour spécifier des interactions a rarement été exploité en dehors de la conception d'environnements et d'animations 3D. **Virtools Dev** [Virtools SA, 2001] est



(a) *Thinglab*. une vue de l'éditeur.



(b) *Fabrik*. Description d'un composant interactif.

FIGURE 9.6 – *Thinglab* et *Fabrik*.

un exemple de l'utilisation d'un éditeur de flot de donnés pour spécifier interactivement des techniques d'interaction en entrée. Le système **VRED** [Jacob *et al.*, 1999] utilise à la fois un éditeur de flot de contrôle (diagrammes état-transition) et un éditeur de flot de données pour décrire les aspects discrets et continus de l'interaction 3D.
Cette approche de flot de données a prouvé son potentiel prometteur pour décrire des techniques d'interaction utilisant plusieurs périphériques d'entrée, mais à notre connaissance, en écartant bien sur MAGGLITE et ICON, la seule tentative d'utilisation pour décrire l'interaction 2D est **Whizz'Ed** [Esteban *et al.*, 1995]. Cette notation a permis de spécifier des animations et certaines techniques bimanuelles, sans toutefois aller aussi loin que le permet MAGGLITE en matière de périphériques d'entrée, de techniques d'interaction et d'objets graphiques.

9.6 Techniques d'interaction avancées

En plus des techniques non-standard basées sur les traces (dessin, gestes et Marking menus), la parole, les outils transparents ou les transformations (Fisheyes) que nous avons déjà évoquées et dont l'usage tend de nos jours à s'intensifier, les techniques proposées dans MAGGLITE présentent aussi des similitudes avec des paradigmes plus récents ou moins *visibles*.

9.6.1 Local Tools

FIGURE 9.7 – *KidPad* [Druin *et al.*, 1997] est une application pour enfants basée sur les *Local Tools*.

Avec les **Local Tools** [Bederson *et al.*, 1996], Ben BEDERSON décrit une alternative aux palettes d'outils dans laquelle « chaque outil peut être sélectionné (il remplace alors le curseur principal), utilisé et reposé n'importe où sur l'espace de travail ». L'application *KidPad* [Druin *et al.*, 1997] (voir figure 9.7 page précédente) utilise ces outils ainsi que la bibliothèque **MID** [Hourcade et Bederson, 1999] pour gérer plusieurs souris. Ainsi, plusieurs outils sont utilisables en même temps. Ces propriétés sont très proches des *Outils Internes* de MAGGLITE. Il n'est toutefois pas possible d'associer librement des dispositifs d'entrée aux outils (ce qui permet de rendre leur sélection implicite), ni d'utiliser de dispositifs plus avancés tels que des tablettes graphiques ou des dispositifs isométriques 3D (permettant de définir des outils plus complets).

9.6.2 Interactions de franchissement et de proximité

Le paradigme des interactions de franchissements a récemment été introduit pour améliorer l'interaction dans les interfaces standards basées sur la seule notion de pointage [Accot, 2001; Accot et Zhai, 2002]. Utilisant des événements produit lors du franchissement des bords de l'objet d'intérêt par le pointeur, de telles interactions peuvent améliorer ou même remplacer les techniques de pointage. **CrossY** (voir figure 9.8), par exemple, est une application expérimentale de dessin dont toutes les interactions (sélection d'outils, paramètres, etc.) reposent sur ce modèle [Apitz et Guimbretière, 2004].

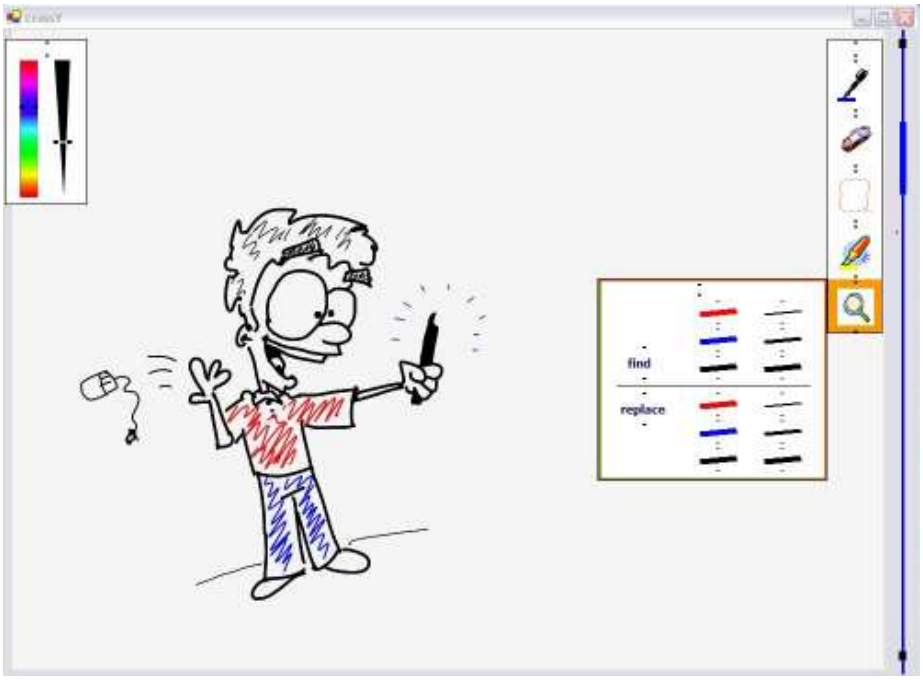


FIGURE 9.8 – *CrossY* est une application dont toutes les interactions reposent sur des franchissements.

La technique du **Fold'n Drop** est un autre exemple d'application des interactions de franchissement pour faciliter l'opération de Glisser-Déposer lorsque des fenêtre se chevauchent [Dragicevic, 2004a]. Lors d'un Glisser-Déposer, le franchissement des bords d'une fenêtre permet de la *plier* pour accéder simplement à celles du dessous (voir figure 9.9 page suivante).

FIGURE 9.9 – La technique du *Fold'n Drop* [Dragicevic, 2004a] combine des interactions de franchissement et la métaphore du pliage de papier pour faciliter le Glisser-Déposer dans des fenêtres qui se chevauchent.

Seulement, proposer de telles interactions dans les boîtes à outils traditionnelles nécessite encore une fois d'importants efforts de programmation et ne garantit pas la réutilisabilité, ni la diffusion de la technique proposée. Le principe de *sélection par proximité* de MAGGLITE, associé à la sélection standard est un moyen simple et efficace de détecter les franchissements des bords des objets. Dès lors, la réalisation de telles interactions est grandement simplifiée, assurant de plus leur réutilisabilité dans toutes les applications développées avec la boîte à outils. Par exemple, le *pliage* dans MAGGLITE est une capacité graphique des objets. Il peut être contrôlé directement avec un périphérique d'entrée (calques de SVALABARD), mais aussi à l'aide d'un dispositif d'interaction basé sur le franchissement de l'objet pliable (comme dans le cas du *Fold'n Drop*).

Mais nous pouvons aussi dresser un parallèle entre les *interactions de franchissement* et la notion d'*interactions de proximité* qu'introduit notre technique originale des *Responsive Handles*. En effet, l'apparition des poignées est déclenchée par le franchissement de la proximité de l'objet par le pointeur, permettant d'ajouter un niveau d'interaction *autour* de celui-ci. Cette approche permet d'améliorer l'interaction avec les objets, sans pour autant interférer avec celles réalisables à l'intérieur comme pourraient le faire de simples franchissement de ses bords. Nous pensons que cette approche est aussi une voie intéressante à développer et à explorer pour améliorer les interfaces de pointage.

9.7 Conclusion : limites et perspectives

Finalement, nous dressons un bilan synthétique des apports de MAGGLITE sur trois points :

- L'implémentation de la boîte à outil repose sur un **modèle logiciel flexible**, **modulaire** et suffisamment **générique** pour prendre en compte différents modèles d'interaction. Cela lui confère des capacité inédites pour l'intégration de nombreuses techniques d'interaction non-standard dans un même environnement.
- La librairie des composants de la boîte à outils fournit un support natif et complet pour de **nombreux dispositifs d'entrée**, associé à un ensemble extensible de **techniques d'interaction avancées** *prêtes à l'emploi*.
- Outre les abstractions et mécanismes pour faciliter le développement d'applications, MAGGLITE propose des **outils interactifs** pour le prototypage et la conception des interfaces et des interactions, ainsi que pour leur configuration dynamique.

Ces apports font de MAGGLITE un outil adapté au prototypage et à la conception d'interfaces innovantes et d'interactions non-standard. Notre approche ne se veut pas exhaustive en terme de problèmes résolus et d'intégration de techniques d'interaction. Par contre, son ouverture et sa flexibilité permettra de juger de sa pertinence et de ses limites.

9.7.1 Limites

Nous proposons d'identifier les limites actuelles de notre approche sur trois plans importants : les notions de *concurrence et de contrôle*, les *limites de la programmation visuelle* et l'*évolution vers un contexte industriel* de notre approche.

Concurrence et contrôle

Le modèle des graphes combinés sur lequel repose MAGGLITE nous a permis de remplir notre objectif d'intégration de divers paradigmes d'interaction dans une même boîte à outils de manière flexible et modulaire. Pourtant, la gestion concurrente de diverses techniques et dispositifs d'entrée pose des problèmes au niveau du **multiplexage** (contrôle de la fusion ou de la repartition des flots de données) et des **modes d'interaction**. Bien que plus présente dans les paradigmes traditionnels (WIMP), cette notion de **contrôle** intervient aussi dès lors que les interfaces avancées s'orientent vers une intégration plus poussée de paradigmes différents. Dans ce cadre des interfaces post-WIMP, cet écueil peut être contourné par une approche totalement instrumentée, où un périphérique va contrôler un instrument unique. Mais bien que techniquement possible avec notre approche, cette solution peut rapidement devenir une faiblesse du point de vue de l'utilisabilité en multipliant les périphériques d'entrée. SVALABARD, par exemple, a à notre avis atteint cette limite et nous avons constaté que le paradigme de flot de données ne se prête pas (ou peu) à la description de configurations de multiplexage dans des cas complexes.

Pierre DRAGICEVIC a déjà évoqué ce problème dans sa thèse [Dragicevic, 2004b] et a proposé des pistes pour une hybridation de son modèle en flot de données avec des approches orientées contrôle :

1. une vision **macroscopique** qui, en conjonction avec un formalisme basé sur un système de transition, nécessiterait des redirections dynamiques des flux de données dans les configuration d'entrées ;
2. une vision **microscopique**, qui consisterait à gérer le contrôle par des machines à états encapsulées à l'intérieur des dispositifs.

Bien que plus adaptée pour l'intégration des deux modèles, la solution microscopique nécessite l'emploi d'un langage de programmation prenant en compte les transitions d'état. La solution macroscopique permettrait l'emploi d'un formalisme visuel pour la description du contrôle, plus en accord avec nos propositions de prototypage visuel dans MAGGLITE. Toutefois, la reconnexion dynamique des graphes d'interactions pose alors un problème de visibilité et de compréhension.

Notre modèle des graphes combinés pourrait apporter une nouvelle solution à un niveau *intermédiaire* : un système de transition configurable graphiquement tel que **Pet Shop** [Bastide *et al.*, 2002] contrôlerait la connexion dynamique des IAPs aux composants du graphe de scène et des objets d'intérêt de l'application, sans briser la cohérence des graphes d'interaction (deconnexions et reconnexions des flux d'interaction).

Les limites de la programmation visuelle

La flexibilité et l'association dynamique des graphes de notre modèle a montré sa pertinence pour le prototypage visuel de la partie graphique (le graphe de scène) ou des interactions (le graphe d'interaction) avec notre constructeur interactif d'interfaces. Toutefois, cette approche présente encore des limites évidentes sur ces deux points.

Actuellement, les différents *niveaux* de programmation d'applications interactives avec notre boîte à outils sont :

1. Un **paradigme de programmation visuelle** pour décrire la partie graphique (dessin) et les interactions (ICON).
2. Des outils de **programmation traditionnelle** pour construire des applications et leur partie *métier*.

Au niveau des graphes d'interaction, la modularité et la décomposition fine en de nombreux dispositifs montre ses limites pour la réalisation de techniques complexes. Ce n'est pas le modèle en lui-même qui est en cause, mais surtout la complexité visuelle et la compréhension de configurations d'entrées complexe. Il est alors nécessaire de se poser la question de la réalisation de nouveaux dispositifs pouvant « généraliser » et « factoriser » la technique à décrire. Bien que cette approche nécessite un retour à la programmation traditionnelle, elle demeure selon nous indispensable. Nous n'y voyons toutefois pas une régression de notre approche, le principe des dispositifs IAPs encourageant et facilitant d'ailleurs cette démarche dans un cadre de réutilisabilité et de généricité. Il est toutefois important de considérer le problème du choix du niveau de granularité lors de la création de nouveaux dispositifs et de proposer des principes de conception pour aider les développeurs dans cette optique.

Outre la nécessité d'améliorer les outils qu'elle propose pour permettre la création de graphismes avancés (gradients de couleurs, effets, ombres, etc.), la création interactive de composants graphiques pose aussi le problème de la définition d'*objets du domaine* ou *composants métier*. Actuellement, les objets graphiques de la boîte à outils sont essentiellement des vues, n'offrant pas un modèle de description et de communication de haut-niveau. La partie métier nécessite donc d'être programmée au niveau de l'application ou dans de nouvelles classes d'objets graphiques (composites, par exemple). Cette notion n'était pas notre préoccupation première, et les composants actuels suffisent à valider notre modèle, en particulier du point de vue de la liaison graphique/interactions.

Il serait maintenant nécessaire d'étudier l'intégration et la réalisation de tels concepts de haut niveau dans notre modèle sans en briser la cohérence et les avantages pour le prototypage interactif. L'extension des *Manipulateurs* pour le contrôle des objets du domaine est une première piste pour externaliser leur communication dans le même formalisme et le même langage de description que le comportement des objets graphiques. Mais nous pensons aussi à une représentation graphique interactive du graphe de scène qui, associée à des modules *métier* dépendant du contexte de développement permettrait de les mettre simplement en relation.

Évolution vers un contexte industriel

Il convient toutefois de garder à l'esprit que MAGGLITE est un prototype de recherche, ayant prouvé la faisabilité et les avantages de notre nouveau modèle d'implémentation : adéquations à des modèles d'interaction, intégration de techniques d'interaction variées et d'entrées enrichies, graphismes avancés, et adéquation à des outils de prototypage visuels et interactifs.

Il nous parait alors essentiel d'étudier les points nécessaires à son évolution vers un système plus complet de prototypage d'IHM. En effet, de telles approches de séparation des descriptions commencent à être utilisées dans un contexte industriel pour le développement d'applications à grande échelle. La société INTUILAB, par exemple, a développé la boîte à outils **IntuiKit** dans cette optique, prouvant ainsi l'efficacité de cette approche pour la prise en compte et l'intégration des divers acteurs de la conception d'interfaces tout au long d'un projet [Chatty *et al.*, 2004]. Alors que les concepteurs graphique utilisent leurs logiciels habituels pour produire les graphismes de l'interface, les développeurs construisent l'application et ses interaction de manière indépendante. La liaison entre les modules s'opère par une architecture proche de nos graphes combinés.

Notre modèle offre en plus un aspect dynamique permettant à notre avis d'aller plus loin dans l'unification dans un même environnement des différentes *compétences* impliquées dans la conception d'interfaces et le prototypage d'interactions. L'introduction d'une approche orientée contrôle et d'une représentation graphique d'objets du domaine permettrait la construction d'un **Atelier de Génie Logiciel** pour le prototypage en IHM reposant essentiellement sur des techniques interactives, le rendant alors accessible aux graphistes, ergonomes et programmeurs. Mais cette proposition implique d'étudier la qualité du modèle en terme de robustesse et de performances, ainsi que d'améliorer les outils actuels (outils de dessin du constructeur d'interfaces, documentation, etc.)

9.7.2 Perspectives et avenir des boîtes à outils

MAGGLITE nous semble être un bon support pour l'enseignement de l'IHM et de la conception d'interfaces. En effet, le principe de construction et de connexion graphique des interactions et des composants de l'interface permet de réaliser des exemples simples, mettant plus l'accent sur les modèles sous-jacents que sur leur implémentation. Si l'on se réfère à notre expérience sur ce sujet, beaucoup d'étudiants comprennent facilement le modèle *MVC* mais peinent à le réaliser de manière claire et intuitive avec des boîtes à outils telles que *Swing*. L'utilisation de MAGGLITE permettrait d'en proposer une première approche visuelle afin de comprendre la place et le comportement de chaque agent. Dans cette optique, nous projetons d'étudier un ensemble de problèmes significatifs et de réaliser leurs solutions avec MAGGLITE dans un but pédagogique.

Ensuite, la modularité et la flexibilité de notre approche en fait une plateforme adaptée au prototypage d'interactions. Nous souhaitons d'ailleurs en tirer parti pour nos recherches futures dans ce domaine. Mais nous envisageons aussi d'améliorer son implémentation dans un souci de robustesse, puis d'enrichir la documentation et les exemples fournis afin de permettre sa diffusion à une plus grande échelle dans les domaines de la recherche et de l'enseignement. C'est peut-être une perspective ambitieuse, mais MAGGLITE pourrait alors permettre la réalisation, la diffusion et l'utilisation de techniques innovantes qui restent trop souvent à l'état de prototypes faute de possibilité d'intégration dans des environnements standards.

Avenir des boîtes à outils et des modèles

Nos travaux nous ont menés à constater un paradoxe soulevant de nombreuses questions quant à l'avenir des boîtes à outils d'interaction et des modèles sur lesquels elles se basent (architectures logicielles) ou qu'elles permettent de réaliser (modèles d'interaction). Alors que le rapport au monde des systèmes informatiques évolue dans une voie toujours plus communiquante et ouverte, les solutions

techniques qui les régissent à grande échelle restent toujours aussi rigides et fermées.

Car si l'architecture des ordinateurs n'a pas vraiment évoluée depuis ses débuts, comme le souligne Jean-Daniel FEKETE dans [Fekete, 2005], la manière dont nous les utilisons et les projections que nous faisons de leur utilisation a par contre beaucoup changée. Dès lors, si nous devions donner une seule cause à la relative pauvreté des systèmes actuels en termes d'interaction, nous l'expliquerions par une préoccupation de **standardisation** plutôt que **d'unification**. Et c'est justement sur un souci d'unification que se sont penchés nos travaux : une architecture tendant à unifier l'utilisation de diverses techniques d'interaction et par conséquent de différents modèles d'architectures logicielles et modèles d'interaction.

L'on peut alors se poser la question de savoir qui des boîtes à outils ou des systèmes informatiques doit évoluer pour mieux correspondre à l'autre et pour mieux prendre en compte les besoins. Même si les premières ont parcourus plus de chemin que les seconds vers ce point de rencontre, il s'avère toutefois que des plateformes matérielles dédiées à une seule utilisation existent et prouvent leur efficacité (des PC *Media Center* aux consoles de jeux), bien que ne changeant pas d'architecture physique.

Mais nous pensons surtout que l'avenir des systèmes interactifs passe par deux points cruciaux :

1. pousser encore plus loin notre approche d'unification afin d'atteindre un niveau encore plus riche de description de l'interaction, intégrant même *l'utilisateur* dans le modèle.
2. plus que dans des boîtes à outils, intégrer des architectures et des outils tels que ceux que nous proposons au niveau des systèmes d'exploitation afin de rendre l'interaction plus explicite, visible et adaptable.

Si la première de ces propositions est plus qu'envisageable au vu du chemin actuellement parcouru par la recherche en IHM, la seconde est plus utopique, mais à notre avis nécessaire pour un passage de la *standardisation* à *l'unification*.

Conclusion générale

Concepteurs, ergonomes et informaticiens s'accordent sur le fait que les systèmes informatiques peuvent apporter de nouvelles voies à la démarche créative. L'accès et la manipulation des connaissances, indispensables à l'exploration et la génération de solutions à un problème, peuvent par exemple être améliorés. Des simulations avancées et des visualisations adaptées peuvent être un atout pour l'évaluation de ces solutions. Bien que de nombreux travaux de recherche proposent régulièrement des avancées significatives dans ces différents domaines, leur utilisation dans une démarche créative reste anecdotique.

Dans le cadre de la conception architecturale, par exemple, les systèmes de CAO n'interviennent que dans une optique de simulation et de support aux phases avancées du projet (vérification et construction), lorsque les principaux concepts en ont déjà été *dessinés*. Pourtant, les différentes vues qu'ils offrent sur l'objet de la conception ainsi que les outils de simulation qu'ils renferment pourraient être de nouveaux atouts dans les premières phases de la conception, délivrant alors une autre vision, plus globale, du ou des problèmes à résoudre. Mais, dans cette phase de conception créative, là où le concepteur va manipuler des concepts et des idées pour raffiner ses solutions, le logiciel va attendre de lui des données précises, des entités géométriques ou des opérations définies. Considérer de telles données dans ces phases exploratoires de la conception va de fait éliminer de nombreuses possibilités de solution pour le concepteur. Mais, plus qu'une simple mésentente entre lui et la machine, c'est toute la démarche de conception qui est alors faussée et limitée, faute d'un support figuratif et interactif assez souple et effacé pour supporter la création.

C'est pourquoi, avant d'explorer les valeurs ajoutées possibles des systèmes informatiques aux prémices de la démarche créative, nous avons orienté nos travaux vers leur *intégration* dans cette démarche. Nous sommes conscients que ces deux notions (apports de l'outil et son utilisation) sont intimement liées et nous avons donc tenté de proposer une solution suffisamment ouverte et flexible pour son évolution, mais aussi pour son utilisation.

Intégrer sans dénaturer

De l'état de l'art que nous avons proposé, ressortent alors deux points primordiaux : l'importance et la qualité du dessin dans le processus de conception créative, et son utilisation actuelle dans les systèmes informatiques.

Le dessin, tout d'abord, tient lieu d'outil figuratif privilégié du concepteur. Mais en ce sens, nous rejetons le qualificatif trop communément employé de *photographie de la pensée*, en particulier dans le domaine de l'informatique. Dans les étapes créatives de la conception, réduire le dessin à une vue externe et précise d'une *image mentale* à un instant donné revient à le réduire à une simple *technique*.

Or, les travaux menés par des chercheurs tels que Eugene FERGUSSON et Vinod GOËL montrent que le dessin est un *prolongement de la pensée* et même un *compagnon de la conception*. Il n'est pas figé, il évolue, il régresse, il suggère à son créateur et établit un dialogue. Dans ce sens, le dessin est beaucoup plus qu'une simple représentation graphique de l'objet en cours de création.

Dès lors, un système informatique tirant parti du dessin pour fournir un support créatif doit être en mesure de supporter ce dialogue, tout en sachant l'interpréter sans le briser. Et c'est là le deuxième constat que nous pouvons faire : il n'existe pas, à notre connaissance, de système informatique de dessin interprété suffisamment *libre* pour inspirer le même rapport que celui établi entre un concepteur et son carnet de croquis. Bien sur, de nombreuses avancées ont été proposées, les travaux récents que nous avons présentés pour la modélisation 3D sont des étapes importantes, mais ils considèrent toujours le dessin comme une technique, et limitent son emploi à ce dont le système à besoin : tracer les bons traits au bon endroit et de manière continue (au mieux *à main levée*, au pire *de manière guidée*). Il nous semble alors qu'une telle utilisation du dessin ne peut pas être aussi stimulante pour la créativité qu'un papier et un crayon si elle ne propose pas au moins des possibilités identiques.

Ainsi, le problème que nous avons abordé dans cette thèse peut être étendu à la question plus vaste des possibilités d'association entre conception créative et informatique : est-il possible de réaliser des outils de dessin suffisamment *libres* pour supporter la créativité, tout en offrant des valeurs ajoutées au *numérique* par rapport au *papier* ?

Limites et ambitions

Nos travaux ont d'abord montré la nécessité de restreindre la portée d'un système créatif à un domaine précis, afin d'identifier en amont et de s'adapter en aval aux spécificités et aux habitudes des utilisateurs potentiels. La méthode que nous avons suivi pour cela a consisté en une étude de l'état de l'art du domaine (l'architecture) et de la tâche (le dessin) afin d'établir des lignes directrices mêlant contexte global de la créativité et spécificités de l'architecture.

Façonner l'outil

Nous avons alors été en mesure de proposer une approche globale, basée à la fois sur l'interprétation de dessins à main levée, mais aussi et surtout sur une démarche libre dans un environnement de conception instrumenté. La métaphore de la *table à dessin virtuelle* naît de l'assemblage de paradigmes d'interaction avancés issus des habitudes des concepteurs (dessins et gestes, feuilles de papier, calques) et de leur association à des périphériques d'entrée exploités comme des instruments. Pour ce qui est du dessin, nous avons essayé de ne pas le considérer comme une simple technique, mais aussi comme un partenaire, en proposant des traitements permettant de relever les contraintes imposées par les approches similaires (détection des phases du dessin et fusion des segments). Nous sommes toutefois conscients des limites actuelles de notre réalisation et de la nécessité de l'évaluer et de l'améliorer. C'est pour cela que nous l'avons conçue comme un système *ouvert* du point de vue de son évolution, mais aussi de son utilisation. C'est selon nous une première voie possible à une meilleure intégration dans la démarche créative : permettre de *façonner l'outil*.

Langages d'interaction

Mais nous avons aussi fait la preuve que poursuivre de telles ambitions n'était pas sans peine du point de vue de leur réalisation. À l'heure actuelle, sortir des stéréotypes et des conventions dans les interactions et les interfaces homme-machine nécessite de résoudre ou de contourner des problèmes essentiellement d'ordre *technique*. C'est pourquoi nous avons essayé d'aborder aussi cette question d'un point de vue global, en proposant un modèle d'architecture logicielle suffisamment *flexible* et *modulaire* pour permettre de faire cohabiter et de démocratiser des techniques actuellement marginales, bien qu'efficaces.

Certes, ce modèle nécessite encore d'être amélioré, contesté et éprouvé avant de dévoiler si il peut être une solution *unifiée* aux nombreux problèmes que pose la conception d'interactions et d'interfaces. Quoiqu'il en soit, nous sommes persuadé que les outils et paradigmes qu'il supporte (programmation visuelle et techniques d'interaction avancées) sont l'une des clés d'un meilleur *dialogue* avec nos ordinateurs. Ou plutôt devrait-on dire de la *construction* d'un meilleur dialogue. Car si la recherche en IHM a jusqu'à présent proposé un large dictionnaire de mots (techniques d'interaction), de règles de conjugaison (règles d'utilisabilité) et de style (modèles d'interaction), il nous semble maintenant nécessaire de pouvoir les utiliser, les expérimenter, les faire évoluer, pour construire de nouveaux *langages d'interaction* aussi divers et variés que les utilisateurs censés les pratiquer.

Et si finalement la réponse à la question du rapport entre conception créative et informatique tenait en partie dans la créativité même que permettent les outils de conception d'interfaces ? Explorer, proposer des solutions, les évaluer et remettre en question les conventions et les acquis sont les termes que nous avons utilisés pour définir les activités de la créativité. Ce sont ces activités que doivent aussi permettre les futurs outils pour la conception d'applications, afin que les limites de ces applications ne soient pas celles de leurs programmeurs ou des concepteurs de leurs interfaces, mais celles définies par leurs utilisateurs.

« A display connected to a digital computer gives us a chance to gain familiarity with concepts not realizable in the physical world. It is a looking glass into a mathematical wonderland. » Ivan E. Sutherland, [Sutherland, 1965].

Bibliographie

[Abvent, 2005] cité page 34
Abvent. Page web de Archicad™, 2005. http://www.abvent.com/softwares/archicad/ .

[Accot et Zhai, 2002] cité page 221
Johnny Accot et Shumin Zhai. More than dotting the i's — foundations for crossing-based interfaces. Dans *CHI '02 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 73–80. Minneapolis, Minnesota, USA, 2002. ACM Press, New York, NY, USA.

[Accot, 2001] cité page 221
Johnny Accot. *Les Tâches Trajectorielles en Interaction Homme-Machine – Cas des tâches de navigation*. Thèse de Doctorat, Université de Toulouse 1, Toulouse, France, Janvier 2001.

[Adobe, Inc, 2004] cité page 44
Adobe, Inc. Page web de canoma., 2004. http://www.canoma.com .

[Alias, 2005a] cité page 138
Alias. Page web de alias sketchbook pro., 2005. http://www.sketchbookpro.com .

[Alias, 2005b] cité pp. 34, 39
Alias. Page web de alias® (alias wavefront)., 2005. http://www.alias.com .

[Allan Christian Long, Jr et Landay, 1999] cité dans les annexes
Allan Christian Long, Jr et James A. Landay. Implications for a gesture design tool. Dans *Proceedings of the ACM CHI 1999 Conference on Human Factors in Computing Systems*, pages 40–47. Pittsburgh, Pennsylvania, USA, 15–20 Mai 1999. ACM/Press, New York, NY, USA.

[Allan Christian Long, Jr, 2001] cité dans les annexes
Allan Christian Long, Jr. *Quill : A Gesture Design Tool for Pen-Based User Interface*. Thèse de Doctorat, Univeristy of Berkeley, 2001. Chair : James A. Landay.

[Apitz et Guimbretière, 2004] cité page 221
Georg Apitz et François Guimbretière. CrossY : a crossing-based drawing application. Dans *UIST '04 : Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 3–12. Santa Fe, NM, USA, 2004. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Arc Second, Inc, 2004] cité page 138
Arc Second, Inc. Page web de pocketcad pro., 2004. http://www.pocketcad.com/ .

[Balakrishnan *et al.*, 1999] cité page 50
Ravin Balakrishnan, George Fitzmaurice, Gordon Kurtenbach, et William Buxton. Digital tape drawing. Dans *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 161–170. Asheville, NC, USA, 7–10 Novembre 1999. ACM Press, New York, NY, USA.

[Ballagas *et al.*, 2003] cité pp. 150, 216
Rafael Ballagas, Meredith Ringel, Maureen Stone, et Jan Borchers. istuff : A physical user interface toolkit for ubiquitous computing environments. Dans *Proceedings of the ACM CHI 2003 Conference on Human Factors in Computing Systems*, pages 537–544. Ft. Lauderdale, FL, USA, Avril 2003. New York, NY, USA.

[Bastide *et al.*, 2002] cité pp. 218, 223
Rémi Bastide, David Navarre, et Philippe Palanque. A model-based tool for interactive prototyping of highly interactive applications. Dans *CHI'02 extended abstracts*, pages 516–517. Minneapolis, MN, USA, 2002. ACM/SIGGCHI, New York, NY, USA.

[Bastien et Scapin, 1993] cité page 88
Christian Bastien et Dominique Scapin. Critères ergonomiques pour l'évaluation des interfaces utilisateurs. Rapport Technique RT nř156, INRIA, Juin 1993.

[Baudel, 1994] cité pp. 120, 137
Thomas Baudel. A Mark-Based Interaction Paradigm for Free-Hand Drawing. Dans *Proceedings of the 7th ACM Symposium on User Interface Software and Technology (UIST '94)*. Marina Del Rey, USA, 1994. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Baudel, 1995] cité pp. 131, 137
Thomas Baudel. *Morphologie de l'Interaction Humain-Ordinateur : Étude de Modèles d'Interaction Gestuelle*. Thèse de doctorat, Université de Paris-Sud, U.F.R. scientifique d'Orsay, Décembre 1995.

[Beaudouin-Lafon et Lassen, 2000] cité pp. 94, 150, 158, 214, 215
Michel Beaudouin-Lafon et Henry Michael Lassen. The architecture and implementation of CPN2000, a post-WIMP graphical application. Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 181–190. San Diego, CA, USA, 5–8 Novembre 2000. ACM Press, New York, NY, USA.

[Beaudouin-Lafon, 1997] cité pp. 36, 130, 211
Michel Beaudouin-Lafon. Interaction instrumentale : de la manipulation directe à la réalité augmentée. Dans *Actes des Neuvièmes Journées sur l'Interaction Homme-Machine, IHM'97*. Poitiers, France, Septembre 1997. Cépaduès-Editions.

[Beaudouin-Lafon, 2000] cité pp. 84, 91, 130, 150, 211, 212, 216
Michel Beaudouin-Lafon. Instrumental interaction : an interaction model for designing post-WIMP user interfaces. Dans *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI-00)*, pages 446–453. The Hague, The Netherlands, 1–6 Avril 2000. ACM Press, New York, NY, USA.

[Beaudouin-Lafon, 2001] cité pp. 96, 129, 192
Michel Beaudouin-Lafon. Novel Interaction Techniques for Overlapping Windows. Dans *Proceedings of the 14th ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 153–154. Orlando, FL, USA, Novembre 2001. ACM Press, New York, NY, USA.

[Beaudouin-Lafon, 2004] cité page 150
Michel Beaudouin-Lafon. Designing interaction, not interfaces. Dans *AVI '04 : Proceedings of the working conference on Advanced visual interfaces*, pages 15–22. Gallipoli, Italy, 25–28 Mai 2004. ACM Press, New York, NY, USA.

[Bederson *et al.*, 1996] cité page 220
Benjamin B. Bederson, James D. Hollan, Allison Druin, Jason Stewart, David Rogers, et David

Proft. Local tools : an alternative to tool palettes. Dans *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 169–170. Seattle, WA, USA, 1996. ACM Press, New York, NY, USA.

[Bederson *et al.*, 2000] cité pp. 158, 161, 214
Benjamin B. Bederson, Jon Meyer, et Lance Good. Jazz : an extensible zoomable user interface graphics toolkit in java. Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 171–180. San Diego, CA, USA, 5–8 Novembre 2000. ACM Press, New York, NY, USA.

[Bederson *et al.*, 2004] cité pp. 158, 161, 214
Benjamin B. Bederson, Jesse Grosjean, et Jon Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30(8) :535–546, 2004.

[Bederson, 2000] cité page 194
Benjamin B. Bederson. Fisheye menus. Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 217–226. San Diego, CA, USA, 5–8 Novembre 2000. ACM Press, New York, NY, USA.

[Berry, 2000] cité page 153
Gérard Berry. *The Foundations of Esterel*. MIT Press, 2000.

[Bier *et al.*, 1993] cité page 199
Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, et Tony D. DeRose. Toolglass and magic lenses : the see-through interface. Dans *SIGGRAPH '93 : Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. Anaheim, CA, USA, 1993. ACM Press, New York, NY, USA.

[Bier et Freeman, 1991] cité pp. 150, 214
E. A. Bier et S. Freeman. MMM : A user interface architecture for shared editors on a single screen. Dans *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST '91)*, pages 79–86. Hilton Head, SC, USA, 11-13 Novembre 1991. ACM Press, New York, NY, USA.

[Boden, 1997] cité page 8
Margaret A. Boden. *The Creative Mind : Myths and Mechanisms*. Weidenfeld and Nicolson, London, 1997.

[Bolt, 1980] cité page 109
Richard A. Bolt. « put-that-there » : Voice and gesture at the graphics interface. Dans *SIGGRAPH '80 : Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 262–270. Seattle, Washington, United States, 1980. ACM Press, New York, NY, USA.

[Bonnardel et Piolat, 2003] cité page 14
Nathalie Bonnardel et Annie Piolat. Design activities : how to analyze cognitive effort associated to cognitive treatments ? *The International Jouranl of Cognitive Technology*, 8(1) :6–15, 2003.

[Borning, 1979] cité page 218
Alan Hamilton Borning. *Thinglab - A Constraint-Oriented Simulation Laboratory*. Thèse de Doctorat, Stanford University, Juillet 1979. Également disponible comme rapports de recherche STAN-CS-79-746 (Stanford Computer Science Department) et SSL-79-3 (XEROX Palo Alto Research Center).

[Boucard *et al.*, 2002] cité page 44
Didier Boucard, Stéphane Huot, Christian Colin, Daniel Siret, et Gérard Hégron. An image-based

and knowledge-based system for efficient architectural and urban modeling. Dans *Proceedings of the international conference ACADIA*, pages 231–240. Los Angeles, CA, USA, Octobre 2002.

[Boucard et Colin, 2002] cité page 43
Didier Boucard et Christian Colin. Vers un modeleur architectural cognitif : le projet MArCo. *Revue internationale de CFAO et d'informatique graphique*, 17 :197–216, 2002.

[Boucard, 2004] cité pp. 26, 43
Didier Boucard. *Une méthode de reconstitution de scène 3D fondée sur une base de connaissances. Application à l'architecture classique*. Thèse de doctorat, Université de Nantes, CERMA, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, Novembre 2004.

[Broll *et al.*, 2004] cité page 139
Wolfgang Broll, Irma Lindt, Jan Ohlenburg, Michael Wittkämper, Chunrong Yuan, Thomas Novotny, Ava Fatah gen. Schieck, Chiron Mottram, et Andreas Strothmann. Arthur : A Collaborative Augmented Environment for Architectural Design and Urban Planning. *Journal of Virtual Reality and Broadcasting*, 1(1), Décembre 2004.

[Buxton, 1986] cité page 108
William Buxton. There is more to interaction than meets the eye : Some issues in manual input. Dans Donald A. Norman et Steve W. Draper, éditeurs, *User Centered System Design : New Perspectives on Human-Computer Interaction*, pages 319–337. 1986. Lawrence Erlbaum Associates, Hillsdale, NJ, USA.

[Béguin et Darses, 1998] cité pp. 5, 29
Pascal Béguin et Françoise Darses. Les concepteurs au travail et la conception des systèmes de travail : points de vue et débats. Dans *Deuxièmes journées Recherche et Ergonomie*. Toulouse, 1998. `http://www.ergonomie-self.org/rechergo98/index.html` .

[Cadoz, 1994] cité page 100
Claude Cadoz. Le geste, canal de communication homme/machine : la communication instrumentale. *Technique et Science de l'Information*, 13(1) :31–61, 1994.

[Candy et Edmonds, 1999] cité page 7
Linda Candy et Ernest A. Edmonds. Introducing creativity to cognition. Dans *C&C '99 : Proceedings of the third conference on Creativity & cognition*, pages 3–6. Loughborough, United Kingdom, 1999. ACM Press, New York, NY, USA.

[Candy et Edmonds, 2002] cité page 29
Linda Candy et Ernest A. Edmonds. Modeling co-creativity in art and technology. Dans *C&C '02 : Proceedings of the 4th conference on Creativity & cognition*, pages 134–141. Loughborough, UK, 2002. ACM Press, New York, NY, USA.

[Chambers *et al.*, 1983] cité page 203
John M. Chambers, William S. Cleveland, Beat Kleiner, et Paul A. Turkey. *Graphical Methods for Data Analysis*. Wadsworth, 1983.

[Chatty *et al.*, 2004] cité pp. 17, 151, 224
Stéphane Chatty, Stéphane Sire, Jean-Luc Vinot, Patrick Lecoanet, Alexandre Lemort, et Christophe P. Mertz. Revisiting visual interface programming : creating gui tools for designers and programmers. Dans *Proceedings of the 17th ACM Symposium on User Interface and Software Technology (UIST 2004)*, pages 267–276. Santa Fe, NM, USA, 24–27 Octobre 2004. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Chatty et Lecoanet, 1996] cité page 100
Stéphane Chatty et Patrick Lecoanet. A pen-based Workstation for Air Traffic Controllers. Dans *Proceedings of the ACM CHI 1996 Conference on Human Factors in Computing Systems*, pages 87–94. 1996. ACM/SIGCHI, ACM Press.

[Chatty, 1994] cité page 150
Stephane Chatty. Extending a graphical toolkit for two-handed interaction. Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Two Hands and Three Dimensions, pages 195–204. 1994.

[Chupin, 2002] cité pp. 9, 15
Jean-Pierre Chupin. *« La mariée mise à nu... » (à propos de l'enseignabilité des modèles de la conception)*, Dans Mario Borillo et Jean-Pierre Goulette, éditeurs, *Cognition et création : explorations cognitives des processus de conception*, chapitre 3, pages 65–95. Mardaga, Bruxelles, 2002.

[Coutaz, 1987] cité page 210
Joelle Coutaz. PAC, an object-oriented model for dialog design. Dans Hans-Jorg Bullinger et Brian Shackel, éditeurs, *Proceedings INTERACT'87 - 2nd IFIP International Conference on Human-Computer Interaction*, pages 431–436. Stuttgart, Germany, 1–4 Septembre 1987.

[Coüasnon et Camillerapp, 2003] cité page 98
Bertrand Coüasnon et Jean Camillerapp. Accès par le contenu aux documents manuscrits d'archives numérisés. *Document numérique*, 7(3-4) :61–84, 2003.

[Cross, 2002] cité pp. 7, 10, 36
Nigel Cross. *Comprendre la pensée du concepteur*, Dans Mario Borillo et Jean-Pierre Goulette, éditeurs, *Cognition et création : explorations cognitives des processus de conception*, chapitre 1. Mardaga, Bruxelles, 2002.

[Csikszentmihalyi, 1999] cité page 8
Mihaly Csikszentmihalyi. *Creativity : Flow and the Psychology of Discovery and Invention*. Harper Collins, New York, 1999.

[Cycling'74, 2004] cité page 143
Cycling'74. Page web de jitter., 2004. `http://www.cycling74.com/products/jitter.html` .

[Dassault Systemes, 2002 2004] cité pp. 25, 34
Dassault Systemes. Page web de catia., 2002–2004. `http://www.3ds.com/products-solutions/brands/CATIA/` .

[Day, 2004] cité page 25
Martin Day. Architect frank gehry finds cad a boon to art and business. CAD Digest, 23 Février 2004. `http://www.caddigest.com/subjects/aec/select/022304_day_gehry.htm` .

[de Bono, 1973] cité page 8
Edward de Bono. *Lateral Thinking*. Harper Colophon Books, New York, 1973.

[De Paoli, 2004] cité page 6
Giovanni De Paoli. Méthodes de modélisation numérique et artistique : cas de la plate-forme virtuelle de création pour les artistes des nouveaux complexes cirque du cirque du soleil. *Computer Art Journal*, 1(1), 2004.

[Debevec *et al.*, 1996] cité page 44
Paul E. Debevec, Camillo J. Taylor, et Jitendra Malik. Modeling and rendering architecture from

photographs : a hybrid geometry- and image-based approach. Dans *SIGGRAPH '96 : Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. 1996. ACM Press, New York, NY, USA.

[Denoue *et al.*, 2003] cité page 129
Laurent Denoue, Les Nelson, et Elizabeth Churchill. A fast, interactive 3d paper-flier metaphor for digital bulletin boards. Dans *UIST '03 : Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 169–172. Vancouver, Canada, 2003. ACM Press, New York, NY, USA.

[Do, 1998] cité page 52
Ellen Yi-Luen Do. *The Right Tool at the Right Time, investigation of freehand drawing as an interface to knowledge based design tools*. Thèse de Doctorat, College of Architecture at Georgia Institute of Technology, 1998.

[Do, 2000] cité page 52
Ellen Yi-Luen Do. Sketch That Scene For Me. Creating Virtual Worlds by Freehand Drawing. Dans *eCAADe 2000 'Promise and Reality'. Proceedings of the 18th conference on Education in Computer Aided Architectural Design in Europe*, pages 265–268. 22–24 Juin 2000.

[Donikian et Hégron, 1993] cité page 42
Stéphane Donikian et Gérard Hégron. A Declarative Design Method for 3D Scene Sketch Modeling. Dans *Proceedings of the Eurographics'93 Conference*. Barcelona, Spain, Septembre 1993.

[Dragicevic *et al.*, 2004] cité page 153
Pierre Dragicevic, David Navarre, Philippe Palanque, Amélie Schyn, et Rémi Bastide. Very-high-fidelity prototyping for both presentation and dialogue parts of multimodal interactive systems. Dans *DSVIS/EHCI 2004, joint conference 11th workshop on Design Specification and Verification of Interactive Systems and Engineering for HCI*. Tremsbüttel Castle, Hamburg, Germany, 11–13 Juillet 2004. ? Press.

[Dragicevic et Fekete, 1999] cité page 111
Pierre Dragicevic et Jean-Daniel Fekete. Étude d'une boîte à outils multi-dispositifs. Dans *Actes de la 11ième conférence francophone d'Interaction Homme-Machine (IHM99)*, pages 55–62. Montpellier, France, Novembre 1999. Cepadues.

[Dragicevic et Fekete, 2001] cité page 152
Pierre Dragicevic et Jean-Daniel Fekete. Input device selection and interaction configuration with ICON. Dans A. Blandford, J. Vanderdonkt, et P. Gray, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, pages 543–558. Lille, France, 2001. Springer Verlag.

[Dragicevic et Fekete, 2004] cité pp. 111, 153, 180
Pierre Dragicevic et Jean-Daniel Fekete. The input configurator toolkit : Towards high input adaptability in interactive applications. Dans *AVI'2004 Advanced Visual Interfaces*. Gallipoli, Italie, 25–28 Mai 2004. ACM Press, New York, NY, USA.

[Dragicevic, 2004a] cité pp. 192, 221
Pierre Dragicevic. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. Dans *Proceedings of the 17th ACM Symposium on User Interface and Software Technology (UIST 2004)*, pages 193–196. Santa Fe, NM, USA, 24–27 Octobre 2004. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Dragicevic, 2004b] cité pp. 75, 108, 109, 111, 148, 152, 153, 155, 180, 210, 223
Pierre Dragicevic. *Un modèle d'interaction en entrée pour des systêmes interactifs multi-dispositifs*

hautement configurables. Thèse de doctorat, Université de Nantes, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, Mars 2004.

[Druin *et al.*, 1997] cité page 220
Allison Druin, Jason Stewart, David Proft, Benjamin B. Bederson, et Jim Hollan. Kidpad : a design collaboration between children, technologists, and educators. Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 463–470. Atlanta, GA, United States, 1997. ACM Press, New York, NY, USA.

[Eckstein et Loy, 2002] cité page 149
Robert Eckstein et Marc Loy. *Java Swing*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2e edition, 2002.

[Edmonds et Candy, 2002] cité page 7
Ernest A. Edmonds et Linda Candy. Creativity, Art Practice, and Knowledge. *Communications of the ACM*, 45(10) :91–95, Octobre 2002.

[Eggli *et al.*, 1997] cité page 52
Lynn Eggli, Ching yao Hsu, Beat D. Brüderlin, et Gershon Elber. Inferring 3d models from free-hand sketches and constraints. *Computer-Aided Design*, 29(2) :101–112, 1997.

[Eisentraut et Günther, 1997] cité page 15
Renate Eisentraut et J Günther. Individual styles of problem solving and their relation to representations in the design process. *Design Studies*, 18(4) :369–383, Octobre 1997.

[Esteban *et al.*, 1995] cité page 218
O. Esteban, S. Chatty, et P. Palanque. Whizz'Ed : a visual environment for building highly interactive software. Dans *Proceedings of IFIP INTERACT'95 : Human-Computer Interaction*, pages 121–126. Lillehammer, Norway, 1995.

[Esteban, 1997] cité pp. 150, 152
Olivier Esteban. *Programmation visuelle pour la construction d'interfaces homme-machine hautement interactives*. Thèse de Doctorat, Laboratoire Interface Homme Systèmes (LIHS), Avril 1997.

[Estevez, 2001] cité pp. 18, 19, 22, 28, 36, 38, 39, 57, 143
Daniel Estevez. *Dessin d'architecture et infographie. L'évolution contemporaine des pratiques graphiques*. CNRS Editions, Paris, 2001.

[Faure et Julia, 1993] cité page 104
Claudie Faure et Luc Julia. Interaction homme-machine par la parole et le geste pour l'édition de documents : TAPAGE. Dans *Actes Interface des Mondes Réels et Virtuels*, pages 171–180. Montpellier, 1993.

[Fekete *et al.*, 1995] cité page 212
Jean-Daniel Fekete, Érick Bizouarn, Éric Cournarie, Thierry Galas, et Frédéric Taillefer. Tictactoon : a paperless system for professional 2d animation. Dans *SIGGRAPH '95 : Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 79–90. 1995. ACM Press, New York, NY, USA.

[Fekete et Beaudouin-Lafon, 1996] cité pp. 86, 131, 185, 212
Jean-Daniel Fekete et Michel Beaudouin-Lafon. Using the multi-layer model for building interactive graphical applications. Dans *Proceedings of the 9th ACM Symposium on User Interface Software and Technology (UIST'96)*, Papers : Tools, pages 109–118. Seattle, WA, USA, 1996. ACM Press, New York, NY, USA.

[Fekete, 1996] cité pp. 86, 131, 185, 212
Jean-Daniel Fekete. *Un modèle multicouche pour la construction d'applications graphiques interactives*. Thèse de doctorat, Université de Paris-Sud, Orsay (France), Janvier 1996.

[Fekete, 2004] cité page 194
Jean-Daniel Fekete. The infovis toolkit. Dans *Proceedings of the 10th IEEE Symposium on Information Visualization (InfoVis'04)*, pages 167–174. Austin, TX, USA, 2004. IEEE Press.

[Fekete, 2005] cité page 226
Jean-Daniel Fekete. Nouvelle génération d'Interfaces Hommes-Machine pour mieux agir et mieux comprendre. Habilitation à diriger les recherches, Mai 2005.

[Ferguson, 1992] cité page 10
Eugene S. Ferguson. *Engineering and the mind's eye.* MIT Press, Cambridge, MA, 1992.

[fieldDesigner, Inc, 2004] cité page 138
fieldDesigner, Inc. Page web de fieldprofessional., 2004. `http://www.fielddesigner.com/` .

[Flach et Bennett, 1996] cité page 15
John M Flach et Kevin B Bennett. *A theoretical framework for representational design*, Dans R. Parasuraman et M. Mouloua, éditeurs, *Automation and Human Performance : Theory and Application*, pages 65–87. Lawrence Earlbaum Associates, Mahwah, N.J., USA, 1996.

[Flemming et Woodbury, 1995] cité page 26
Ulroch Flemming et Robert Woodbury. Software environment to support early phases in building design (SEED) : Overview. *Journal of Architectural Engineering*, 1(4) :147–152, 1995.

[Foley *et al.*, 1990] cité page 166
James Foley, Andries van Dam, Steve Feiner, et John Hughes. *Computer graphics Principles and Practice*. Addison-Wesley, 2e edition, 1990.

[Fonseca *et al.*, 2004] cité page 43
Manuel J Fonseca, Alfredo Ferreira, et Joaquim A Jorge. Towards 3D Modeling using Sketches and Retrieval. Dans John F Hughes et Joaquim A Jorge, éditeurs, *Proceedings of EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, pages 127–136. Grenoble, France, 30–31 Août 2004.

[Furnas, 1981] cité page 194
George W. Furnas. The fisheye view : a new look at structured files. *Readings in information visualization : using vision to think (1999 reprinting)*, pages 312–330, 1981.

[Furnas, 1986] cité page 194
George W. Furnas. Generalized fisheye views. Dans *CHI '86 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23. Boston, Massachusetts, United States, 1986. ACM Press, New York, NY, USA.

[Fürst *et al.*, 2003] cité page 127
Frédéric Fürst, Michel Leclère, et Franky Trichet. Ontological engineering and mathematical knowledge management : a formalization of projective geometry. *Annals of Mathematics and Artficial Intelligence*, 2003.

[Fürst, 2004] cité page 127
Frédéric Fürst. *Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationalisation*. Thèse de Doctorat, Université de NANTES, Novembre 2004.

[Gero et Maher, 1993] cité page 26
John S. Gero et Mary L. Maher, éditeurs. *Modeling Creativity and Knowledge-Based Creative Design*, Hillsdale, NJ, USA, 1993. Lawrence Erlbaum.

[GiveMePower Corporation, 2004] cité page 138
GiveMePower Corporation. Page web de powercad pro., 2004. `http://www.givemepower.com/` .

[Goldsmith, 2002] cité pp. 10, 12, 37
Gabriela Goldsmith. Syntax of mark and gesture. *Tracey (the online journal dedicated to contemporary drawing issues)*, Septembre 2002. `http://www.lboro.ac.uk/departments/ac/tracey/somag/gabi.html` .

[Gooch *et al.*, 1998] cité page 185
Amy Gooch, Bruce Gooch, Peter Shirley, et Elaine Cohen. A Non-photorealistic Lighting Model for Automatic Technical Illustration. Dans *SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452. 2–4 Juillet 1998. ACM Press, New York, NY, USA.

[Goël et Pirolli, 1989] cité pp. 6, 10
Vinod Goël et Peter Pirolli. Motivating the notion of generic design within information-processing theory : The design problem space. *AI Magazine*, 10(1) :19–36, 1989.

[Goël, 1995] cité page 10
Vinod Goël. *Sketches of Thought*. The MIT Press, Cambridge, MA, USA, 1995.

[Grange, 1983] cité page 6
Kenneth Grange. *Kenneth Grange at the Boilerhouse*. Conran Foundation, Londres, 1983.

[Greenberg et Boyle, 2002] cité page 150
Saul Greenberg et Michael Boyle. Customizable physical interfaces for interacting with conventional applications. Dans *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST-02)*, pages 31–40. Paris, France, Octobre 27–30 2002. ACM Press, New York, NY, USA.

[Greenberg et Fitchett, 2001] cité pp. 150, 216
Saul Greenberg et Chester Fitchett. Phidgets : Easy Development of Physical Interfaces through Physical Widgets. Dans *Proceedings of the 14th Annual Symposium on User Interface Software and Technology (UIST-01)*, pages 209–218. Orlando, FL, USA, 11–14 Novembre 2001. ACM Press, New York, NY, USA.

[Greene, 2002] cité page 15
Sharon L. Greene. Characteristics of applications that support creativity. *Commununications of the ACM*, 45(10) :100–104, Octobre 2002.

[Gross et Do, 1996] cité page 52
Mark D. Gross et Ellen Yi-Luen Do. Demonstrating the electronic cocktail napkin. acm human factors in computing. Dans *ACM CHI 1996 Conference on Human Factors in Computing Systems Companion*. 1996. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Guena, 1997] cité pp. 26, 43
François Guena. Le raisonnement par classification appliqué à la cao, Mai 1997. Rapport d'Habilitation à diriger des recherches.

[Guiard, 1987] cité pp. 57, 106
Yves Guiard. Asymmetric division of labor in human skilled bimanual action : The kinematic chain as a model. *The Journal of Motor Behavior*, 19(4) :486–517, 1987.

[Guimbretière, 2003] cité page 28
François Guimbretière. Paper augmented digital documents. Dans *UIST '03 : Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 51–60. Vancouver, Canada, 2003. ACM Press, New York, NY, USA.

[Henry *et al.*, 1990] cité pp. 150, 214
Tyson R. Henry, Scott E. Hudson, et Gary L. Newell. Integrating Gesture and Snapping into a User Interface Toolkit. Dans *Proc. of the 3rd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST'90)*, pages 112–122. Snowbird, UT, USA, 1990. ACM Press, New York, NY, USA.

[Hong et Landay, 2000] cité pp. 150, 214
Jason I. Hong et James A. Landay. SATIN : a toolkit for informal ink-based applications. Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 63–72. San Diego, CA, USA, Novembre5–8 Novembre 2000. ACM Press, New York, NY, USA.

[Hopkins, 1991] cité page 105
Don Hopkins. The design and implementation of pie menus. *Dr. Dobb's Journal*, 16(12) :16–26, Décembre 1991.

[Hourcade et Bederson, 1999] cité pp. 150, 220
Juan Pablo Hourcade et Benjamin B. Bederson. Architecture and implementation of a java package for multiple input devices (MID). Rapport interne CS-TR-4018, University of Maryland, College Park, Mai 1999.

[Hudson *et al.*, 1997] cité page 149
Scott E. Hudson, Roy Rodenstein, et Ian Smith. Debugging lenses : A new class of transparent tools for user interface debugging. Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Making Things Visible, pages 179–187. Banff, Alberta, Canada, 1997. ACM Press, New York, NY, USA.

[Hudson et Smith, 1996] cité pp. 149, 214
Scott E. Hudson et Ian Smith. Ultra-lightweight constraints. Dans *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 147–155. Seattle, WA, USA, 1996. ACM Press, New York, NY, USA.

[Huot *et al.*, 2003] cité page 56
Stéphane Huot, Cédric Dumas, et Gérard Hégron. Toward creative 3D modeling : an architects' sketches study. Dans *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'03)*, pages 785–788. Zurich, Switzerland, Septembre 2003. IOS Press.

[Huot *et al.*, 2004a] cité page 159
Stéphane Huot, Cédric Dumas, Pierre Dragicevic, Jean-Daniel Fekete, et Gérard Hégron. The MAGGLITE post-wimp toolkit : Draw it, connect it and run it. Dans *Proceedings of the 17th ACM Symposium on User Interface and Software Technology (UIST 2004)*, pages 257–266. Santa Fe, NM, USA, 24–27 Octobre 2004. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Huot *et al.*, 2004b] cité page 159
Stéphane Huot, Cédric Dumas, Pierre Dragicevic, et Gérard Hégron. Conception et utilisation d'interactions avancées avec la boîte à outils MAGGLITE. Dans *Actes de la 16ème conférence francophone sur l'Interaction Homme-Machine, IHM 2004*, pages 177–178. Namur, Belgique, Août–Septembre 2004. AFIHM, ACM Press, New York, NY, USA.

[Huot *et al.*, 2004c] cité page 85
Stéphane Huot, Cédric Dumas, et Gérard Hégron. Svalabard : Une table à dessin virtuelle pour la modélisation 3D. Dans *Actes de la 16ème conférence francophone sur l'Interaction Homme-Machine, IHM 2004*, pages 85–92. Namur, Belgique, Août–Septembre 2004. AFIHM, ACM Press, New York, NY, USA.

[Huot et Colin, 2001] cité pp. 44, 87
Stéphane Huot et Christian Colin. MARINA : 3D reconstruction from images using formal projective geometry. Rapport de recherche 01/1/INFO, École des Mines de Nantes, Janvier 2001.

[Huot et Dumas, 2002] cité page 56
Stéphane Huot et Cédric Dumas. Vers des modeleurs 3D créatifs : Étude de dessins d'architectes. Dans *Actes de la 14éme conférence Francophone sur l'Interaction Homme-Machine, IHM 2002*, pages 267–270. Poitiers, France, 26–29 Novembre 2002. AFIHM, ACM Press, New York, NY, USA.

[Huot, 2000] cité pp. 44, 87, 139
Stéphane Huot. Reconstruction de bâtiments 3D à partir d'images. Mémoire de dea, École des Mines de Nantes, Université de Nantes, Septembre 2000.

[Huot, 2004] cité page 207
Stéphane Huot. Page web du projet MAGGLITE. Ecole des Mines de Nantes, équipe CMI, 2004. `http://www.emn.fr/x-info/magglite` .

[Hégron *et al.*, 2000] cité pp. 44, 139
Gérard Hégron, Daniel Siret, Christian Colin, Pierre Macé, Stéphane Huot, Eric Monin, et Didier Boucard. Une nouvelle approche informatique de la modélisation architecturale : la restitution de la place Ludovise de Louis. Dans *Colloque international Victor Louis et son temps*. Décembre 2000.

[Igarashi *et al.*, 1999] cité pp. 48, 128
Takeo Igarashi, Satoshi Matsuoka, et Hidehiko Tanaka. Teddy : A sketching interface for 3d freeform design. Dans *Proceedings of SIGGRAPH'99 conference*, pages 409–416. Los Angeles, CA, USA, 08-13 Août 1999. ACM/SIGGRAPH, ACMP Press, New York, NY, USA.

[Igarashi et Cosgrove, 2001] cité page 48
Takeo Igarashi et Dennis Cosgrove. Adaptive Unwrapping for Interactive Texture Painting. Dans *Proceedings of the ACM Symposium on Interactive 3D Graphics (ACM I3D'01)*, pages 209–216. Research Triangle Park, NC, USA, 19–21 Mars 2001. ACM Press, New York, NY, USA.

[Igarashi et Hughes, 2001] cité pp. 49, 135
Takeo Igarashi et John F Hughes. A suggestive interface for 3d drawing. Dans *Proceedings of the 14th ACM Symposium on User Interface and Software Technology (UIST 2001)*, pages 173–181. Orlando, FL, USA, 11-14 Novembre 2001. ACM/SIGCHI, ACMP Press, New York, NY, USA.

[Igarashi, 2003] cité page 48
Takeo Igarashi. Smooth Meshes for Sketch-based Freeform Modeling. Dans *Proceedings of the ACM Symposium on Interactive 3D Graphics (ACM I3D'03)*, pages 139–142. Monterey, CA, USA, 27–30 Avril 2003. ACM Press, New York, NY, USA.

[Ingalls *et al.*, 1988] cité page 218
Dan Ingalls, Scott Wallace, Yu-Ying Chow, Frank Ludolph, et Ken Doyle. Fabrik : A visual programming environment. Dans Norman Meyrowitz, éditeur, *OOPSLA'88 : Object-Oriented Programming Systems, Languages and Applications : Conference Proceedings*, pages 176–190. San Diego, CA, USA, 1988.

[ISO/CIE, 1991] cité page 92
ISO/CIE. Joint iso/cie standard : Colorimetric observers. Rapport technique, ISO/CIE, 1991.

[Jacob *et al.*, 1999] cité page 218
R. Jacob, L. Degliannidis, et S. Morrison. A Software Model and Specification Language for Non-WIMP User Interfaces. *ACM Transactions on Computer-Human Interaction*, 6(1) :1–46, Mars 1999.

[Jourdan *et al.*, 2004] cité page 137
Frédéric Jourdan, Gérard Hégron, et Pierre Macé. Automatic tessellation of quadric surfaces using grassmann-cayley algebra. Dans *Proceedings of International Conference on Computer Vision and Graphics (ICCVG-2004)*, page 674. Warsaw, Poland, Septembre 2004.

[Julia, 1995] cité page 104
Luc Julia. *Interface Homme-Machine Multimodale pour la Conception et l'Edition de Documents Graphiques*. Thèse de Doctorat, Ecole Nationale Supèrieure des Télécommunications, Paris, 1995.

[Krasner et Pope, 1988] cité pp. 158, 210
Glenn E. Krasner et Stephen T. Pope. A Description of the Model-View-Controller User Interface Programming. *Journal of Object Oriented Programming*, 1(3) :26–49, 1988.

[Kurmann *et al.*, 1997] cité page 40
David Kurmann, Nathanea Elte, et Maia Engeli. Real-time modeling with architectural space. Dans *CAADFutures 1997 - Proceedings of the 7th International Conference on Computer Aided Architectural Design Futures*. 1997.

[Kurmann, 1995] cité page 40
David Kurmann. A Tool for Intuitive Architectural Design. Dans *CAADFutures 1995 - Proceedings of the 5th International Conference on Computer Aided Architectural Design Futures*, pages 323–330. 1995.

[Kurtenbach et Buxton, 1991] cité page 105
Gordon Kurtenbach et William Buxton. Issues in Combining Marking and Direct Manipulation Techniques. Dans *Proceedings of the 4th ACM Symposium on User Interface Software and Technology (UIST '91)*, pages 137–144. Hilton Head, SC, USA, 11-13 Novembre 1991. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Kurtenbach et Buxton, 1994] cité page 105
Gordon Kurtenbach et William Buxton. User learning and performance with marking menus. Dans *Proceedings of the ACM CHI 1994 Conference on Human Factors in Computing Systems*, pages 258–264. 1994. ACM/SIGCHI, ACM Press, New York, NY, USA.

[Kuzo, 1999] cité pp. xiv, 76
Phil Massan Kuzo. *Des contraintes projectives en modélisation tridimensionnelle interactive*. Thèse de doctorat, Université de Nantes, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, Novembre 1999.

[Lacquaniti *et al.*, 1983] cité dans les annexes
F. Lacquaniti, F. Terzuolo, et P. Viviani. The law relating the kinematic and figural aspects of drawing movements. *Acta Psychologica*, 54 :115–130, 1983.

[Lacy, New York] cité page 10
Bill Lacy. *100 Contemporary architects : Drawings & sketches*. Abrams, New York.

[Lamb et Bandopadhay, 1990] cité page 52
Del Lamb et Amit Bandopadhay. Interpreting a 3d object from a rough 2d line drawing. Dans *Proceedings of Visualization'90*, pages 59–66. 1990.

[Landay et Myers, 1995] cité pp. 201, 217
James A. Landay et Brad A. Myers. Interactive sketching for the early stages of user interface design. Dans *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers : Programming by Example*, pages 43–50. Denver, CO, USA, 1995. ACM Press, New York, NY, USA.

[Landay, 1996] cité pp. 201, 217
James A. Landay. Silk : Sketching interfaces like krazy. Dans *Technical Video Program of CHI'96*. Vancouver, BC, Canada, apr 1996.

[@Last Software, 2000 2004] cité page 40
@Last Software. Page web de SketchUp™, 2000–2004. `http://www.sketchup.com` .

[Lavoisier-Hermes science publications, 2000 2004] cité page 98
Lavoisier-Hermes science publications. Revue document numérique, 2000-2004.

[Lavoisy, 2000] cité page xiii
Olivier Lavoisy. *La Matière et l'Action. Le graphisme technique comme instrument de la coordination industrielle dans le domaine de la mécanique depuis 3 siècles*. Thèse de Doctorat, université Pierre Mendès France, Grenoble, France, 2000.

[Lebahar, 1983] cité pp. 18, 20, 21, 24, 28, 61
Jean-Charles Lebahar. *Le dessin d'architecte - simulation graphique et réduction d'incertitude*. Collection architecture outils. Editions Parenthèses, 1 edition, 1983.

[Leclercq *et al.*, 2004] cité pp. 28, 51, 134
Pierre Leclercq, Geneviève Martin, Catherine Deshayes, et François Guena. Vers une interface multimodale pour une assistance à la conception architecturale. Dans *Actes de la 16ème conférence francophone sur l'Interaction Homme-Machine, IHM 2004*, pages 109–116. Namur, Belgique, Août–Septembre 2004. AFIHM, ACM Press, New York, NY, USA.

[Leclercq et Juchmes, 2002] cité pp. 88, 137
Pierre Leclercq et Roland Juchmes. The Absent Interface in Design Engineering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, AIEDAM Special Issue : Human-computer Interaction in Engineering Contexts*, 16(3) :219–227, 2002.

[Leclercq, 1996] cité page 67
Pierre Leclercq. Environnement de conception architecturale préintégrée. eléments d'une plateforme d'assistance basée sur une représentation sémantique. *Collection des publications de la Faculté des Sciences Appliquées de l'Université de Liège*, pages 174–175, 1996.

[Leclercq, 2004] cité pp. 51, 57
Pierre Leclercq. Invisible sketch interface in architectural engineering. *Lecture Notes in Computer Science*, 3088 :353–363, Octobre 2004.

[Lecolinet, 2003] cité pp. 158, 214
Eric Lecolinet. A molecular architecture for creating advanced interfaces. *CHI Letters*, pages 135–144, 2003.

[Lee *et al.*, 2005] cité page 46
Chia-Hsun Lee, Yuchang Hu, et Ted Selker. isphere : a proximity-based 3d input interface. Dans *Proceedings of CAAD Futures 2005*. 2005.

[Maccoby, 1991] cité page 6
Michael Maccoby. The innovative mind at work. *IEEE Spectrum*, pages 23–35, 1991.

[Mackay *et al.*, 1998] cité page 15
Wendy E. Mackay, Anne-Laure Fayard, Laurent Frobert, et Lionel Médini. Reinventing the familiar : exploring an augmented reality design space for air traffic control. Dans *CHI '98 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 558–565. Los Angeles, CA, USA, 1998. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Mamykina *et al.*, 2002] cité page 29
Lena Mamykina, Linda Candy, et Ernest Edmonds. Collaborative creativty. *Communications of the ACM*, 45(10) :96–99, Octobre 2002.

[Mankoff *et al.*, 2000] cité pp. 149, 150
Jennifer Mankoff, Scott E. Hudson, et Gregory D. Abowd. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. Dans Thea Turner, Gerd Szwillus, Mary Czerwinski, et Paternò Fabio, éditeurs, *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI-00)*, pages 368–375. The Hague, The Netherlands, Avril 1–6 2000. ACM Press, New York, NY, USA.

[Masry *et al.*, 2004] cité pp. 52, 138
Mark Masry, Dong Joong Kang, et Hod Lipson. A Freehand sketching interface for progressive construction and analysis of 3d objects. Dans *2004 AAAI Fall Symposium on making pen-based sketching intelligent and natural*. Washington, DC, USA, 2004.

[McCormack et Asente, 1988] cité page 149
Joel McCormack et Paul Asente. Using the X toolkit or how to write a widget. Dans *Proceedings of the USENIX Summer Conference*, pages 1–14. Juin 1988. USENIX Association, Berkeley, CA, USA.

[McGuire et Hughes, 2004] cité page 128
Morgan McGuire et John F. Hughes. Hardware-determined feature edges. Dans *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 35–147. Annecy, France, 2004. ACM Press, New York, NY, USA.

[Miller, 1956] cité dans les annexes
Georges A. Miller. The magical number seven plus or minus two - Some limits on our capacity for processing information. *Psychological Review*, 63 :81–97, 1956.

[Monin, 2001] cité page 29
Eric Monin. *Ambiances et dispositifs éphémères en milieu urbain : Une analyse critique de projets d'aménagements temporaires réalisés en France au XVIIIe et XXe siècles*. Thèse de Doctorat, Université de NANTES (École polytechnique de l'Université de NANTES), 2001.

[Myers *et al.*, 1997] cité pp. 149, 214
Brad A. Myers, Richard G. McDaniel, Robert C. Miller, Alan S. Ferrency, Andrew Faulring, Bruce D. Kyle, Andrew Mickish, Alex Klimovitski, et Patrick Doane. The Amulet environment : New models for effective user interface software development. *IEEE Transactions on Software Engineering*, 23(6) :347–365, Juin 1997.

[Myers, 1990] cité pp. 149, 151, 214, 217
Brad A. Myers. A new model for handling input. *ACM Transactions on Information Systems*, 8(3) :289–320, Juillet 1990.

[Myers, 2004] cité page 210
Brad A. Myers. *Graphical User Interface Programming*, Dans *Computer Science Handbook*, chapitre 48, pages 48.1–48.29. Chapman & Hall/CRC Press, Inc., second edition, 2004.

[Nadeau *et al.*, 1997] cité page 161
Dave Nadeau, Brad Grantham, Colin McCartney, Mitra, et Henry Sowizral. What 3d api for java should i use and why ? (panel). Dans *SIGGRAPH '97 : Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 452–453. 1997. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Nemetschek, 2005] cité page 34
Nemetschek. Page web de Allplan™, 2005. `http://www.nemetschek.fr` .

[Nielsen, 1993] cité page 114
Jakob Nielsen. Noncommand User Interfaces. *Communications of the ACM*, 36(4) :83–99, Avril 1993.

[Nienhaus et Döllner, 2004] cité page 128
Marc Nienhaus et Jürgen Döllner. Sketchy drawings. Dans *AFRIGRAPH '04 : Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 73–81. Stellenbosch, South Africa, 2004. ACM Press, New York, NY, USA.

[Norman et Draper, 1986] cité page 15
D. A. Norman et St.W. Draper. *User Centered System design*. Lawrence Earlbaum Assoc., Hillsdale, 1986.

[Norman, 1993] cité page 9
Donald A. Norman. *Things That Make us Smart : Defending Human Attributes in the Age of the Machine*. Perseus Publishing, Cambridge, MA, 1993.

[Norman, 1998] cité page 15
Donald A. Norman. *The Invisible Computer*. MIT Press, Cambridge, MA, 1998.

[Paulin, 2003] cité page 61
Michel Paulin. *Vocabulaire illustré de la construction*. Le Moniteur. Le Moniteur, 2 edition, 2003.

[Pereira *et al.*, 2004] cité page 48
João P Pereira, Vasco A Branco, Joaquim A Jorge, Nelson F Silva, Tiago D Cardoso, et Fernando Nunes Ferreira. Cascading Recognizers for Ambiguous Calligraphic Interaction. Dans John F Hughes et Joaquim A Jorge, éditeurs, *Proceedings of EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, pages 63–72. Grenoble, France, 30–31 Août 2004.

[Pérouse de Montclos, 2000] cité page 61
Jean-Marie Pérouse de Montclos. *Architecture : méthode et vocabulaire*. Editions du patrimoine, 3 edition, 2000.

[Petersen et Russel, 2004] cité page 185
Daniel Petersen et Kenneth Russel. 3D Application and Game Development With OpenGL®. Dans *Sun's 2004 Worldwide Java Developer Conference - `http://java.sun.com/javaone/` .* 2004. `https://jogl.dev.java.net/` .

[Puckette, 1991] cité pp. 143, 218
Miller Puckette. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal*, 15(3) :50–57, 1991.

[Pugh, 1992] cité page 52
David Pugh. Designing solid objects using interactive sketch interpretation. *Computer Graphics*, 25 :117–126, Mars 1992.

[Ravani *et al.*, 2003] cité page 42
Ioana Ravani, Dimitrios Makris, George Miaoulis, Petros Constantinides, Alexandros Petridis, et

Dimitri Plemenos. Implementation of Architecture-oriented Knowledge Framework in MultiCAD Declarative Scene Modeling System. Dans *Proceedings of 1st Balkan Conference in Informatics (BCI 2003)*. Thessaloniki, Greece, 21–23 Novembre 2003.

[Rekimoto et Saitoh, 1999] cité page 140
Jun Rekimoto et Masanori Saitoh. Augmented surfaces : A spatially continuous workspace for hybrid computing environments. Dans *CHI'99 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 378–385. Pittsburgh, PA, USA, 1999. ACM Press, New York, NY, USA.

[Rosin, 1997] cité page 113
Paul L. Rosin. Techniques for Assessing Polygonal Approximations of Curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6) :659–666, 1997.

[Roussel, 2002] cité page 149
Nicolas Roussel. Videoworkspace : une boîte à outils pour l'exploration de nouvelles techniques de gestion de fenêtres. Dans *Actes de la 14éme conférence Francophone sur l'Interaction Homme-Machine, IHM 2002*, pages 271–274. Poitiers, France, 26–29 Novembre 2002. AFIHM, ACM Press, New York, NY, USA.

[Roussel, 2003] cité page 129
Nicolas Roussel. Ametista : a mini-toolkit for exploring new window management techniques. Dans *CLIHC '03 : Proceedings of the Latin American conference on Human-computer interaction*, pages 117–124. Rio de Janeiro, Brazil, 2003. ACM Press, New York, NY, USA.

[Ruchaud et Plemenos, 2002] cité page 42
William Ruchaud et Dimitri Plemenos. Multiformes. a declarative modeller as a 3d scene sketching tool. Dans *Proceedings of International Conference on Computer Vision and Graphics, ICCVG 2002*. Zakopane, Poland, 25–29 Septembre 2002.

[Safin *et al.*, 2005] cité page 139
Stéphane Safin, Christelle Boulanger, et Pierre Leclercq. Premières évaluations d'un bureau virtuel pour un processus de conception augmenté. Dans *Actes de la 17ème conférence francophone sur l'Interaction Homme-Machine, IHM 2005 (à paraitre)*, page à paraitre. Toulouse, France, 27–30 Septembre 2005. AFIHM, ACM Press.

[Salber *et al.*, 1999] cité page 216
Daniel Salber, Anind K. Dey, et Gregory D. Abowd. The context toolkit : Aiding the development of context-enabled applications. Dans Marian G. Williams, Mark W. Altom, Kate Ehrlich, et William Newman, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI-99)*, pages 434–441. Pittsburgh, PA, USA, 15–20 Mai 1999. ACM Press, New York, NY, USA.

[Saund et Lank, 2003] cité page 114
Eric Saund et Edward Lank. Stylus input and Editing Without Prior Selection of Mode. Dans *Proceedings of the 16th ACM Symposium on User Interface and Software Technology (UIST 2003)*. Vancouver, BC, Canada, 24–27 Octobre 2003. ACM/SIGCHI, ACM Press, New York, NY, USA.

[sbm,] cité page 52
Annual eurographics workshop on sketch-based interfaces and modeling (since 2004). `http://www.eg.org/sbm` .

[Schmucker, 1986] cité page 158
K. Schmucker. MacApp : an application framework. *Byte Magazine*, 11(8) :189–193, Août 1986.

[Schumann *et al.*, 1996] cité pp. 89, 128
Jutta Schumann, Thomas Strothotte, Stefan Laser, et Andreas Raab. Assessing the effect of non-photorealistic rendered images in cad. Dans *CHI '96 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 35–41. Vancouver, British Columbia, Canada, 1996. ACM Press, New York, NY, USA.

[Schweikardt et Gross, 2000] cité page 52
Eric Schweikardt et Mark D. Gross. Digital Clay : deriving digital models from freehand sketches. *Automation in Construction*, 9(1) :107–115, Janvier 2000.

[Schön, 1983] cité pp. 8, 10
Donald A Schön. *The Reflective Practitioner : How Professionals Think in Action*. Basic Books, New York, 1983.

[Seitamaa-Hakkarainen et Hakkarainen, 2000] cité page 10
Pirita Seitamaa-Hakkarainen et Kai Hakkarainen. Visualization and sketching in the design process. *Design Journal*, 3(1), 2000.

[Sensable Technologies Inc, 2005] cité page 45
Sensable Technologies Inc. Page web de la société sensable., 2005. `http://www.sensable.com/` .

[Sezgin *et al.*, 2001] cité pp. 113, 137
Tevfik Metin Sezgin, Thomas Stahovic, et Randall Davis. Sketch based interfaces : early processing for sketch understanding. Dans *Proceedings of the 2001 workshop on Perceptive User Interfaces*, pages 1–8. Orlando, FL, USA, 2001. ACM Press, New York, NY, USA.

[Sezgin, 2001] cité dans les annexes
Tevfik Metin Sezgin. Feature point detection and curve approximation for early processing of free-hand sketches. Master's Thesis, Massachussetts Institute of Technology, Mai 2001.

[Shneiderman, 1983] cité pp. 91, 211
Ben Shneiderman. Direct manipulation : A step beyond programming languages. *IEEE Computer*, 16(8) :57–69, Août 1983.

[Shneiderman, 2000] cité pp. 16, 135
Ben Shneiderman. Creating creativity : User interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction*, 7(1) :114–138, Mars 2000.

[Shneiderman, 2002] cité page 16
Ben Shneiderman. *Leonardo's Laptop : Human Needs and the New Computing Technologies*. MIT Press, 2002.

[Shpitalni et Lipson, 1997] cité page 117
Moshe Shpitalni et Hod Lipson. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *Transactions of ASME, Journal of Mechanical Design*, 119(2) :131–135, 1997.

[Silicon Graphics, Inc, 2005] cité page 159
Silicon Graphics, Inc. Page web de open inventor, 2005. `http://oss.sgi.com/projects/inventor/` .

[Sosnov *et al.*, 2002] cité page 76
Alexey Sosnov, Stéphane Huot, Pierre Macé, et Gérard Hégron. Rapid incremental architectural modeling from imprecise perspective sketches and geometric constraints. Dans *Proceedings of the international conference Graphicon*. Linz, Autriche, 2002.

[Sosnov, 2003] cité pp. xiv, 76, 127
Alexey Sosnov. *Modélisation géométrique par séparation de contraintes*. Thèse de doctorat, Université de Nantes, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, Décembre 2003.

[Sun Microsystems, Inc, 2004a] cité dans les annexes
Sun Microsystems, Inc. Game technology group - java input api project., 2004. `https://jinput.dev.java.net/` .

[Sun Microsystems, Inc, 2004b] cité page 161
Sun Microsystems, Inc. Page web de java3d., 2004. `http://java.sun.com/products/java-media/3D/` .

[Sun Microsystems, Inc, 2005] cité dans les annexes
Sun Microsystems, Inc. Page web de java media framework., 2005. `http://java.sun.com/products/java-media/jmf/` .

[Sutherland, 1963] cité pp. xiii, 32
Ivan Edwards Sutherland. *Sketchpad : A Man-Machine Graphical Communication System*. Thèse de Doctorat, Massachusetts Institute of Technology, Janvier 1963.

[Sutherland, 1965] cité page 229
Ivan Edwards Sutherland. The Ultimate Display. Dans *Proceedings of IFIP Congress*, pages 506–508. 1965.

[Suwa *et al.*, 1999] cité page 10
Masaki Suwa, John S. Gero, et Terry Purcell. Unexpected discoveries and s-inventions of design requirements : A key to creative designs. Dans *Computational models of creative design IV*. Sydney, Australia, 1999.

[Suwa et Tversky, 1997] cité pp. 15, 58
Masaki Suwa et Barbara Tversky. What do architects see in their sketches ? a protocol analysis. *Design Studies*, 18(4) :385–403, Octobre 1997.

[Terry *et al.*, 2004] cité pp. 15, 16
Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, et Yasuhiro Yamamoto. Variation in element and action : supporting simultaneous development of alternative solutions. Dans *Proceedings of the 2004 conference on Human factors in computing systems*, pages 711–718. Vienna, Austria, 2004. ACM Press, New York, NY, USA.

[Terry et Mynatt, 2002] cité pp. 15, 16
Michael Terry et Elizabeth D Mynatt. Side-Views : Persistent, on demand previews for open-ended tasks. Dans *Proceedings of the 15th Annual Symposium on User Interface Software and Technology (UIST-02)*, pages 71–80. Paris, France, 27–30 Octobre 2002. ACM Press, New York, NY, USA.

[Tolba *et al.*, 2001] cité page 52
Osama Tolba, Julie Dorsey, et Leonard McMillan. A projective drawing system. Dans *SI3D '01 : Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 25–34. 2001. ACM Press, New York, NY, USA.

[Tyson R. *et al.*, 1990] cité page 149
Henry Tyson R., Hudson Scott E., et Newell Gary L. Integrating gesture and snapping into a user interface toolkit. Dans *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 112–122. Snowbird, UT, USA, 1990. ACM Press, New York, NY, USA.

[van Dam, 1997] cité pp. 44, 85
Andries van Dam. The human connection : Post-wimp user interfaces. *communications of the ACM*, 40(2) :63–67, Février 1997.

[Vander Zanden et Myers, 1995] cité pp. 151, 217
Bradley T. Vander Zanden et Brad A. Myers. Demonstrational and Constraint-Based Techniques for Pictorially Specifying Application Objects and Behaviors. *ACM Transaction on Computer Human Interaction*, 2(4) :308–356, Décembre 1995.

[Varley *et al.*, 2004] cité page 126
Peter Ashley Clifford Varley, Hiromasa Suzuki, et Ralph R Martin. Can Machines Interpret Line Drawings ? Dans John F Hughes et Joaquim A Jorge, éditeurs, *Proceedings of EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, pages 107–116. Grenoble, France, 30–31 Août 2004.

[Varley, 2003] cité page 126
Peter Ashley Clifford Varley. *Automatic Creation of Boundary-Representation Models from Single Line Drawings*. Thèse de Doctorat, Cardiff University, 2003.

[Vinot, 2004] cité page 92
Jean-Luc Vinot. De l'Art à l'interface. Cours donné lors de la 16ème conférence francophone sur l'Interaction Homme-Machine, IHM 2004, Septembre 2004.

[Virtools SA, 2001] cité pp. 150, 218
Virtools SA. Page web de virtools dev., 2001. `http://www.virtools.com` .

[Wacom Europe GmbH, 2003] cité page 98
Wacom Europe GmbH. Page web de wacom europe - produit *Cintiq Partner*, 2003. `http://www.wacom-europe.com/fr/products/cintiq/cintiqpartner.asp` .

[Wernecke, 1993] cité page 159
Josie Wernecke. *The Inventor Mentor : Programming Object-Oriented 3d Graphics with Open Inventor, Release 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[Whitefield, 1986] cité page 14
Andy D Whitefield. An analysis and comparison of knowledge use in designing with and without CAD. Dans *Knowledge engineering and computer modelling in CAD. Proceedings of the Seventh International Conference on the Computer as a Design Tool. (CAD 86)*. London, 1986. Butterworths.

[Wiseman *et al.*, 1969] cité page 105
Neil E. Wiseman, Heinz U. Lemke, et J O. Hiles. Pixie : a new approach to graphical man-machine communication. Dans *Proceedings of the 1969 CAD Conference 51*, page 463. Southampton, 1969. IEEE Conference Publications.

[Xith3D Project Group, 2003] cité page 185
Xith3D Project Group. Page web de xith3d., 2003. `http://xith.org/` .

[Zeleznik *et al.*, 1996] cité pp. 46, 89
Robert C Zeleznik, Kenneth P Herndon, et John F Hughes. Sketch : An interface for sketching 3d scenes. Dans *Proceedings of SIGGRAPH'96 conference*, pages 163–170. New Orleans, USA, Août 1996. ACM/SIGGRAPH, Addison Wesley.

[Zreik, 1990] cité page 6
Khaldoun Zreik. Sur la créativité assistée par ordinateur. Dans *Sur la modélisation des processus de conception créative, 01 DESIGN'90*, page 21. Octobre 1990. EUROPIA.

Table des figures

Table des tableaux

Table des algorithmes

Index

Annexes

Annexe A

Filtres de dessin : précisions et algorithmes

Sommaire

Dans cette annexe, nous apportons plus de précisions sur la réalisation et l'implémentation des filtres de dessin de SVALABARD, proposés dans la section 4.4 page 110 du chapitre 4. Dans un premier temps, nous reviendrons sur l'architecture globale de la cascade de filtres de traitements du dessin. Nous en détaillerons la réalisation basée sur le principe des dispositifs asynchrones d'ICON. Ensuite, nous détaillerons les principes et algorithmes de chacun des traitements effectués sur les traits du dessin.

A.1 Structure générale et architecture d'implémentation

Nous avons exposé et justifié dans la section 4.4 page 110 notre choix de construire une *cascade de filtres* pour rendre les traitements du dessin modulaires et évolutifs. Dès lors, nous avons utilisé ICON pour construire des dispositifs de traitement offrant à la fois la modularité et la flexibilité voulue, mais intégrant de plus ces filtres à la description de l'interaction. Cette intégration offre des possibilités de configuration avancée et d'adaptabilité de l'interaction aux résultats des traitements. Par contre, elle a nécessité une implémentation particulière pour que justement ces traitements ne pénalisent pas l'interaction.

En effet, les dispositifs standards d'ICON répondent à une logique de flot de données où les valeurs émises en sortie dépendent de celles reçues en entrée. Les dispositifs sont donc *synchrones*, ils reçoivent des valeurs, les traitent et émettent alors les sorties correspondantes. C'est pourquoi, afin de ne pas ralentir l'exécution de la configuration d'entrée et donc pénaliser l'interaction temps réel, il est nécessaire que les dispositifs respectent l'hypothèse réactive en n'effectuant pas de traitements trop lourd (supposant alors un temps de traitement nul). Or, les traitements que nous proposons sur les traits du dessin, bien qu'optimisés pour une utilisation temps réel, ne garantissent pas toujours un temps d'exécution assez rapide pour éviter un blocage de l'interaction. C'est pourquoi nous avons externalisé ces traitements dans des processus indépendants, liés à la configuration d'entrée par des dispositifs *asynchrones*.

A.1.1 Dispositifs asynchrones

Les dispositifs asynchrones d'ICON sont des dispositifs dont les sorties ne dépendent pas uniquement des entrés reçues. Ils peuvent donc émettre des valeurs de sortie même lorsqu'ils n'ont pas reçu de valeur d'entrée, ou bien des sorties qui ne dépendent pas uniquement de valeurs d'entrée (l'environnement extérieur à la configuration d'entrée). Par exemple, les dispositifs systèmes qui gèrent les périphériques d'entrée sont des dispositifs asynchrones. Ils sont liés à l'environnement et émettent des valeurs lorsqu'ils reçoivent des données de celui-ci. Pour cela, ICON fournit un mécanisme d'entrées/-sorties implicites associé à des principes de mise à jour spécifique des dispositifs (voir figure A.1 page ci-contre).

C'est ce type de dispositifs que nous avons utilisés pour réaliser les filtres de traitements du dessin. Ainsi, ils se contentent de recevoir les données à traiter en entrée (les traits, points ou segments) et de les envoyer à l'environnement. Dans le même temps, ils émettent les éventuelles données reçues de cet environnement. Ainsi, ils ne réalisent pas eux-mêmes les opérations de traitement, garantissant un temps d'exécution adapté à l'hypothèse réactive.

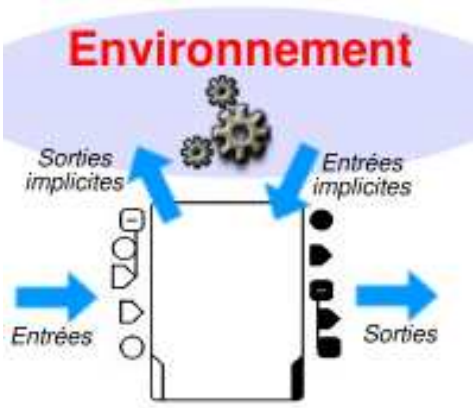


FIGURE A.1 – Les dispositifs asynchrones utilisent le mécanisme des entrées/sorties implicites pour communiquer avec l'environnement extérieur à la configuration d'entrée. Ils peuvent émettre des sorties même s'ils n'ont pas reçu de nouvelles valeurs en entrée.

A.1.2 Processus légers (*threads*)

Dés lors, chacun de ces dispositifs asynchrones de traitement du dessin est lié à un processus fils indépendant dans l'environnement d'exécution. Pour un dispositif, son processus léger (ou *thread*) réalise le traitement voulu en parallèle à l'exécution de la configuration d'entrée (voir figure A.2). De plus, pour la gestion des retours graphiques ou pour des traitements nécessitant une coopération (tels que la fusion des points et des segments, par exemple), les processus peuvent communiquer entre eux par l'intermédiaire d'un processus *superviseur*.

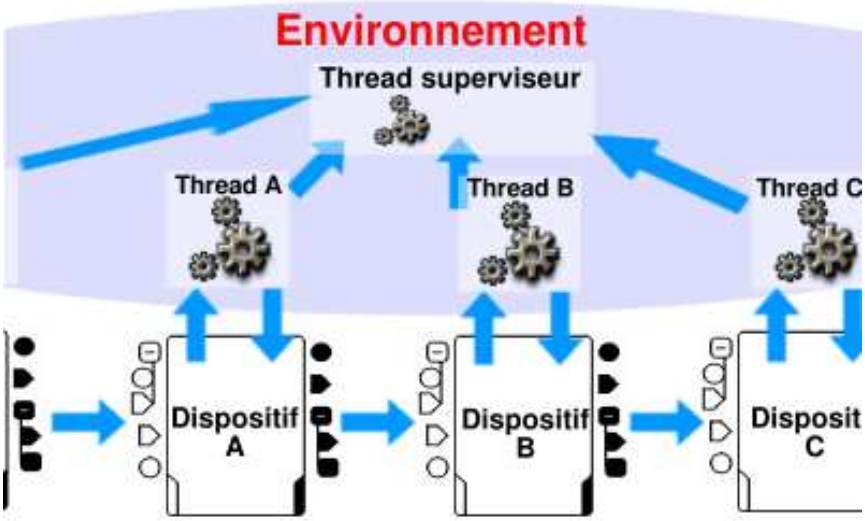


FIGURE A.2 – Chaque dispositif de traitement du dessin est lié à un thread de l'environnement d'exécution. Il lui envoie les données reçues en entrée et émet en sortie ses résultats. Les threads peuvent aussi communiquer entre eux par l'intermédiaire d'un superviseur.

À chaque pas d'exécution de la configuration d'entrée, les dispositifs envoient leurs éventuelles données reçues en entrée à leur processus associé (l'algorithme de traitement). Dans le même temps, ils émettent les éventuelles données de sortie (résultats du traitement) fournies par ce processus. Ainsi, l'émission de valeurs de sortie n'est pas bloquée par des traitements trop longs et coûteux et ne rallonge pas la durée d'exécution globale d'un « tick » de la machine réactive. Par contre, il est nécessaire d'évoquer les conséquences possibles d'une telle architecture pouvant induire en particulier des décalages dans le traitement des données.

A.1.3 Conséquences sur les traitements

En effet, cette externalisation des traitements des dispositifs implique que leurs valeurs de sortie (les résultats du traitement d'un trait) ne sont pas forcément émises dans le même pas d'exécution que celui où les entrées (le trait) sont reçues[(1)]. Cette conséquence de l'architecture proposée pourrait provoquer des « retards » dans le traitement des traits par rapport au moment où ils ont été tracés. De tels retards, si ils prenaient trop d'ampleur, pourraient s'avérer pénalisants, en particulier pour un filtre tel que la détection des phases de dessin qui peut être utilisé pour adapter l'interaction. Par exemple, un changement automatique d'outil basé sur cette détection pourrait intervenir trop tard par rapport à l'état courant du dessin.
Nous ne pouvons actuellement apporter qu'une réponse pratique à ce problème, basée sur les précautions que nous avons prises pour le minimiser et sur notre expérience d'utilisation du système. En effet, nos observations sur son comportement, sur les filtres réalisés et sur l'environnement d'exécution d'ICON nous permettent à l'heure actuelle d'éviter de tels dysfonctionnement.

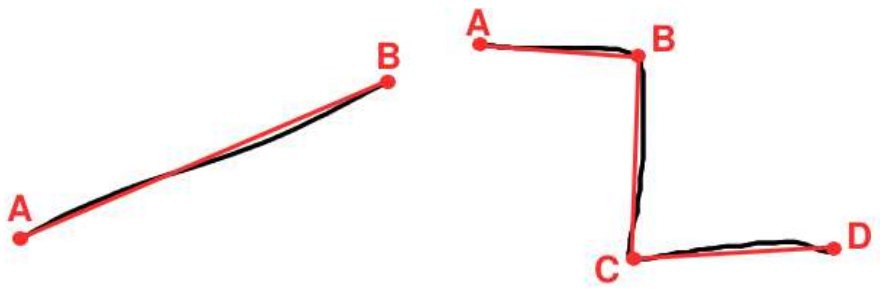
Tout d'abord, nous avons minimisé un éventuel décalage dans les traitements en conservant le trait comme donnée propagée entre les dispositifs de traitement. En effet, l'opération de segmentation peut décomposer un trait en plusieurs segments. Dès lors, si elle émettait des segments en sortie, elle provoquerait leur accumulation, et des segments issus d'un même trait seraient émis à des moments différents. Pour éviter ce problème, nous avons utilisé une seule donnée de propagation à travers les filtres de traitement : une structure de donnée de trait évoluée, contenant les informations de chaque traitement (segmentation, contexte, etc.). Ainsi, aucun décalage n'est induit si un trait comporte plusieurs segments.

Ensuite, la fréquence moyenne d'exécution d'une configuration d'entrée est de 60Hz, c'est à dire d'un pas toutes les 16,7ms. Cette valeur est largement inférieure à la durée de dessin d'un trait, même les plus rapides, ce qui implique que plusieurs pas sont franchis pendant qu'un trait est tracé. De plus, les traitements que nous avons proposés sont pour la pluspart de complexité linéaire et de temps moyen d'exécution assez faible pour produire un résultat en ne laissant passer que peu de pas. Nous avons alors constaté que les traitements sur les traits déjà dessinés étaient achevés avant la fin d'un nouveau trait.
Il convient toutefois de garder à l'esprit que cette affirmation est expérimentale, basée sur l'observation et que ce comportement devra être étudié plus en profondeur pour des traitements plus coûteux en temps.

[(1)] C'est d'ailleurs pour cela que de tels dispositifs sont qualifiés d'asynchrones.

A.2 Segmentation des traits

L'opération de *segmentation* d'un trait consiste à détecter les différents segments géométriques que représente un trait à main levée (voir section 4.4.3 page 113). La figure A.3 présente deux exemples de traits et leurs décompositions, les deux cas que l'on peut rencontrer lors d'un dessin à main levée : un trait représente un segment (voir figure A.3(a)) ou un trait représente plusieurs segments (voir figure A.3(b)).



(a) Un trait représente un segment.

(b) Un trait représente plusieurs segments.

FIGURE A.3 – Segmentation de traits. Cette figure représente des traits dessinés (en noir) et les segments qu'ils représentent (en rouge).

L'expérience que nous avons du dessin d'architecte et l'étude que nous avons menée ont montrées que le premier cas est le plus fréquent, un trait continu représente souvent un seul segment. Dès lors, l'opération de segmentation est on ne peut plus simple : le segment correspondant au trait est composé par les deux points extrêmes de celui-ci (A et B dans la figure A.3(a)). Mais, dans notre paradigme de dessin à main levée sans contraintes, nous ne pouvons pas écarter le second cas que l'on rencontre tout de même plus ou moins fréquemment chez les dessinateurs (selon leur style ou leurs habitudes). Dans ce cas général, le principe de la segmentation est de détecter, parmi les points capturés lors du tracé, ceux qui vont devenir les extrémités des segments (A, B, C et D dans la figure A.3(b)). Ces points sont appelés *points critiques* ou *coins*. Comme nous l'avons déjà évoqué dans ce mémoire, de nombreuses méthodes ont été proposées pour déterminer ces points dans les domaines de l'analyse d'images ou de l'interprétation temps réel. Nous ne détaillerons donc que celle que nous avons employée, combinant les données de courbure du trait et de vitesse de dessin en chaque point pour déterminer cette segmentation trait par trait.

A.2.1 Retour sur la méthode

Car en effet, beaucoup de travaux ont montré que le trait présente en ses points caractéristiques une courbure plus importante que les autres. C'est sur le calcul et l'analyse de cette donnée que sont basées la plupart des méthodes de détection. Toutefois, des variations trop minimes et lisses du rayon de courbure, ou des changements de direction trop fréquents peuvent entraîner la détection de « faux-positifs » ou l'omission de points. C'est pourquoi, dans une optique de détection temps réel où les information temporelle sont disponibles, Tevfik SEZGIN a proposé de combiner cette information de

courbure avec la vitesse de dessin. À partir des résultats proposés dans [Lacquaniti *et al.*, 1983] et de mesures de la diminution de la vitesse de dessin au voisinage des points critiques, il a proposé dans [Sezgin *et al.*, 2001] une méthode hybride que nous avons reproduite pour la segmentation des traits dans SVALABARD.

Nous en donnerons ici les grandes lignes. Pour des justifications plus approfondies et des résultats plus précis de cette méthode, nous renvoyons le lecteur aux publications dont elle est tirée [Sezgin *et al.*, 2001; Sezgin, 2001]. Le principe général est de calculer, pour chaque point enregistré du trait, une « mesure de certitude » qui va représenter son aptitude à être un point critique du trait. Ensuite, il faut constituer un ensemble de solutions hybrides et choisir la meilleure d'entre elles.

Mesures de la certitude des points

Courbure. La première mesure est exprimée par la grandeur mesurée de la courbure au voisinage de chaque point p_i :

Formule A.1 *Calcul de la mesure de courbure au voisinage d'un point.*

$$\frac{|d_{i-k} - d_{i+k}|}{l}$$

où l est la longueur de la courbe entre les points p_{i-k} et p_{i+k}, k est un entier qui définit la taille du voisinage considéré autour du point. Les valeurs d_i représentent la direction de la courbe au point p_i (l'angle formé par la tangente au trait au point donné avec l'axe des x).

Le calcul de la tangente soulève des problèmes de précision, résolus dans la méthode originale par l'utilisation d'une approximation numérique discrète : l'ODR (pour Orthogonal Distance Regression). Dans notre implémentation, nous avons utilisé un simple filtrage moyen des points qui assure une distance minimale entre deux points consécutifs pour un calcul simple par la formule : $d = \arctan(\Delta x, \Delta y)^2$ (Δx et Δy représentent le changement de position relatif entre deux points). Bien que moins précis que la méthode originale, les résultats sont tout de même satisfaisants [(2)]. Enfin, cette mesure est normalisée pour obtenir des valeurs dans l'intervalle $[0,1]$ pour chaque point.

Vitesse. La seconde mesure correspond à une mesure du ralentissement du stylet en chaque point p_i, exprimée par :

Formule A.2 *Calcul du ralentissement en un point.*

$$1 - \frac{v_i}{v_{max}}$$

où v_i est la vitesse instantanée entre les points p_{i-1} et p_i, et v_{max} est la vitesse instantanée maximale atteinte lors du dessin du trait [(3)].

Cette mesure donne aussi des valeurs comprises dans l'intervalle $[0,1]$.

[(2)] Une évolution prochaine de ce traitement consistera à l'implémentation de l'ODR en Java pour un calcul plus précis de la direction.

[(3)] *Les vitesses instantanées entre deux points sont simplement calculées en divisant la distance entre ces points par le temps écoulé entre la création de ces points.*

Ensemble de solutions hybrides

Ces premiers calculs donnent donc deux ensembles de points F_c et F_s, obtenus et triés par ordre décroissant à partir des données respectives de courbure et de vitesse. Une première solution hybride H_0 est alors construite par intersection de F_c et de F_s (le premier et le dernier point du trait, ainsi que les points de même ordre dans F_c et dans F_s).
Ensuite, deux nouvelles possibilités sont construites à partir de la solution précédente en lui ajoutant comme nouveau point caractéristique :

1. Le meilleur point candidat des données de vitesse. On obtient $H'_i = H_i \cup \{p_s\}$.
2. Le meilleur point candidat des données de courbure. On obtient $H''_i = H_i \cup \{p_c\}$.

À chacune de ces solutions est associée une mesure de pertinence, basée sur un calcul d'erreur aux moindres carrés : la somme moyenne des carrés des distances de chaque point du trait S à la solution.

Formule A.3 *Calcul de l'erreur pour une solution.*

$$\varepsilon_i = \frac{1}{|S|} \sum_{s \in S} ODSQ(s, H_i)^{(4)}$$

Ainsi, des deux solution possibles (vitesse ou courbure), celle de plus faible erreur (la plus *proche* du trait original) est conservée et devient H_{i+1}. Les itérations continuent alors à partir de cette solution jusqu'à ce que tous les points candidats aient été utilisés afin de construire un ensemble de solutions hybrides.

Choix de la solution

Dès lors, le choix de la segmentation à conserver parmi l'ensemble construit nécessite d'établir un compromis entre l'augmentation du nombre de points et la diminution de l'erreur. La solution retenue et justifiée par l'auteur de cette méthode est de conserver comme unique solution celle dont l'erreur est en dessous d'un seuil fixé, avec le moins de points possibles. Dans notre implémentation, nous avons choisi de rendre ce seuil d'erreur paramétrable afin de pouvoir simplement et interactivement ajuster la précision de l'algorithme.

A.2.2 Algorithme

L'algorithme simplifié[5] A.1 page suivante donne une vue plus synthétique de ce traitement.

A.3 Détection de la phase de dessin

L'algorithme que nous proposons pour détecter les phases du dessin architectural en perspective consiste à établir, pour chaque nouveau trait reçu, un classement des phases candidates selon la phase courante et les caractéristiques du trait reçu.

[4] *ODSQ (pour Orthogonal Distance SQuared) est le carré de la distance du point du trait au segment correspondant de la solution.*

[5] Certaines opérations, décomposées pour la compréhension, sont en fait regroupées dans l'implémentation à des fin d'optimisation.

Algorithme A.1 Algorithme de segmentation des traits.

proc SegmentationTrait(trait)

liste points ← trait.points //*liste des points du trait*

```
//calcul des mesures pour chaque point
pour tout point de points faire
    F_c.insert(point, calculMesureCourbure(point))
    F_s.insert(point, calculMesureVitesse(point))
fin pour

//construction de la première solution
H_0.add(points.first)
pour i=1 à points.length faire
    si F_c[i].point = F_s[i].point alors
        H_0.add(F_c[i].point)
        F_c.remove(i)
        F_s.remove(i)
    finsi
fin pour
H_0.add(points.last)
H_courante ← H_0

//construction et évaluation des solutions suivantes
tantque erreur(H_courante, trait) > seuil faire
    H_c ← copie(H_courante)
    si non F_c.vide() alors
        H_c.insert(F_c.removeFirst())
    finsi
    H_s ← copie(H_courante)
    si non F_s.vide() alors
        H_s.insert(F_s.removeFirst())
    finsi
    si erreur(H_c) < erreur(H_s) alors
        H_courante ← H_c
    sinon
        H_courante ← H_s
    finsi
fin tantque

//sauvegarde de la segmentation
trait.setSegmentation(H_courante)
```

fin proc

A.3.1 Retour sur la méthode

Le principe est donc d'affecter un *score* à chaque phase candidate en se basant sur des heuristiques établies lors de notre étude du dessin en perspective. Il est important de noter que nous ne cherchons pas à étiqueter précisément chaque trait selon la taxinomie que nous avons proposée. Cette caractérisation automatique nous semble d'ailleurs difficilement réalisable du fait d'une vision plus sémantique que géométrique ou numérique. Par contre, les données de notre étude ont montré que des séquences et des enchaînements de traits et de phases peuvent permettre de caractériser celles-ci, sans pour autant avoir une classification précise des traits.

Nous avions identifié trois phases dans le dessin architectural en perspective : *construction*, *amélioration* et *style*. Dès lors, à chaque nouveau trait reçu, nous sommes dans une situation de transition possible entre la phase courante et trois phases candidates. Ces phases candidates vont être évaluées en trois étapes, correspondant à trois types d'heuristiques (voir section 4.4.4 page 114) :

1. les *probabilités de transition* entre phases, établies selon les données de l'étude ;
2. certaines *caractéristiques mesurées ou observées des traits* de chaque phase ;
3. les *temps de pause* mesurés entre les phases.

À l'issue de cette évaluation, la phase dont le score est le plus élevé deviendra la phase courante, jusqu'à analyse d'un nouveau trait.

A.3.2 Algorithme

L'algorithme A.2 page suivante présente les détails de ce traitement. Outre les heuristiques de calcul et de modification des scores affectés à chaque phase potentielle, il fait aussi appel à des opérations de calcul et de détection de certaines caractéristiques des traits. Dans l'écriture de l'algorithme, ces opérations sont repérées par des notes que nous détaillons ci-après.

- **note 1.** La prise en compte du « gribouillage » part du constat lors de notre étude que de telles pratiques ne sont présentes que lors des phases d'amélioration ou de style. Dès lors, les scores sont modifiés en conséquence. L'opération `gribouillage(trait)` associée à la détection de ces traits est basée sur les données issues de la segmentation du trait (nombre de points critiques et importance des changements de direction) et sur le rapport entre la longueur du tracé et la longueur entre les deux extrémités.
- **note 2.** Ce rapport de longueurs est aussi une mesure rapide de la « linéarité » du trait, utilisée dans l'algorithme principal. Nous avons en effet aussi constaté que les traits des phases constructives sont en général plus linéaires, moins complexes que ceux des phases d'amélioration ou de style, d'où l'utilisation de cette donnée.
- **note 3.** L'opération `enZoneConstructive(trait)` permet de déterminer si un trait intersecte ou non la boîte englobante des traits déjà tracés lors des phases de constructives précédentes, ou s'il en est proche (à l'échelle du dessin). C'est une information importante pour la caractérisation des phases de construction et d'amélioration.
- **note 4.** L'opération `pauseLongue(traitPrecedent, trait, pause)` détermine le temps écoulé entre la fin du tracé du trait précédent et le début du tracé du trait en cours de traitement. Elle maintient aussi à jour la moyenne des temps de pause tout au long du dessin afin d'y comparer le temps de pause courant. Ainsi, le seuil permettant de déterminer si une

Algorithme A.2 Algorithme de détection des phases du dessin.

```
phase phaseCourante //phase courante du dessin
trait traitPrecedent //trait precedent le trait courant

proc DetectePhase(trait)
  si traitPrecedent = null alors
    //premier trait du dessin
    phaseNouvelle.valeur ← CONSTRUCTION
    phaseCourante ← phaseNouvelle
    traitPrecedent ← trait
    retour
  finsi

  //initialisation des scores de la nouvelle phase
  inititaliseScores(phaseCourante, phaseNouvelle)

  //pondération des scores par rapport aux caractéristiques du trait
  si gribouillage(trait)(note 1) alors
    phaseNouvelle.scoreAmelioration +← 100
    phaseNouvelle.scoreStyle +← 100
  finsi
  si trait.longueur < 20 alors
    phaseNouvelle.scoreConstruction +← 80
  finsi
  si trait.pressionMoyenne < 0.1 alors
    phaseNouvelle.scoreConstruction +← 20
  finsi
  si trait.rapportLongueurs > 0.8(note 2) alors
    phaseNouvelle.scoreConstruction +← 80
    phaseNouvelle.scoreAmelioration +← 60
    phaseNouvelle.scoreStyle +← 40
  finsi
  si enZoneConstructive(trait)(note 3) alors
    phaseNouvelle.scoreConstruction +← 100
    phaseNouvelle.scoreAmelioration +← 100
  finsi

  //pondération des scores par rapport au temps de pause
  si pauseLongue(traitPrecedent, trait, pause)(note 4) alors
    choix selon phaseCourante.valeur
      CONSTRUCTION : phaseNouvelle.scoreAmelioration +← pause + 83
                     phaseNouvelle.scoreStyle +← pause + 16
      AMELIORATION : phaseNouvelle.scoreConstruction +← pause + 74
                     phaseNouvelle.scoreStyle +← pause + 26
      STYLE : phaseNouvelle.scoreConstruction +← pause + 51
              phaseNouvelle.scoreAmelioration +← pause + 49
    finchoix
  finsi

  //mise à jour de la phase courante avec celle de meilleur score
  phaseNouvelle.valeur ← meilleurScore(phaseNouvelle)
  phaseCourante ← phaseNouvelle
  traitPrecedent ← trait
fin proc
```

pause est longue est adaptatif par rapport aux temps de pauses moyens afin de reproduire le comportement observé par nos analyses. De plus, cette méthode renvoie une valeur de pondération du score des phases candidates dans la variable `pause` afin de prendre en compte la distance entre le temps de pause courant et le seuil d'acceptation d'une pause longue (plus le temps de pause sera éloigné de la valeur seuil, plus cette pondération sera élevée, augmentant alors le score des phases potentielles). Cette approche *floue* procure un comportement moins catégorique à l'algorithme qu'une simple comparaison à un seuil fixe, améliorant sensiblement son efficacité lors de nos premiers tests sur les données de notre étude.

Ce traitement est basé sur les heuristiques et données numériques issues des observations de notre étude. Malgré la possibilité de paramétrer les seuils, il reste en partie fragile dans un contexte réel d'utilisation, avec un taux d'erreur encore trop élevé (de l'ordre de 10 pourcents sur les dessins rejoués de l'étude), en particulier lors de l'analyse de traits aux caractéristiques limites et ambiguës, et surtout propres au style de dessin de l'utilisateur. Nous pensons toutefois qu'il représente une première étape qui a permis d'identifier les caractéristiques utiles à la détection de ces phases.
Nous envisageons maintenant de rendre les heuristiques adaptatives à la manière de dessiner de l'utilisateur par des dessins d'apprentissage guidés où le dessinateur signifiera lui-même les changements de phase. Cette constitutions de profils d'utilisateurs permettra d'adapter les paramètres du traitement, en utilisant par exemple un réseau de neurones où le calcul des heuristiques fera l'objet des fonctions de transition.

A.4 Fusion des points

Étant donné les relatives imprécisions du dessin à main levée, associées à la précision des dispositifs numériques qui en permettent la saisie, il est rare que les extrémités de traits voulus concourants par le dessinateur coïncident exactement. Ainsi, pour la structuration et l'analyse du dessin, il est nécessaire d'opérer à la fusion des extrémités des segments détectés.

A.4.1 Retour sur la méthode

Comme nous l'avons exposé dans la section 4.4.5 page 116, nous avons implémenté la méthode proposée par Moshe SHPITALNI dans [Shpitalni et Lipson, 1997]. Elle présente l'avantage d'introduire des seuils de fusion adaptatifs aux éléments dessinés, contrairement aux approches précédentes qui utilisaient des seuils fixes pour la fusion de points considérés comme proches.

Ce seuil de tolérance, zone circulaire autour du point, va permettre de déterminer que deux points sont candidats à la fusion lorsqu'ils appartiennent chacun à la zone de l'autre. Ces zones sont calculées de la manière suivante :

1. tout d'abord, une taille maximale est définie pour chaque point, comme étant le tiers de la longueur du plus petit segment auquel il appartient (cette valeur expérimentale, fixée à la moitié de la longueur dans la méthode originale, permet d'assurer que les seuils de fusion ne grandirons pas dans des proportions trop incohérentes). Nous pouvons donc définir la valeur :

$$T_{max} = \min\{l_{s_0}, l_{s_1}, ..., l_{s_n}\}/3$$

avec l_{s_i} les longueurs des segments dont le point est une extrémité.

2. pour chaque point, la zone de tolérance est calculée comme étant la moyenne des distances du point à tous les segments du dessin. Pour les segments dont le point est une extrémité, le tiers de la longueur du segment est utilisée. Nous avons donc :

   Formule A.4 *Calcul de la zone de tolérance en un point.*

   $$T_p = \min\{T_{max}, \frac{1}{|S|}\sum_{s\in S} d(p,s)\}$$

   avec $d(p,s)$ la distance du point p au segment s ou $l_s/2$ si p est une extrémité de s.

Ainsi, plus il y a de segments dans le voisinage du point, plus la taille de la zone de tolérance à la fusion va diminuer, permettant alors de prendre en compte la densité de traits dans son voisinage.

Conçue à l'origine pour le traitement de dessins terminés, notre réalisation de cet algorithme en montre aussi la pertinence pour l'analyse de dessin en temps réel, moyennant quelques adaptations :

1. La prise en compte de l'évolution du dessin : il faut un minimum de points dessinés avant que la méthode ne rentre en action et les seuils de tolérance de fusion sont aussi dépendants de la résolution du dessin (le calcul de la moyenne des distances du point aux segments est pondéré par le rapport entre la taille de la feuille et la taille courante du dessin, permettant de prendre en compte le fait que le dessin va encore *grandir* et prendre plus d'espace au cours du temps). Nous pouvons donc reprendre la formule A.4 :

   Formule A.5 *Calcul de la zone de tolérance en un point avec adaptation à la résolution du dessin.*

   $$T_p = \min\{T_{max}, \frac{1}{|S| * (A_f/A_d)}\sum_{s\in S} d(p,s)\}$$

   où A_f est l'aire de la feuille de dessin et A_d l'aire du rectangle englobant le dessin.

2. La mise à jour des zones de tolérance et des fusions tout au long du dessin et lors des opérations de traitement sur les traits (ajout, fusion et retraits de segments).

Dès lors, deux points p_1 et p_2 seront fusionnés si :

Proposition A.1 *Conditions de fusion de deux points p_1 et p_2.*

$$d(p_1,p_2) \leq T_{p_1} \text{ et } d(p_1,p_2) \leq T_{p_2}$$

A.4.2 Algorithme

L'algorithme est alors simplement :

L'opération `miseAJourTolerance(point)` effectue le calcul du seuil de tolérance pour un point donné, en utilisant la formule A.5.
L'opération `formeGroupes(points)` permet de former des groupes de points respectant les conditions de fusion de la proposition A.1.
Enfin, l'opération `calculBarycentre(groupe)` renvoie le barycentre d'un groupe de points, utilisé pour remplacer les points dans l'opération `remplace(groupe, barycentre)` (cette opération va alors mettre à jour les segments dont les extrémités ont été remplacées et les plans dont les points ont été supprimés).

Algorithme A.3 Algorithme de fusion des points.

```
liste points //liste des points du dessin

proc FusionPoints()
  pour tout point de points faire
    miseAJourTolerance(point)
  fin pour
  groupes ← formeGroupes(points)
  pour tout groupe de groupes faire
    barycentre ← calculBarycentre(groupe)
    remplace(groupe, barycentre)
  fin pour
  pour tout point de points faire
    miseAJourTolerance(point)
  fin pour
fin proc
```

Cet algorithme est exécuté dans le flot de traitements à chaque fois qu'un nouveau segment est reçu et donc ajouté (ainsi que ses deux extrémités). Mais il est aussi invoqué en dehors de la cascade de filtres, lors de la fusion de segments (le traitement suivant). En effet, la fusion de segments occasionne des changements dans le calcul des seuils de tolérance de fusion des points. Le processus de fusion des segments utilise donc le mécanisme de communication directe par le superviseur pour déclencher la fusion des points si besoin, comme nous l'avons expliqué au début de cette annexe.

A.5 Fusion des segments

La méthode originale de fusion des segments que nous avons proposé permet de nettoyer et structurer le dessin à main levée en combinant des traits (ou plutôt les segments qui en découlent) censés ne représenter qu'un seul segment géométrique. Nous avions montré l'utilité et les principes généraux de ce traitement dans la section 4.4.6 page 117.

A.5.1 Retour sur la méthode

L'opération consiste à examiner deux à deux les segments issus des traits du dessin afin de déterminer si ils remplissent les conditions justifiant leur fusion. Ces conditions sont déterminées par les mesures suivantes, issues de nos observations et études sur le dessin en perspective :

1. des **critères positionnels**, concernant la *position globale* des deux segments dans le dessin (dans la même zone) et leur *position relative* déterminée par une estimation de la proximité des deux segments. Cette proximité est évaluée par l'intersection des boîtes englobantes et la distance séparant les deux segments.
2. un **critère relationnel** qui est l'angle formé par les deux segments.

Ces mesures sont alors comparées à des seuils expérimentaux permettant d'établir les possibilités de fusion des segments.

Lorsque deux segments remplissent les conditions, ils sont fusionnés en un seul par *composition* ou *assimilation* (voir figure 4.31 page 120).

A.5.2 Algorithme

L'algorithme est une boucle itérant sur l'ensemble des segments du dessin jusqu'à atteindre un point fixe (plus de fusion) où jusqu'à ce qu'un nouveau segment soit ajouté (un trait a été dessiné).

Algorithme A.4 Étapes de l'algorithme de fusion des segments.

répéter jusqu'à l'ajout d'un nouveau segment ou jusqu'à ce qu'il n'y ai plus de fusion

1. prendre deux segments non testés
2. si les segments sont dans la même zone, continuer. Sinon retourner à l'étape 1
3. si l'intersection de leur boîtes englobantes ou leurs distances sont inférieures aux seuils respectifs, continuer. Sinon, retourner à l'étape 1
4. si les segments sont concourants :
   (a) si l'extrémité commune appartient à plus de 3 segments et leur angle est inférieur au seuil, fusionner les segments. Retourner à l'étape 1.
   (b) si leur angle est inférieur au seuil, fusionner les segments. Retourner à l'étape 1.

   sinon, continuer
5. si leur angle est inférieur au seuil global, les fusionner. Retourner à l'étape 1.

Le choix des segments à analyser lors de chaque itération est guidé par la chronologie du dessin, c'est à dire que les segments sont étudiés du plus ancien au plus récent. Lorsque deux d'entre eux sont fusionnés, le segment résultant prend la place du plus ancien dans la chronologie. Cette approche se justifie par les résultats de notre étude qui ont montré que les traits (et donc les segments qui en découlent) sont tracés par groupes, dans une même phase. Ainsi, les fusions les plus nombreuses concernent des segments proches chronologiquement. Toutefois, des phases ultérieures de modification et de révision pourront aussi entraîner des fusions, c'est pourquoi chaque segment est tout de même comparé à tous les autres. Bien que pouvant paraître coûteuse au premier abord, cette comparaison est en fait relativement rapide, une majorité de candidats étant rapidement éliminés par le premier critère positionnel (appartenance à la même zone de dessin).

Bien que l'implémentation de cet algorithme soit réalisée à partir d'opération simples et optimisées par notre implémentation globale du système (boites englobantes et distances en cache, comparaisons, etc.), il convient d'anticiper d'éventuelles remarques quand à sa complexité. En effet, le principe de parcourir tous les segments pour les comparer à tous les autres conduit à réaliser des boucles imbriquées, donnant à notre algorithme une complexité d'ordre N^2 (la taille de la donnée est le nombre de segments). Ce n'est toutefois pas une limite, étant donné que le nombre de segments manipulé pour chaque dessin reste tout de même dans des proportions très faibles. Nous en voulons pour preuve que le total des 24 dessins complets de notre étude formaient un corpus d'environ 10000 traits. Même si chacun d'eux est décomposé en deux segments, lors de la segmentation, nous atteignons une moyenne d'environ 900 segments par dessin, ce qui ne posera pas de problème de complexité algorithmique. De plus, le traitement de détection des phases permet déjà de filtrer un certain nombre de segments qui ne seront pas transmis à cet algorithme et enfin notre implémentation sous forme de processus indépendants garantit que l'algorithme ne bloquera pas l'interaction, même en cas de traitements lourds.

Un point d'étude plus intéressant concerne par contre le comportement « local » de cet algorithme, que nous avons déjà évoqué rapidement dans le chapitre 5, page 133. En effet, la méthode proposée est basée sur des heuristiques qui qualifient bien les cas possibles et les singularités présents dans le

domaine d'application de nos travaux. Mais, comme toute méthode de ce type, elle pose le problème du comportement de l'algorithme au voisinage très proche des seuils choisis. C'est pourquoi nous envisageons de combiner ces heuristiques à une étude plus globale du graphe 2D formé par les points et segments du dessin, une étude topologique.

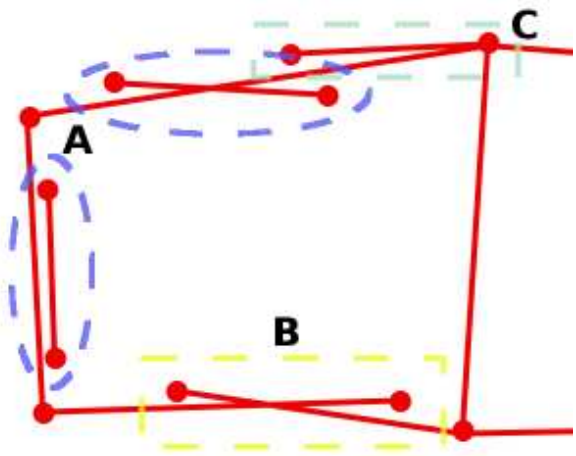


FIGURE A.4 – Fusion topologique de segments. Les segments détectés et connectés par les traitements précédents forment un graphe 2D du dessin. L'analyse de sa topologie, associée à nos heuristiques, pourrait permettre d'améliorer notre algorithme de fusion des segments.

Nous avons commencé à explorer une telle méthode, et identifié déjà trois cas, représentés dans la figure A.4. Le cas A, par exemple, permet d'identifier des segments issus de traits que l'on pourrait qualifier de *révision* (graphes composés de seulement deux sommets et une arrête). Dés lors, nos heuristiques positionnelles peuvent être appliquées pour déterminer à quel segment le fusionner. Le cas B peut être identifié par une recherche de pendants (sommets de degré 1) dans le graphe global afin de les fusionner (en utilisant nos heuristiques) pour former des cycles élémentaires. Cette méthode repose toutefois sur l'hypothèse que le dessin final va être représenté par un graphe planaire où tout sommet est dans un cycle (chaque cycle étant une *face*). Bien que ce soit le cas à la fin du dessin de l'enveloppe du bâtiment, cette hypothèse n'est pas vérifiée en début de dessin (tous les traits de l'enveloppe n'ont pas été dessinés, les faces ne sont pas toutes entièrement formées).
Il faudra alors envisager de retarder l'interprétation, ou de la décomposer en étapes. Il parait concevable d'utiliser dans ce but le résultat de notre étude montrant que la majorité des traits principaux primaires (l'enveloppe de la forme) sont tracés lors de la première phase constructive. Lors de cette phase, seules des fusions de points seraient prises en compte afin de former un premier graphe qui pourrait être révisé par la suite par fusions de segments comme proposé précédemment. Bien que n'étant plus vraiment « temps réel », cette approche permettrait une vision plus globale de l'interprétation, pouvant éviter des choix trop hâtifs grâce à une considération plus globale du problème.

Toujours dans cette optique de considérer le problème d'un point de vue global, il nous semble important d'envisager la mise en place de mécanismes de retour arrière sur les traitements, afin de pouvoir remettre en question l'interprétation et la corriger. Nous avons observé que le comportement des filtres était déterministe, dans le sens où l'interprétation des mêmes traits dans les mêmes conditions (chronologie, machine utilisée et ressources disponibles) sera toujours la même. Par contre, rien ne garantit qu'effectuer un traitement avant l'autre donnera le même résultat (fusionner deux points avant

deux segments, par exemple). C'est pourquoi il nous semble important, pour améliorer ces traitements et leur robustesse, d'étudier ces points et de mettre en place ces mécanismes de retour arrière[6].

A.6 Résultats des interprétations

Dans cette section, nous présentons deux dessins enregistrés sur lesquels nous avons appliqué en temps réel les filtres que nous avons proposés en les rejouant dans l'implémentation actuelle de SVALABARD. Ils présentent des erreurs d'interprétation principalement dues à la détection des phases du dessin (pour le premier) et à la fusion des points (pour le second).

A.6.1 Un dessin de l'étude

Le premier dessin est issu de notre étude préliminaire sur le dessin architectural en perspective (voir figure A.5 page ci-contre). Il faut donc souligner que c'est un dessin réellement « libre », réalisé dans les conditions expérimentales que nous avons définies dans la section 3.3 page 58. Il n'a pas été conçu pour et avec les interactions du système et ses interprétations sous-jacentes, ce qui en fait un bon test pour identifier les problèmes restant à résoudre à ce niveau (voir section 5.3.2 page 133).

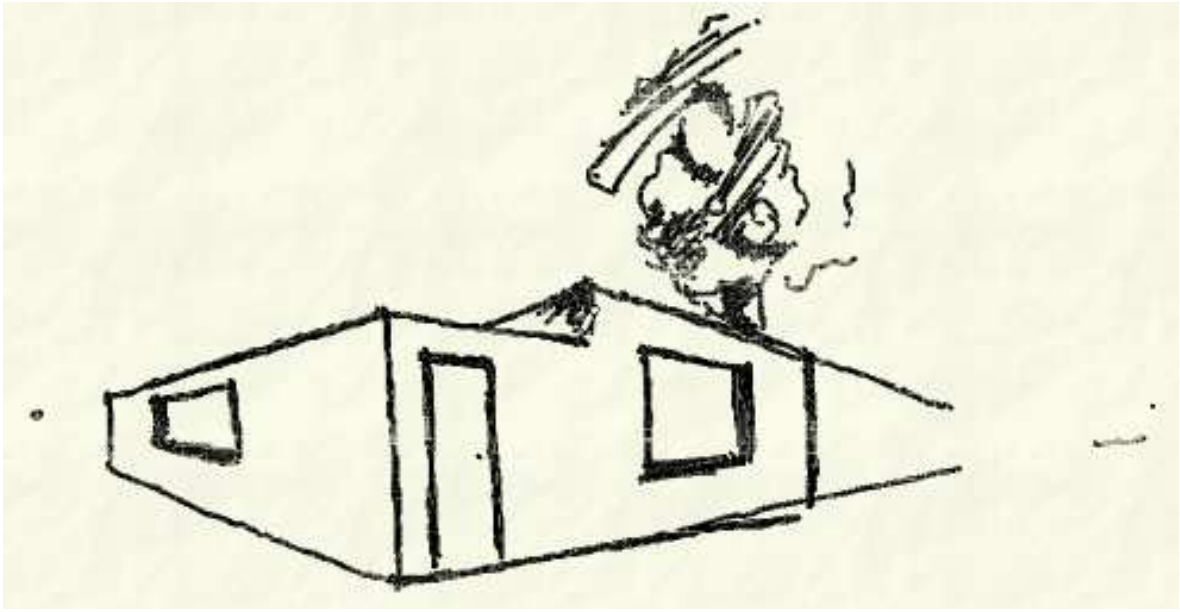
Bien que l'interprétation déduite par le système (voir figure A.5(b) page suivante) puisse suffire pour l'observateur, elle n'en demeure pas moins difficilement exploitable en l'état par un système de reconstruction.
Nous avons encadré et noté les erreurs directement sur la figure pour en commenter les causes :

1. Les erreurs notées A, B, C, D et E sont issues de détections erronées des phases du dessin, ayant alors entraîné un mauvais filtrage des traits. Pour l'erreur A, les traits nécessaires à l'interprétation sont manquants, alors que pour les autres des traits non désirés ont été gardés. Notons toutefois que dans ce cas, l'algorithme de fusion des segment a considérablement réduit leur nombre.
2. Les erreurs C et F sont dues à une mauvais opération lors de la fusion des segments, un « bogue » que nous n'avons pas encore identifié (dans le cas de l'erreur C, il est d'autant plus difficile à déterminer que les traits et le point restant devaient être filtrés par la détection des phases).
3. L'erreur G est un cas limite de la fusion des segments (angle très proche du seuil mais ne le dépassant pas) pour lequel les solutions topologiques que nous avons exposées précédemment seraient une solution possible.

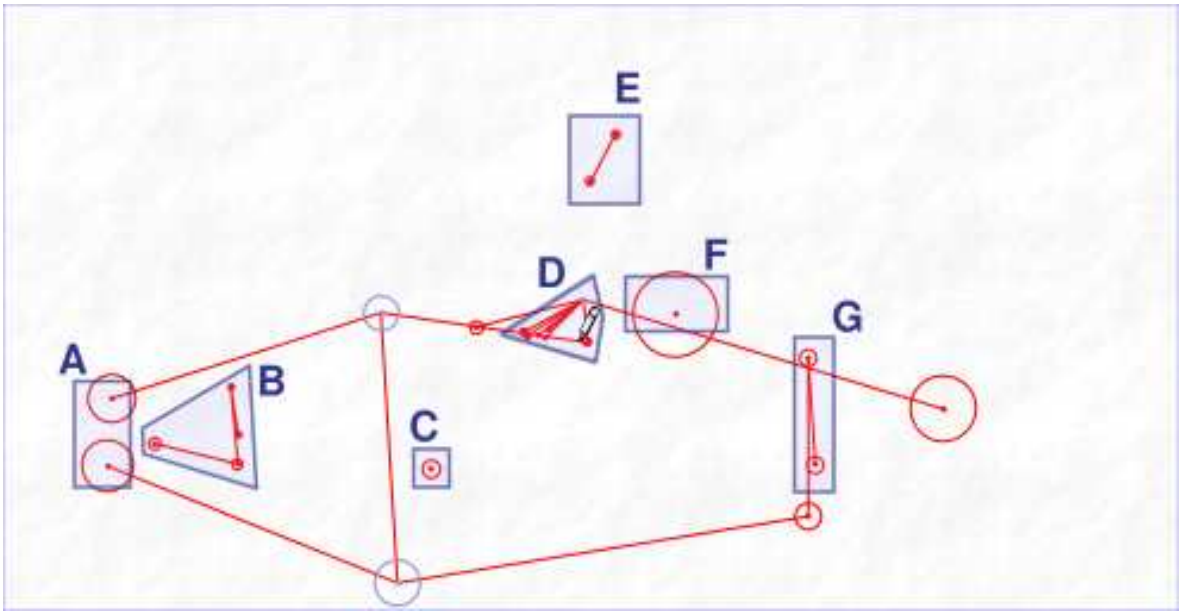
A.6.2 Un dessin avec le système

Le dessin suivant (voir figure A.6 page 21) a été réalisé et enregistré avec notre première implémentation du système. Toutefois, le dessinateur n'observait pas les traitement et ne disposait pas non plus des interactions avancées que nous avons proposées dans nos travaux ultérieurs. Le but était donc aussi de dessiner librement mais en observant un modèle, ceci dans l'optique d'enregistrer quelques

[6]Notons que même si nous ne nous sommes pas penchés sur ce problème, nos filtres conservent les données originales du traitement précédent, même si ils ne maintiennent pas d'historique de leurs traitements.



(a) Dessin original.



(b) Interprétation.

FIGURE A.5 – Dessin (figure (a)), interprétation (figure (b)) et erreurs de traitements (cadres).

dessins pour des essais futurs. Par contre, dans ce cas, l'utilisateur connaissait la teneur de nos travaux et les opérations de traitement qui se déroulaient en tâche de fond.

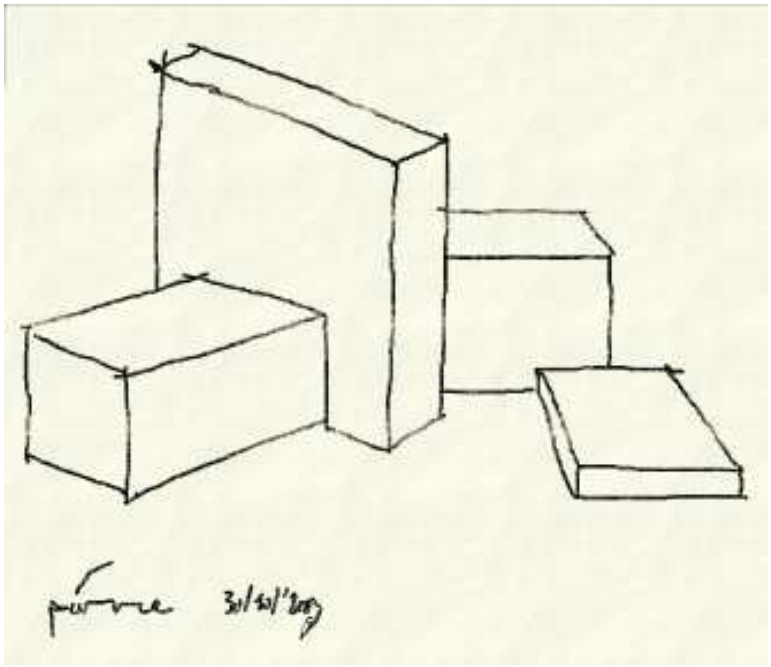
Dans ce cas, l'interprétation du dessin ne présente aucune erreur de filtrage des phases (toutefois moins nombreuses que pour le dessin précédent). Par contre, nous pouvons noter des erreurs dans la fusion des points, toutes dues à des cas « limites » :

1. Les erreurs *B* et *C* sont des cas où des points n'ont pas été fusionnés car à la limites des zones les un des autres.
2. Les erreurs *D* et *E*, à l'inverse, présentent des fusions indésirables de points, dues à une mauvaise adaptation de la taille des zones par rapport à la résolution du dessin.

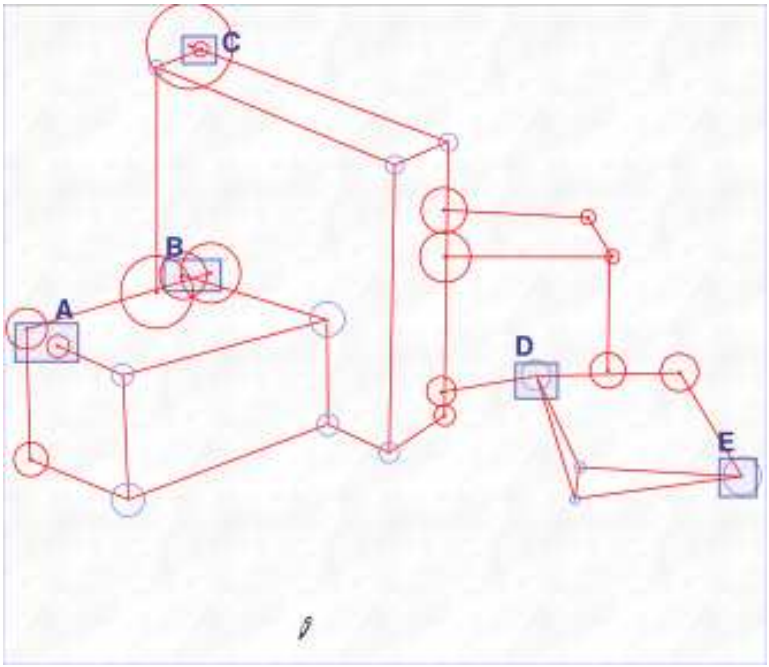
L'erreur *A*, par contre, est issue d'un problème lors de la numérisation d'un trait qui, même si il est complet dans le dessin original, a du être tronqué à cause d'une pression trop faible sur le stylet (celle-ci n'était pas enregistrée, c'est pour cela qu'elle est représentée constante dans la figure A.6(a) page suivante).

Poursuivre de telles observations sur les résultats des traitements proposés pourront permettre d'identifier et de résoudre les problèmes précis, en particulier pour la détection des phases du dessin avec les méthodes adaptatives que nous avons déjà évoquées.
Mais ces résultats soulèvent aussi des voies d'amélioration, en particulier l'introduction de méthodes plus « continues » pour prendre en compte les cas limites de fusion des segments et des points.

Enfin, ces deux exemples montrent relativement peu d'erreurs par rapport au nombre de traits présents dans les dessins (nous ne cachons toutefois pas que certains dessins présentent plus de problèmes, mais toujours du même ordre). Dans le dessin A.5 page précédente, nous n'avons pas non plus noté comme une erreur le segment « trop long » sur la droite du dessin, issu de la fusion d'un trait de construction avec des traits de l'enveloppe du bâtiment. Il nous parait peu envisageable de pouvoir considérer de tels cas de manière automatique. Par contre, vu le peu d'erreurs et de cas comme celui-ci, il est envisageable de les corriger manuellement, avec des interactions adaptées en manipulation directe sur l'interprétation.



(a) Dessin original.



(b) Interprétation.

FIGURE A.6 – Dessin (figure (a)), interprétation (figure (b)) et erreurs de traitements (cadres).

Annexe B

Données de l'étude sur le dessin d'architecture

Sommaire

Cette annexe recense les sujets de notre étude sur le dessin d'architecte présentée dans la section 3.3 page 58. Nous citons chacune de ces personnes (avec leur accord) pour les remercier individuellement et très chaleureusement d'avoir participé à nos travaux.

B.1 Participants

- BACLE Clément,
- BENZERZOUR Mohammed,
- BERNARD Julien,
- BOIDOT Julien,
- BONNEAUD Fréderic,
- BROSSEAU Céline,
- CHAMPION Jean-Baptiste,
- DAVIET Géraldine,
- DECHATELPERRON Arthur,
- DREVEAU Fanny,
- GALOPIN Julie,
- GUILLEMAUT Caroline,
- GUILLEMOT Brendan,
- HECKLER Yann,
- HOUPERT Sylvain,
- HUBET Laure,
- JACHET Isabelle,
- LAUBY Thierry,
- LECLERC Perrine,
- MAUGET Stéphane,
- NEDJAR Rezak,
- OGER Céline,
- PARNAUDEAU Nicolas,
- RAIMBAULT Manon,
- RIVALIN Jean,
- ZERROUK Mustapha.

Nous remercions les personnels du laboratoire CERMA de l'École d'Architecture de NANTES pour leur aide à la réalisation de cette étude (tout spécialement Daniel SIRET), ainsi que Pierre DRAGICEVIC pour son aide lors de la réalisation des divers logiciels utilisés, ainsi que son aptitude à les faire « planter » lors de nos tests.

B.2 Quelques dessins...

FIGURE B.1 –

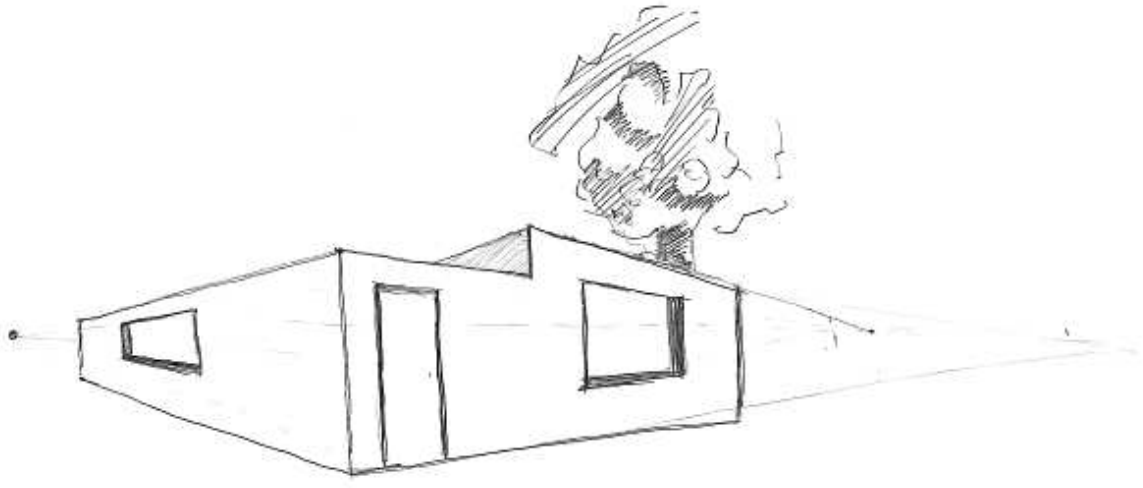


FIGURE B.2 –

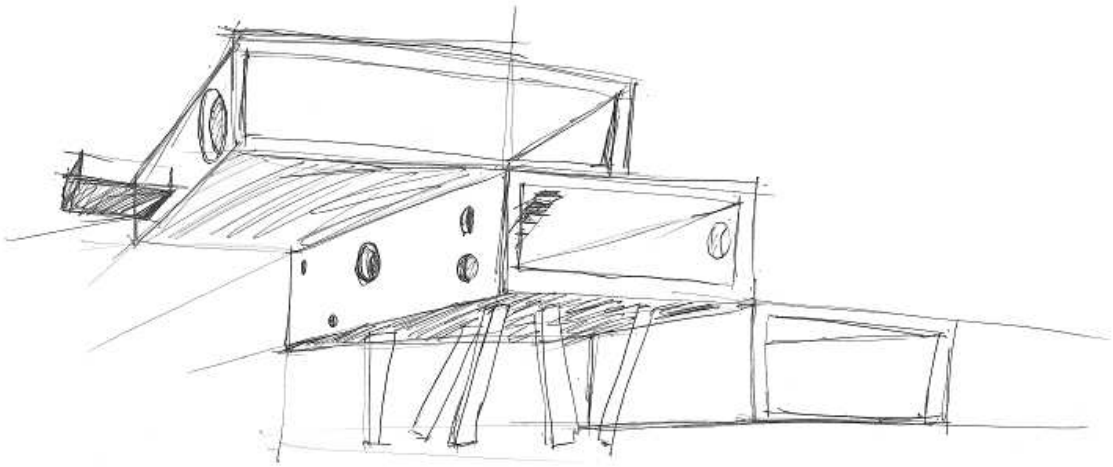


FIGURE B.3 –

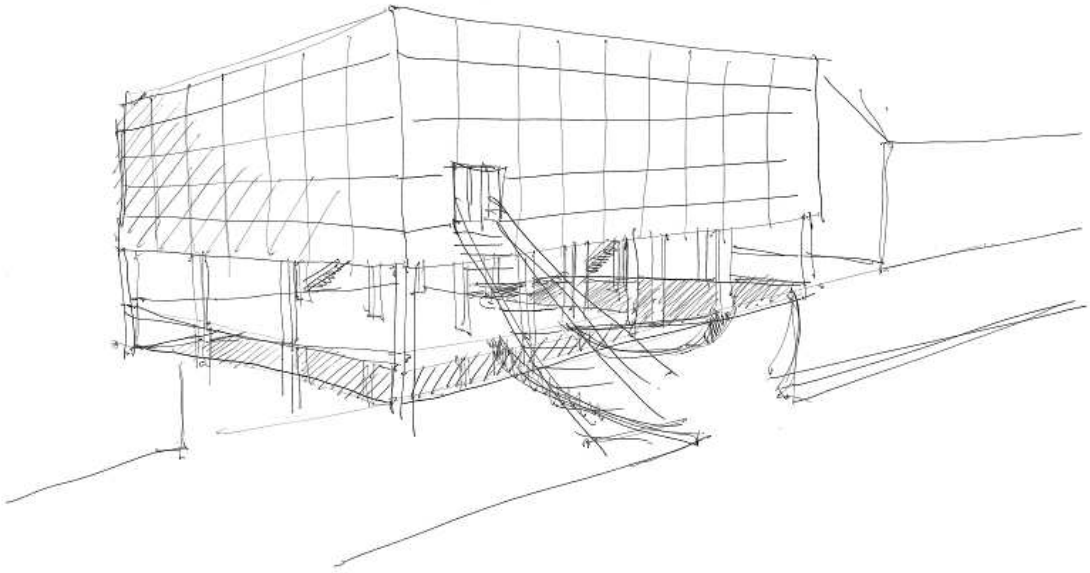


FIGURE B.4 –

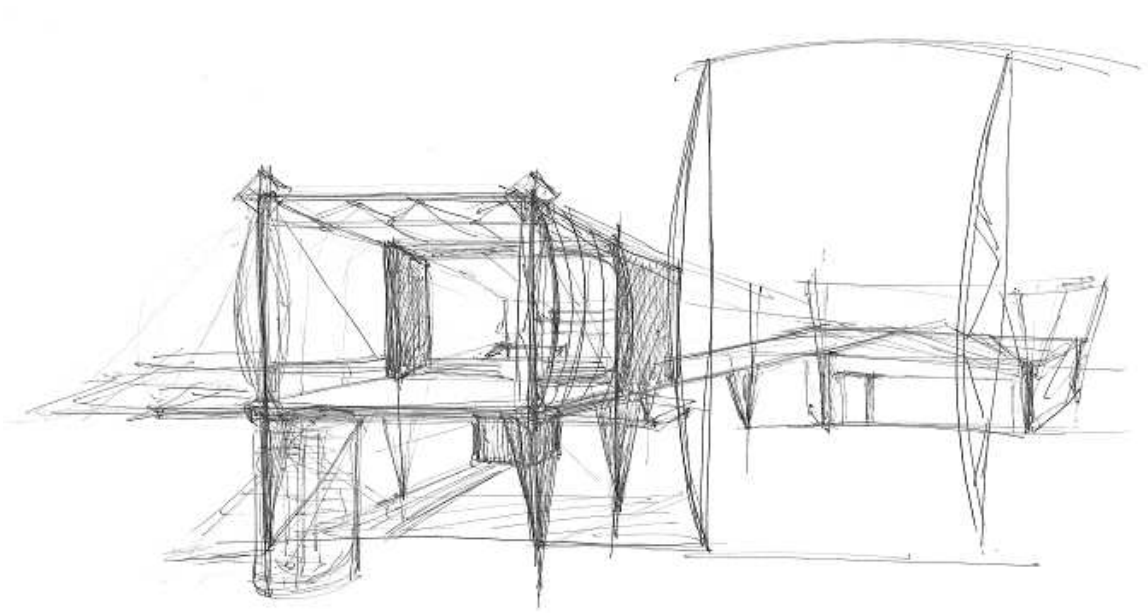


FIGURE B.5 –

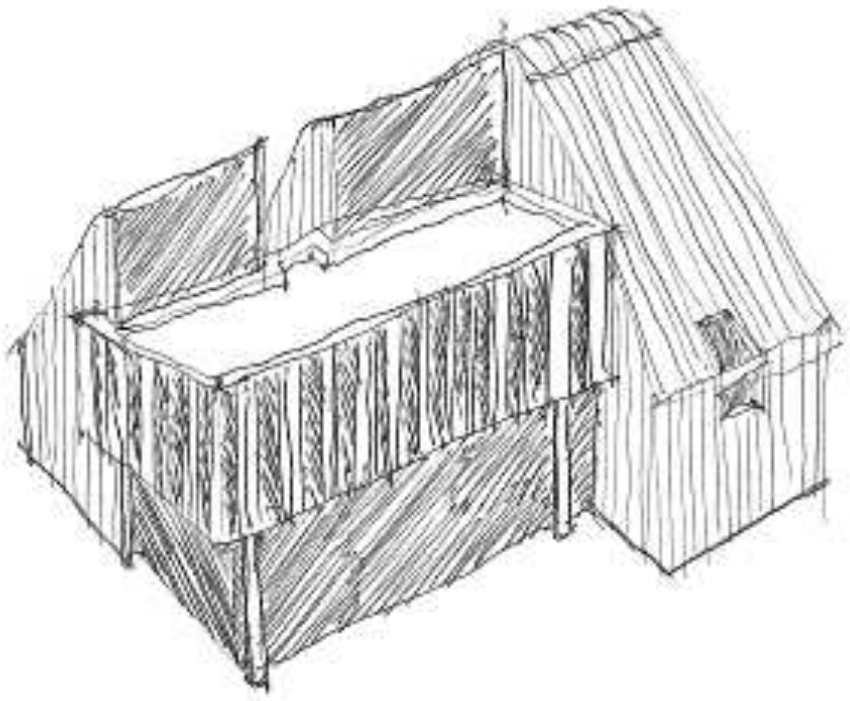


FIGURE B.6 –

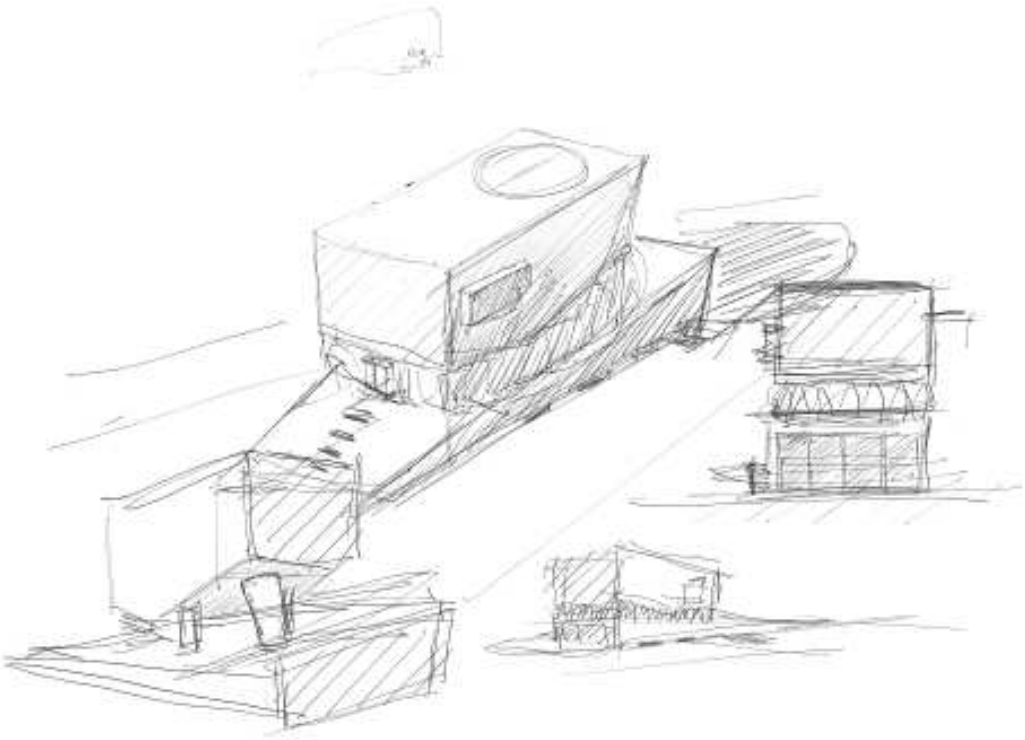


FIGURE B.7 –

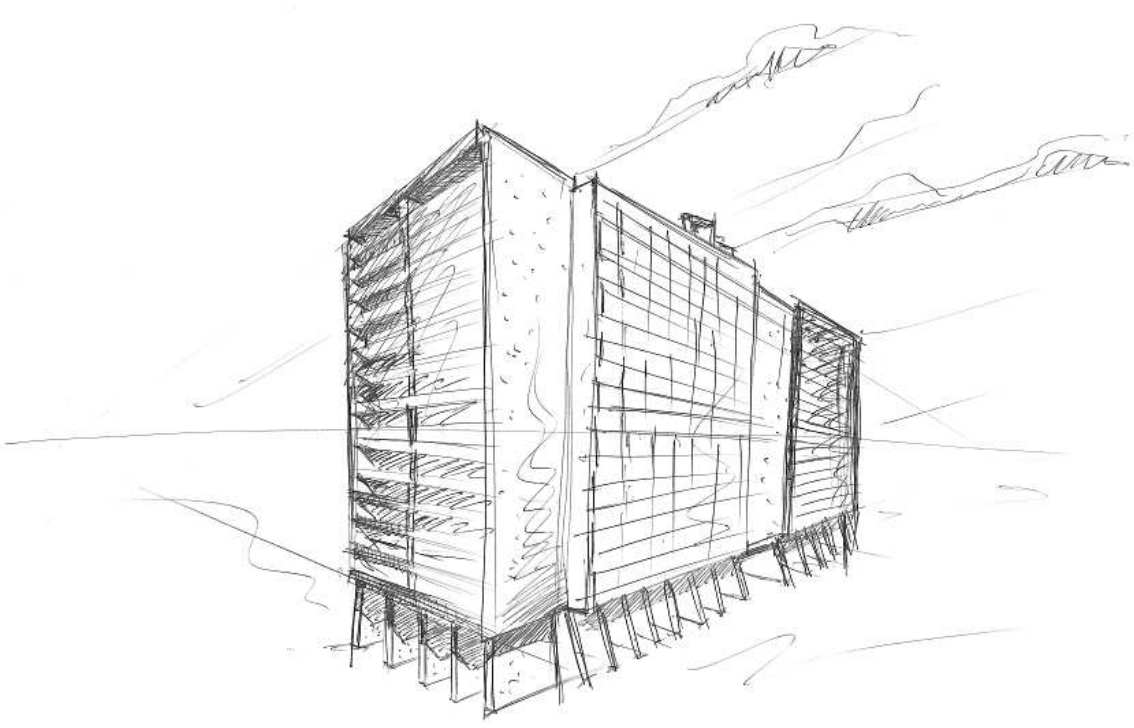


FIGURE B.8 –

Annexe C

ICON+ : Extension d'ICON

Sommaire

Cette annexe détaille ICON+, l'extension que nous avons apportée à ICON pour permettre la réalisation du modèle des graphes combinés dans la boîte à outils MAGGLITE. Cette extension porte sur trois points :

1. Dans un premier temps, nous précisons la place que tiennent les Points d'accès à l'interaction dans le modèle ICOM. C'est une extension du **modèle des dispositifs**.
2. Ensuite, nous présentons les ajouts que nous avons proposés dans la **librairie des dispositifs**, en particulier pour une gestion unifiée des dispositifs système et l'introduction de nouveaux dispositifs pour l'interaction.
3. Enfin, nous introduisons des règles de programmation et des mécanismes basés sur les propriétés d'*introspection* du langage Java pour la **génération automatique des dispositifs** manipulateurs de MAGGLITE.

C.1 Extension du modèle des dispositifs

Comme nous l'avions déjà évoqué dans le chapitre 6, le modèle ICOM définit quatre niveaux de contrôle des applications et des boîtes à outils graphique (voir figure C.1) :

1. le **contrôle par défaut**, niveau de contrôle d'une application « sans ICON » ;
2. le **contrôle générique de surface**, au plus haut niveau des widgets d'une boîte à outils ;
3. le **contrôle générique de modèle**, spécifique à un type de widget, court-circuitant les méthodes d'entrée de la boîte à outils ;
4. le **contrôle dédié**, la stratégie la plus puissante décrivant le contrôle complet d'une application par des *dispositif d'application* dédiés.

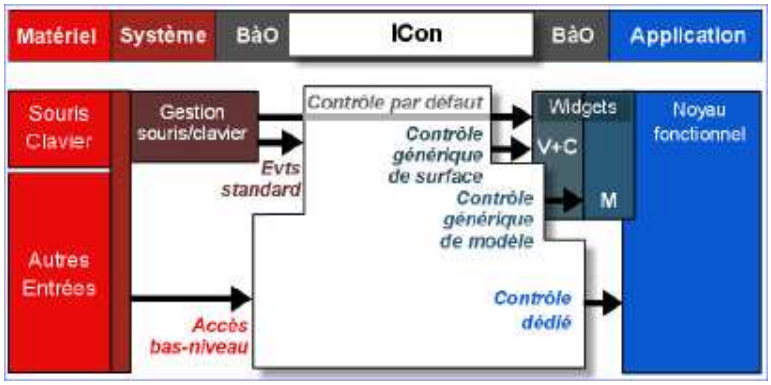


FIGURE C.1 – Niveaux de contrôle d'un système interactif avec ICON.

On peut donc définir deux grandes tendances pour réaliser une application avec ICON :

1. concevoir une application avec une boîte à outils graphique, ou en modifier une existante. Il faut alors utiliser des *dispositifs de boîte à outils graphique*, définis comme des dispositifs « situés à mi-chemin entre la catégorie des dispositifs utilitaires et celle des dispositifs d'application, (ils) permettent un contrôle générique de ces dernières » [Dragicevic et Fekete, 2004]. Le niveau de contrôle est alors générique, de surface ou de modèle. Des exemples de telles applications ont étés fournis par Pierre DRAGICEVIC en utilisant les boîtes à outils graphiques *Swing* ou *Jazz*.

2. concevoir une application originale, à partir d'un environnement graphique de bas niveau (Java2D, par exemple) et externaliser les outils, les tâches de cette application sous forme de *dispositifs d'application*. Le contrôle est alors au niveau le plus profond, il est dédié.

Le premier point facilite le développement de l'application, en permettant l'utilisation d'une boîte à outils graphique. Par contre, il implique des concessions au niveau du contrôle de ses entrées et de ses interactions, ainsi qu'un lourd travail de développement de dispositifs si la boîte à outils utilisée n'est pas encore supportée par ICON.
Le second point offre un niveau maximal de contrôle, permettant au développeur de rendre contrôlables et configurables toutes les interactions de l'application par l'intermédiaire des configurations d'entrée. Toutefois, cela demande un gros travail de programmation au niveau de l'application (pas ou peu d'abstractions et de composants des boîtes à outils graphique) mais surtout une bonne connaissance de la conception de dispositifs ICON.

Grâce au modèle des graphes combinés et de sa réalisation dans MAGGLITE, nous proposons une autre alternative : utiliser une boîte à outils graphique dont les applications sont entièrement contrôlées par ICON, avec les *points d'accès à l'interaction*. Ce sont donc des dispositifs d'application, mais génériques. Nous étendons cette notion et l'intégrons dans le modèle ICOM par la définition suivante :

Définition C.1 *Les* dispositifs génériques d'application *forment une sous-classe des dispositifs d'application du modèle ICOM. Ce sont des dispositifs d'application réutilisables, permettant le contrôle dédié d'un type d'applications donné.*

De tels dispositifs permettent donc d'atteindre le niveau de contrôle maximal (*contrôle dédié*) dans des applications, en utilisant des dispositifs de boîte à outils (jusqu'alors limités aux contrôles génériques de surface et de modèle). Leur première réalisation dans la boîte à outils MAGGLITE offre une granularité encore plus fine au modèle ICOM, permettant de séparer à un niveau encore plus bas les interactions du noyau fonctionnel de l'application.

Du point de vue du développeur, cette séparation en deux niveaux des dispositifs d'application implique il est vrai une certaine réflexion dès lors qu'il doit réaliser des fonctionnalités particulières en utilisant MAGGLITE :

1. Est-il possible à partir des dispositifs déjà disponibles de réaliser ces fonctionnalités en composant une *configuration d'entrée avancée* ?
   Si tel est le cas, il n'est pas nécessaire d'avoir recours à la programmation.
2. Sinon, ces fonctionnalités peuvent-elles être séparées de l'application pour laquelle elles ont été pensées ?
   Si tel est le cas, elles peuvent être introduites dans la boîte à outils comme *dispositifs génériques d'application* sous différentes formes :
   (a) un **dispositif d'interaction**, dans le cas d'une technique d'interaction ;
   (b) une nouvelle classe d'objet graphique, et donc son **manipulateur** associé ;
   (c) un **dispositif de comportement** ;
   (d) ou un **outil interne**.
   Ces dispositifs deviennent alors réutilisables pour toutes les applications.
3. enfin, si ce sont vraiment des fonctionnalités propres à l'application, elles doivent être implémentées lors de la réalisation de l'application, et rendues accessibles par des *dispositifs d'application* (ou par l'intermédiaire du dispositif de l'application que propose le modèle des graphes combiné).

Pour le concepteur d'interfaces, cette notion de dispositifs génériques de l'application offre un grand pouvoir d'expression. Il peut en effet très rapidement prototyper de nombreux styles d'interaction en conjuguant les dispositifs disponibles de plusieurs manières possibles. Il peut ainsi explorer divers paradigmes pour une même interface : gestes, reconnaissance vocale, mais aussi instruments en association directe avec des composants graphiques, etc.

C.2 Extension de la librairie de dispositifs d'ICON

Nous avons aussi, afin d'augmenter encore les possibilités d'ICON et donc de MAGGLITE, développé de nouveaux dispositifs offrant à la fois une meilleure portabilité de la boîte à outils mais aussi des possibilité pour l'exploration d'autres méthodes d'interaction.

C.2.1 Dispositifs système

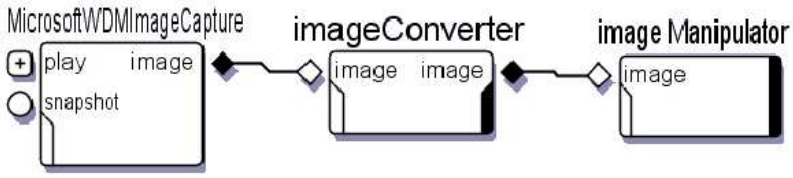
Nous avons introduit dans ICON une gestion des périphériques d'entrée basée sur la librairie java *Jinput* [Sun Microsystems, Inc, 2004a] afin de proposer un support unifié des périphériques sous plusieurs systèmes d'exploitation (Windows, Linux et MacOS). La première version de la boîte à outils proposait en effet des accès bas-niveau aux périphériques par des API souvent dépendantes du système d'exploitation, limitant alors la portabilité de la boîte à outil, mais aussi des configuration d'entrée.
Jinput propose un accès unifié à un grand nombre de périphériques d'entrée, en déléguant la gestion du bas-niveau à des greffons. Ainsi, sur une même configuration matérielle, les mêmes dispositifs d'entrée sont disponibles, quel que soit le système d'exploitation, résolvant en partie le problème de l'hétérogénéité des APIs d'entrée soulevé par la première version d'ICON.

Nous avons aussi complété la librairie d'ICON par des dispositif d'acquisition d'image, basés sur la librairie *Java Media Framework* [Sun Microsystems, Inc, 2005]. Cet ensemble de dispositifs permet l'accès bas-niveau à des périphériques tels que des *webcams* ou des cartes d'acquisition vidéo connectées à la machine hôte. Ces dispositifs (voir figure C.2 page suivante) permettent l'émission d'images dans des configuration d'entrée, sous la forme de signaux ponctuels à déclenchement contrôlé par un événement (capture d'une image) ou en flux continu (capture video).

Des dispositifs adaptateurs asynchrones dédiés au traitement des images peuvent être réalisés à partir d'un squelette de dispositif. Celui-ci permet d'externaliser les algorithmes de traitement d'images à l'aide du principe des entrées/sorties implicites, assurant que ces dispositifs coûteux en ressources ne pénaliserons pas l'exécution réactive de la configuration d'entrée (la thèse de Pierre DRAGICEVIC donne plus de détails sur la réalisation de dispositifs à haut débit d'informations).
Parmi les adaptateurs, nous proposons un dispositif de conversion d'images et un *filtrage des couleurs*. Le premier permet la conversion vers différents types d'images (pour l'affichage, par exemple, comme dans la figure C.2(b) page ci-contre). Le second peut permettre le contrôle d'un curseur par suivi de gestes(1), comme présenté dans la figure C.2(c) page suivante.

Enfin, un nouveau dispositif système permet de jouer des fichiers audio (en complément de celui

(1)Étant donnée la simplicité de cet algorithme, cette configuration nécessite un environnement adapté pour être fiable. Pour contrôler par exemple le curseur par des mouvement de la main ou de la tête, une couleur de fond très contrastée par rapport à la couleur de la peau améliorera considérablement les performances.



(a) Configuration d'entrée.

(b) Affichage en temps réel.

(c) Application temps réel d'algorithmes.

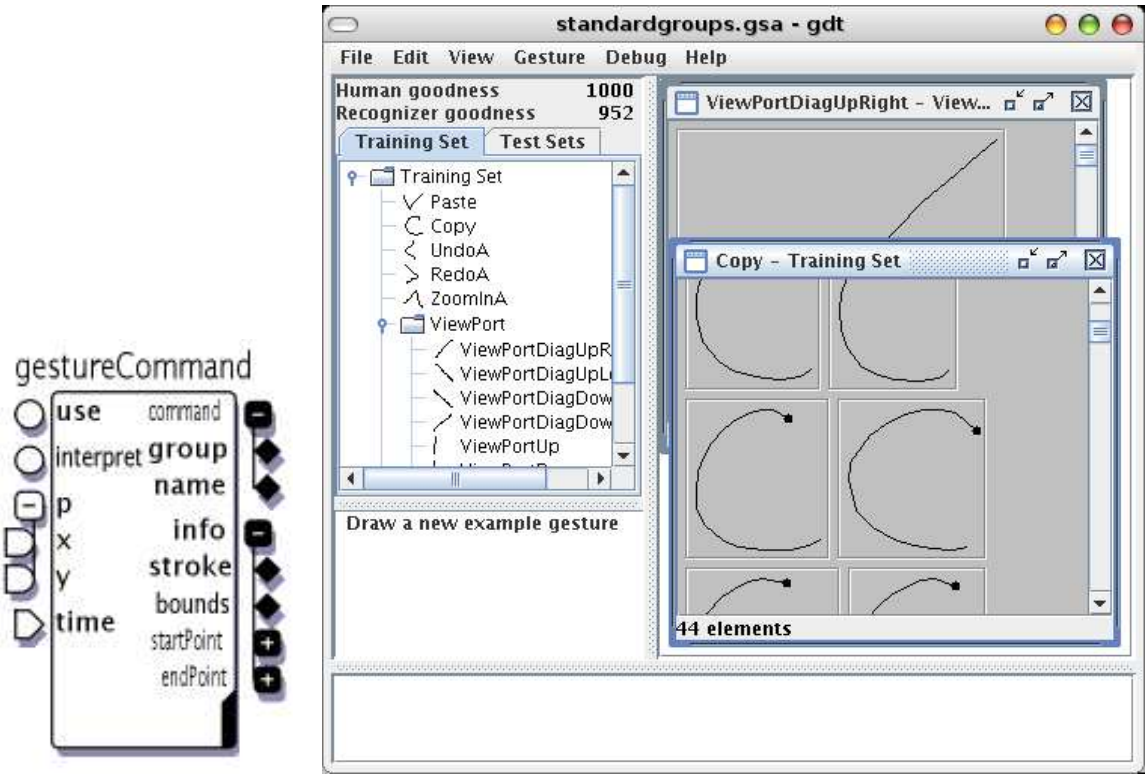
FIGURE C.2 – Dispositif de capture d'images. La figure (a) montre l'utilisation d'un dispositif de capture video dans une configuration d'entrée. Celui-ci est connecté en entrée d'un dispositif adaptateur, lui-même connecté au dispositif manipulateur d'un conteneur d'images. Le flux vidéo est alors affiché dans l'interface (figure (b)). Dans la figure (c), un adaptateur permettant le filtrage de couleurs a été inséré dans les traitements. Il peut par exemple être utilisé pour manipuler un curseur.

d'origine qui n'utilisait que les périphérique de sortie de type *Midi*). Il permet de configurer simplement un retour sonore dans les interfaces, en spécifiant dans la configuration d'entrée l'action qui va déclencher le son.

C.2.2 Techniques d'interaction et dispositifs spéciaux

Nous avons aussi complété la librairie des dispositifs par des techniques d'interaction avancées telles que la reconnaissance de gestes, les marking menus, ou des dispositifs permettant d'enregistrer des flot de données.

Interpréteur de gestes
Le dispositif de reconnaissance de gestes que nous avons développé dans ICON facilite la réalisation d'interfaces gestuelles (voir figure C.3(a)). Il utilise les interpréteurs de la bibliothèque Satin [Hong et Landay, 2000], ainsi que l'outil Quill [Allan Christian Long, Jr et Landay, 1999; Allan Christian Long, Jr, 2001] pour la création et l'édition interactive des gestes.



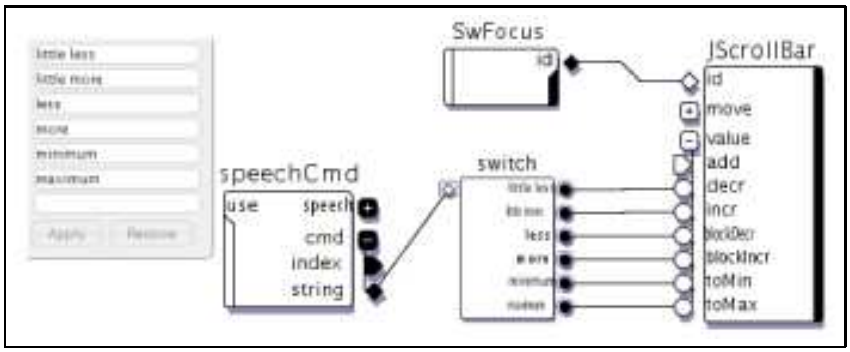
(a) (b)

FIGURE C.3 – Dispositif de reconnaissance de gestes et l'outil de création/édition de gestes Quill [Allan Christian Long, Jr, 2001].

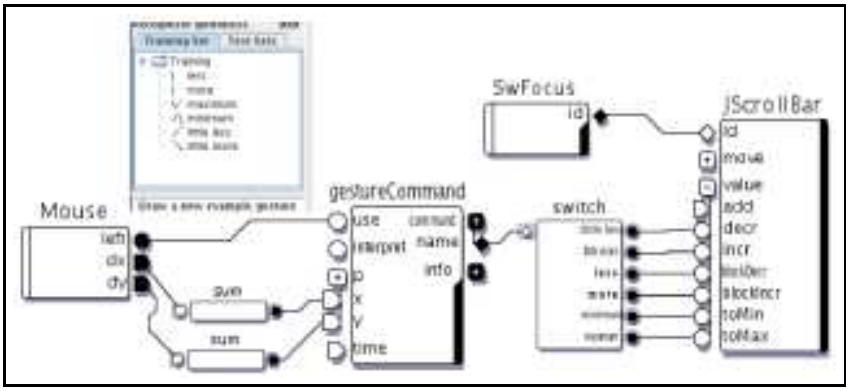
À partir des entrées reçues (position et activation), le dispositif analyse la marque tracée conformément aux gestes définis. Il émet alors le nom et le groupe (les gestes peuvent être regroupés par

catégories) du geste reconnu, ou la commande « unrecognized », et les informations géométriques sur le geste tracé (sa boîte englobante et ses points de départ et d'arrivé). Ainsi, de nombreuses configurations avancées peuvent êtres construites autour de ce dispositif, de la simple commande gestuelle, jusqu'à la sélection d'objets comme nous l'avons fait pour la saisie des contraintes dans SVALABARD (voir les sections 4.3.2 page 98 et 8.4.3 page 204).

Ce dispositif est interchangeable avec d'autres dispositifs de commande (clavier, reconnaissance vocale, etc.), permettant ainsi d'ajouter facilement de la reconnaissance de geste dans une application, même si celle-ci n'était pas à l'origine prévue pour.



(a) Contrôle vocal des barres de défilement de Swing.



(b) Contrôle gestuel des barres de défilement de Swing.

FIGURE C.4 – Interchangeabilité des dispositifs de commande. Le dispositif de reconnaissance vocale de la figure (a) est simplement remplacé par le dispositif de commande gestuelle dans la figure (b).

Par exemple Pierre DRAGICEVIC avait utilisé le contrôle vocal pour manipuler une barre de défilement dans une application Swing (voir figure C.4(a)). Le dispositif de reconnaissance vocale émet les commandes reconnues (plus, un peu moins, minimum, etc.) qui sont alors envoyées au dispositif de contrôle conditionnel *switch* (qui active une commande en fonction de la chaîne de caractères reconnue). En remplaçant le dispositif de reconnaissance vocale par celui de reconnaissance de gestes, en ayant au préalable défini un vocabulaire de gestes compatible (voir figure C.4(b)), il est alors possible de manipuler la barre de défilement à l'aide de gestes. Des configurations plus complexes pourraient

être construites en utilisant les informations sur la marque tracée pour déterminer l'objet d'intérêt (sélection de la barre de défilement) et le taux de défilement, par exemple.

Marking Menus

Nous avons déjà présenté et discuté l'emploi de ces menus dans la description des interaction de SVALABARD, page 105. Rappelons simplement qu'ils combinent menus circulaires (« pie menus ») et gestes rectilignes simples afin de sélectionner l'entrée désirée. Dans [Kurtenbach et Buxton, 1994], les auteurs ont montrés les avantages qu'ils procurent par rapport à des menus linéaires (mémorisation musculaire, différents niveaux d'expertise) et leurs apports dans les interfaces à stylet (cohérence, rapidité et apprentissage).

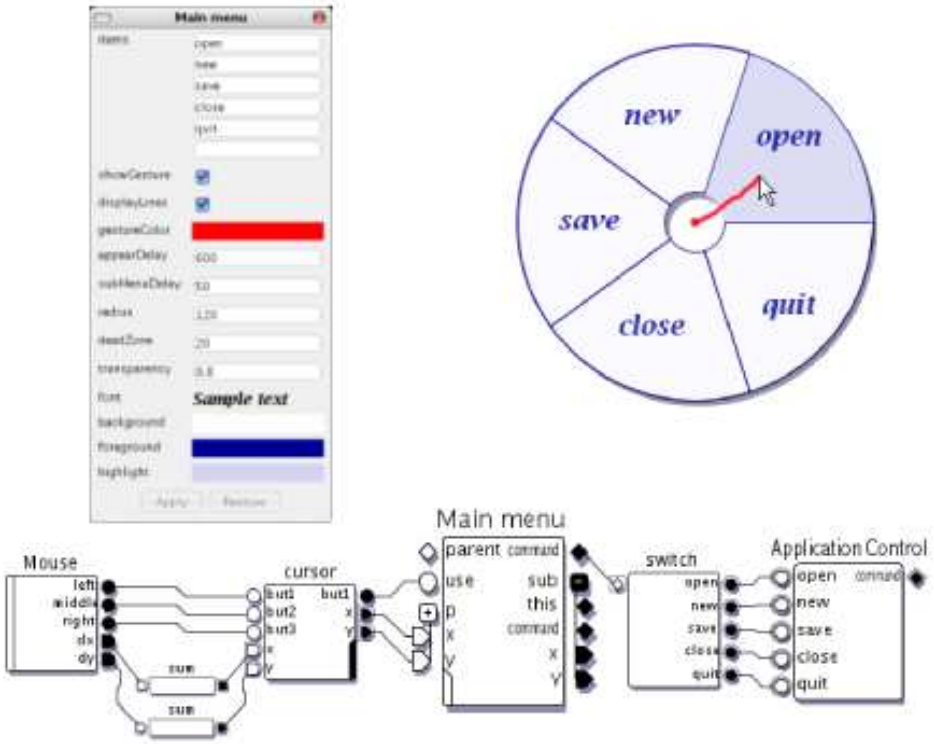


FIGURE C.5 – Dispositif Marking Menu. Le dispositif de Marking Menu est paramétrable (graphismes et comportement) et peut être utilisé dans toutes les applications.

L'implémentation des Marking Menu que nous proposons en fait une technique générique, sous la forme d'un dispositif. Dans la figure C.5, nous avons représenté ce dispositif connecté dans une configuration, sa boîte de paramètres et son retour graphique. Le menu est contrôlé par une souris et activé par le bouton gauche. Il faut ensuite configurer quelques paramètres, en particulier les items du menu : ceux-ci sont saisis dans des zones de textes de la boîte de paramètres. Le retour graphique du menu sera créé selon ces items, dont nous avons limité le nombre à sept [(2)].

[(2)]Cette limitation à est due au fait que l'aspect des menus serait alors trop confus pour un plus grand nombre d'items, ce qui les rendrait mal arrangés. Mais c'est aussi une conséquence de l'assertion de Georges A. MILLER sur le fait que sept items (plus ou moins deux) est le nombre optimal pouvant être assimilés par une personne *lambda* [Miller, 1956]

Les autres paramètres du dispositif sont les options graphiques (taille, couleurs, opacité, tracés et police) et les options de comportement du menu : délai avant l'apparition du menu ou de ses sous-menus, taille de la zone centrale. Le menu peut alors être utilisé dans la configuration : un clic sur le bouton de la souris permet de l'activer. Au bout du temps paramétré, le menu apparaît et permet de sélectionner l'item voulu, validé par le relâchement du bouton (un retour en zone centrale annule l'action). Si un item a été choisi, son nom est alors émis par le slot de sortie *command* du dispositif. Évidemment, si le geste est terminé avant le délai d'apparition du menu, le comportement est le même (envoi du nom de l'item correspondant au geste). Ce slot de sortie de commande peut être connecté à un dispositif *switch*, qui converti la chaîne de caractère reçue en une commande pour l'envoyer à un dispositif de contrôle (application ou outil), comme dans la figure C.5 page ci-contre.

Enfin, plusieurs de ces dispositifs peuvent être composés afin de créer des cascades de sous-menus. Il faut pour cela utiliser les slot de sortie *sub* du dispositif. Le slot *sub.this* d'un menu de niveau N permet de le connecter au slot d'entrée *parent* de son sous-menu de niveau N+1. Ensuite, les slots de sortie *sub.command*, *sub.x* et *sub.y* permettent le contrôle du sous-menu.

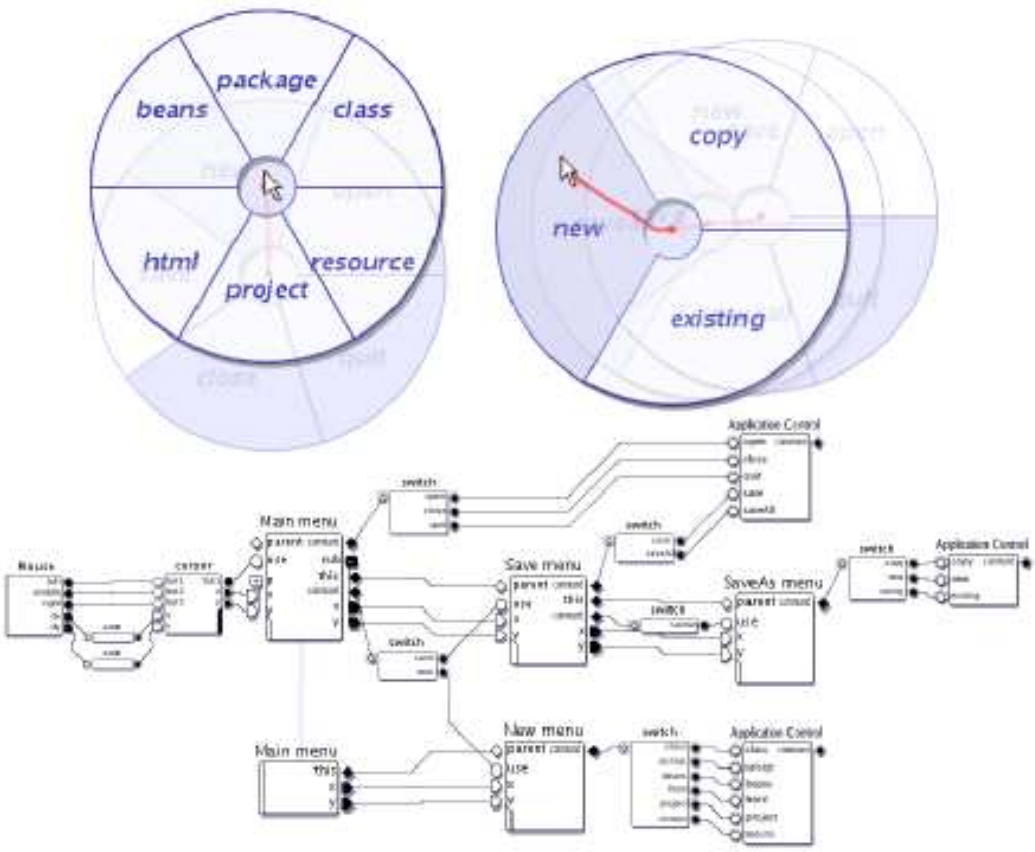


FIGURE C.6 – Dispositifs de Marking Menu en cascade. Les dispositifs peuvent être associés dans une même configuration afin de former des sous-menus en cascade. En haut à gauche, le sous-menu *New* du menu principal ; en haut à droite, le sous-menu *SaveAs* du sous-menu *Save* du menu principal.

La figure C.6 illustre de telles associations. Il y a tout d'abord un menu principal, le dispositif *Main menu*. Celui-ci propose les commandes « open », « new », « save », « close » et « quit ». La sortie

principale (*command*) de ce dispositif est connectée à un dispositif Switch qui va être assigné au traitement des items qui ne nécessitent pas de sous-menu : « open », « close » et « quit ». Par contre, les items « new » et « save » nécessitent un sous-menu : la sortie *sub.command* va donc être connectée à un autre dispositif Switch qui va traiter les commandes « new » et « save » afin de les repartir sur les dispositifs de sous-menu. La commande « open » est transmise au sous-menu *New menu*, la commande « save » au sous-menu *Save menu*.
Le sous-menu *New menu* propose alors six items : « class », « package », « beans », etc. Ces sorties sont elles-mêmes connectées à un dispositif Switch qui va traiter les commandes émises par ce menu. Dès lors, lorsque l'utilisateur choisis l'item « new » du menu principal, le sous-menu *New menu* apparait et propose six nouveaux items. Il en est de même pour l'item « save » qui déclenche le sous-menu *Save menu*, pouvant lui aussi déclencher le sous-menu *SaveAs menu*.

Nous accordons qu'une telle configuration peut paraître assez complexe, mais elle obéit à une simple logique d'aiguillage des signaux, propre à l'architecture en flot de données qui est assez vite assimilée de par sa logique. Bien que l'utilisateur final puisse modifier les configurations, cela reste tout de même plus le travail du concepteur de l'interface. Nos dispositifs lui permettront de créer très rapidement des *Marking menu*, de les tester, les raffiner et les réorganiser sans même avoir à écrire une seule ligne de code. Et il est encore une fois évident que des configurations alternatives peuvent être proposées (reconnaissance vocale, gestes, etc.).

Enregistreur et utilisateur virtuel

Enfin, les dispositifs que nous décrivons maintenant ont étés conçus tout particulièrement pour la pratique d'études utilisateurs. Ils permettent d'enregistrer et de re-émettre les flux traversant une configuration d'entrée, en n'importe quel nœud de celle-ci.

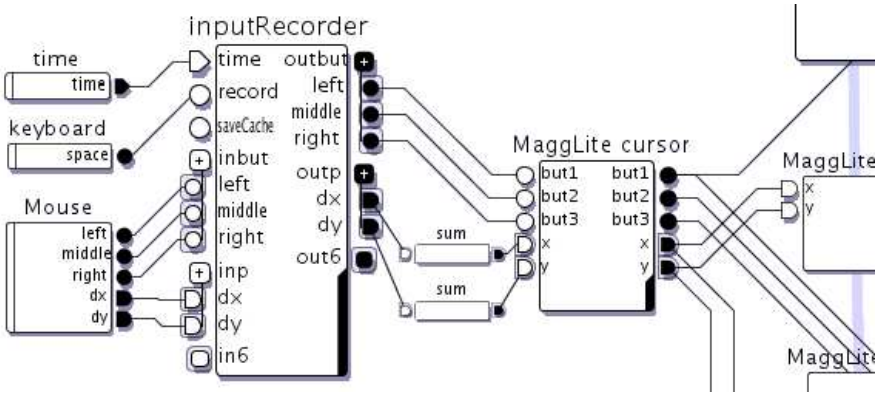


FIGURE C.7 – Dispositif d'enregistrement. Le dispositif *inputRecorder*, intercalé à n'importe quel niveau d'une configuration, permet d'enregistrer les valeurs reçues en entrée dans un fichier XML. Ce fichier pourra être « rejoué » par le dispositif *virtualUser* (figure C.8 page suivante).

Le premier, nommé *inputRecorder*, est un dispositif d'enregistrement que l'on peut intercaler à n'importe quel point d'une configuration d'entrée. Par exemple, dans la figure C.7, il se situe entre le dispositif souris et les adaptateurs qui lui sont normalement connectés pour propager ces valeurs dans le reste de la configuration. Lorsque la configuration est lancée, le dispositif d'enregistrement accumule toutes les valeurs qu'il reçoit des dispositifs qui lui sont connectés, la souris dans notre

exemple. Bien entendu, il ne bloque pas ces valeur et les transmet par ses slots de sortie. Lorsque la configuration d'entrée est stoppée, le dispositif sérialise ces données accumulées dans un fichier XML.

Dans cet exemple, les données enregistrées sont reçues directement d'un périphérique d'entrée afin d'enregistrer tous ses déplacements et actions. Il serait évidemment possible d'enregistrer les sorties d'autres dispositifs, en utilisant même plusieurs enregistreurs dans une configuration.

Les données recueillies pourront ensuite être analysées pour des mesures de performance ou un simple suivi des actions réalisées. Mais il est aussi possible de « rejouer » les actions qui se sont déroulées en utilisant le dispositif *virtualUser*.,Il permet de re-émettre des données enregistrées dans une configuration d'entrée. Il peut être connecté dans la même configuration d'entrée, en lieu et place du dispositif d'enregistrement (voir figure C.8) et de tous les dispositifs qui lui étaient connectés afin de simplement « rejouer » les actions enregistrées. Il peut aussi être connecté dans une configuration alternative, comportant des dispositifs d'analyse, ou même d'autres interactions compatibles avec les données enregistrées.

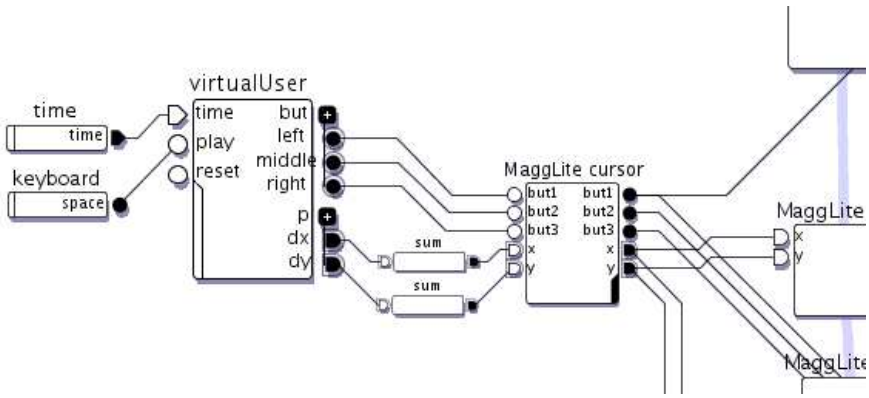


FIGURE C.8 – Dispositif d'entrées virtuelles. Le dispositif *virtualUser* permet de rejouer des fichiers enregistrés par le dispositif *inputRecorder*. Il suffit de le replacer dans la même configuration d'entrée afin qu'il émette les valeurs enregistrées.

Ces dispositifs peuvent aussi cohabiter dans une même configuration. Il est alors possible de définir un tampon d'échange entre eux (à l'aide des paramètres des dispositifs). Ce tampon peut servir, par exemple, à l'enregistrement d'actions répétitives qui pourront alors être rappelées dans le même contexte et lors de la même tâche, dans l'esprit de macros. Il serait alors envisageable d'étudier les possibilités de telles configurations dans le cadre de paradigmes de programmation par démonstration.

C.3 Génération de dispositifs d'application « à la volée »

La création dynamique de dispositifs n'est pas nouvelle dans ICON, en particulier pour les dispositifs système. En effet, la gestion de bon nombre des périphériques d'entrée est activée à l'exécution, et la création des dispositifs prototypes est alors dynamique.

Nous avons généralisé ce principe et ces mécanismes aux dispositifs génériques d'application, en particulier pour les manipulateurs d'objets graphiques de MAGGLITE (voir la section 7.4.2 page 168).

Ces dispositifs, rappelons-le, sont des dispositifs d'instance (liés à un objet graphique particulier) qui permettent un contrôle direct d'un nœud du graphe de scène, externalisant alors les propriétés de cet objet dans le graphe d'interaction. Donc, lorsqu'un nouveau nœud est ajouté au graphe de scène, celui-ci peut, ou non, installer son *manipulateur* dans la librairie des dispositifs disponibles.

Le principe des manipulateurs, d'un point de vue technique, est d'associer un slot du manipulateur à un méthode de l'objet que l'on souhaite externaliser. Il aurait été fastidieux d'avoir à développer une classe de dispositifs ICON à chaque création d'une nouvelle classe d'objet graphique. Nous proposons donc un mécanisme utilisant les capacités d'introspection (réflection) du langage Java afin d'obtenir les méthodes de l'objet correspondantes aux slots à créer, de les invoquer sur l'objet lorsque le dispositif reçoit les signaux correspondants, et de les décrire à l'aide des annotations (metadata) spécifique à Java 5. Il suffit, lors du développement d'une nouvelle classe de nœuds du graphe de scène, de respecter les conventions de nommage que nous avons fixées afin que la boîte à outil crée le dispositif correspondant aux méthodes que nous avons nommées *xmethods*.

La figure C.9 présente sur un exemple les méthodes d'un objet associées à son dispositif. La méthode `xcomVisible()` est externalisée sous la forme d'un slot d'entrée booléen nommé *visible*, la méthode `xsetMove(double dx, double dy)` sous la forme de deux slots d'entrée de type *double* nommés *move.dx* et *move.dy* et enfin la méthode `Object xgetId()` sous la forme d'un slot de sortie de type *Object*.

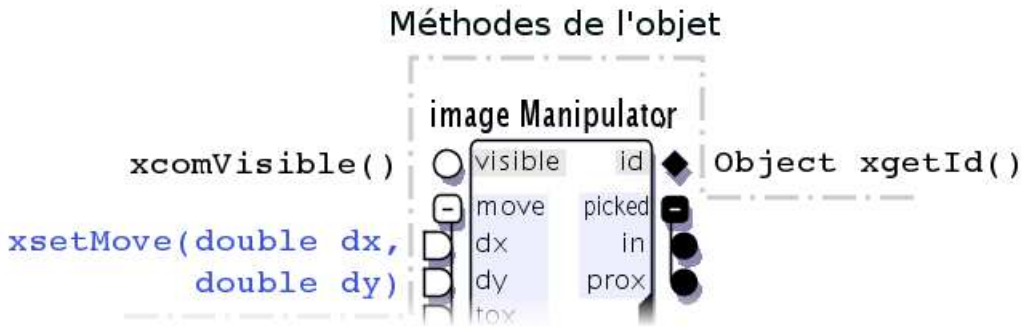


FIGURE C.9 – Correspondance objet/dispositif. Le *dispositif d'application* (un manipulateur de la boîte à outils MAGGLITE) présente des slots associés aux méthodes du composant graphique associé.

Cet exemple introduit trois catégories de xmethods :

1. les **méthodes de commande**, déclarées sous la forme `xcomNom()` . La sémantique associée à ces méthodes est le déclenchement d'une action ou d'un comportement spécifique à l'objet. Elles ne prennent donc pas de paramètres en entrée et ne retournent aucune valeur. Elles seront présentées sur le dispositif sous la forme d'un slot booléen.
2. les **méthodes d'état**, déclarées sous la forme `xsetNom(type p[, type...])` . Ces méthodes représentent des modificateurs d'états ou de propriétés de l'objet. Elle prennent donc autant de paramètres que nécessaire (de types compatibles avec le typage des slots de ICON) et n'ont pas de valeur de retour. Un slot d'entrée est donc créé pour chaque paramètre de la méthode, et ceux-ci sont groupés hiérarchiquement dans un groupe de slots portant le nom de la méthode.
3. les **méthodes d'émission**, déclarées sous la forme `type xgetNom()` . Ces méthodes sont

des accesseurs à des valeurs d'état ou d'attribut de l'objet, qui ne prennent pas de paramètres, mais doivent retourner une valeur (de type compatible avec le typage des slots de ICON). Ainsi, un slot de sortie est créé sur le dispositif.

Les méthodes de commande et d'état sont invoquées lorsque le ou les slots associés reçoivent un signal, les méthodes d'émission sont invoquées à chaque pas de l'exécution. Toutefois, ces appels aux méthodes de l'objet pose un problème de stratégie à employer dans le cas des méthodes d'état et dans le cas des méthodes d'émission.

Pour les méthodes d'état, il est clair qu'elle ne peuvent être invoquées si tous les paramètres ne sont pas définis, c'est à dire si tous les slots d'entrée correspondant du dispositif ne sont pas connectés, et n'ont pas tous reçu au moins une fois un signal. Mais lorsque tous les slots (et donc les paramètres de la méthode) ont une valeur, doit-on invoquer la méthode si tous les slots n'ont pas reçu de signal (et donc une potentielle nouvelle valeur) ?
Pour les méthodes d'émission, le problème est inverse. La méthode de l'objet est invoquée à chaque pas de l'exécution. Mais le dispositif doit-il émettre la valeur de retour de cette méthode à chaque pas, même si celle-ci n'a pas changée ?

À ces problèmes de stratégie, s'ajoute aussi le problème du nommage des slots, en particulier dans le cas des méthodes d'état. En effet, les mécanismes d'introspection de Java nomment les paramètres des méthodes en utilisant des caractères alphanumériques, quel que soit le nom qui leur avait été donné dans le code source. Ainsi, les slots correspondant à une méthode prenant trois paramètres seraient nommé *a*, *b* et *c*. Il faut bien avouer qu'un tel nommage serait un sérieux obstacle au paradigme de programmation visuelle d'ICON. De plus, il pourrait être intéressant de regrouper les slots de certaines méthodes selon la sémantique de ces méthodes.

Nous proposons une solution unifiée à ces problèmes qui va permettre au programmeur *d'annoter* les xmethods de l'objet, et donc de spécifier les noms et groupes des slots qu'il veut voir apparaître pour la méthode, ainsi que la stratégie souhaitée pour leur invocation. Nous avons utilisé pour cela le mécanisme des *annotations* (ou metadata) de Java 5 afin de permettre la construction de *descripteurs de méthodes*. Voici un exemple de descripteur pour la méthode d'état `xsetMove` de la figure C.9 page ci-contre :

Code Java C.1 – Exemple de déclaration d'une *xmethod*. Le descripteur définit le nom de la méthode, de ses paramètres et la stratégie d'invocation.

```
1 @XMethodDescriptor(
2     name = "move",
3     paramsNames = {"dx", "dy"},
4     allSignals = false
5 )
6 public void xsetMove(double dx, double dy) {
7     ...
8 }
```

Le descripteur de cet exemple, écrit juste avant la signature de la méthode, définit le nom qui sera donné à la méthode (« move », ici), les noms de ses paramètres qui seront donc donné aux slots (préfixés par le nom de la méthode car nous somme dans le cas d'une méthode d'état) et la stratégie d'invocation de la méthode (lorsqu'au moins un des slots correspondant à la méthode a reçu un signal). Avec la valeur *Vrai*, ce descripteur *allSignals* signifierait qu'il faut que tous les slots aient reçus un signal au même pas d'exécution pour invoquer la méthode.

L'exemple ci-après présente le champ de descripteur permettant de grouper des slots associés à différentes méthodes :

Code Java C.2 – Exemple de déclaration de *xmethods*. Les descripteurs définissent les noms des méthodes et leur groupe (voir le dispositif correspondant figure C.10).

```
@XMethodDescriptor(
    name = "null",
    methodGroup = "interpolation"
)
public void xcomNull() {
  ...
}

@XMethodDescriptor(
    name = "nearestNeighbour",
    methodGroup = "interpolation"
)
public void xcomNearestNeighbour() {
  ...
}

@XMethodDescriptor(
    name = "bilinear",
    methodGroup = "interpolation"
)
public void xcomBilinear() {
  ...
}
```

Le champ *methodGroup* permet dans ce cas de préfixer le nom des méthodes avec ce nom de groupe. Cela entraîne la création d'un groupe de slots, permettant de regrouper les slots correspondants à des méthodes de même sémantique. La figure C.10 présente le résultat du code précédent.

FIGURE C.10 – Groupement de slots avec les descripteurs de méthodes. Les slots d'entrée créés lors de l'introspection sur les méthodes de l'exemple de code C.2 sont regroupés dans le groupe « interpolation » grâce à l'attribut *methodGroup*.

Globalement, l'architecture de la boîte à outil encourage la réutilisation des composants existants et l'implémentation des interfaces fournies afin de maintenir une cohérence au niveau des manipulateurs d'objets présentant des propriétés ou interfaces communes. Par exemple, tout objet graphique héritera des propriétés de l'objet de base *« MaggLite »*. De ce fait, sont manipulateur présentera automatiquement les slots d'entrée (*visible* et *label*) et de sortie (*id*) communs à tous les objets de la classe

MaggLite. Mais c'est aussi à ce niveau que peuvent être définies des xmethods de manière abstraite. Un composant conçu pour pouvoir être redimensionné implémentera une interface de la boîte à outil nommée *« Sizeable »*. Cette interface impose l'implémentation de la méthode *« xsetResize(double x, double y) »* afin que tous les manipulateurs d'objets ayant cette capacité présente les slots d'entrée correspondants, de manière unifiée. Ainsi, un concepteur d'interfaces ou un utilisateur final retrouvera toujours les mêmes slots pour des capacités identiques (voir figure C.11).

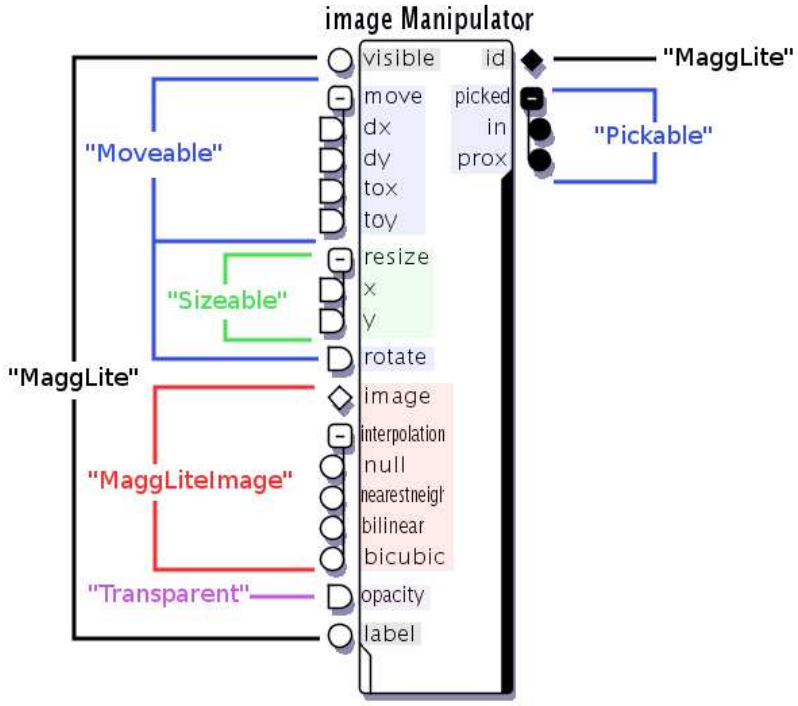


FIGURE C.11 – Détail d'un manipulateur. Ce manipulateur a été généré automatiquement pour contrôler un composant graphique conteneur d'image. Chaque classe héritée ou interface instanciée par le composant fournit des méthodes qui sont alors *externalisées* sous forme de slots d'entrée ou de sortie par le mécanisme de génération des manipulateurs lors de l'instanciation d'un objet. Les slots propres à la classe de l'objet (*MaggLiteImage*) sont désignés ici en rouge.

Ce mécanisme de génération automatique de dispositifs permet de de ne pas se soucier de la programmation des dispositifs ICON lors de la création des objets de l'application que l'on veut externaliser. La notion de *xmethods*, associées à leurs descripteurs, permet un contrôle assez poussé sur les dispositifs générés, bien que n'offrant pas encore un accès à toutes les possibilités des dispositifs ICON (slots de type variant, contrôle plus spécifique des signaux reçus ou émis, paramètres de dispositifs). Il est envisageable de rendre les descripteurs plus précis et adaptés, mais nous pensons alors qu'il serait plus efficace de développer un dispositif de manière traditionnelle (ce que nous avons fait avec les autres dispositifs génériques d'application dans MAGGLITE : Dispositifs d'interaction, Outils internes et Comportements).

Annexe D

Application des mécanismes de MAGGLITE

Sommaire

Dans cet annexe, nous proposons des exemples concrets d'instanciation et d'extension des composants de la boîte à outils MAGGLITE. Pour le composants graphiques, dispositifs d'interaction et outils internes nous présentons des exemples de code Java commentés afin de préciser les mécanismes et paradigmes de programmation sous-jacents à MAGGLITE.

D.1 Les composants graphiques

Dans le modèle des graphes combinés, les objets graphiques sont des nœuds (objets atomiques) ou des sous-graphes (objets composites) du graphe de scène. Dans le chapitre 8 page 179 sur l'implémentation de ce modèle d'architecture logicielle avec la boîte à outils MAGGLITE, nous avons déjà présenté les classes abstraites de base pour la création de nœuds (`Magglite` et `MaggLiteContainer`), ainsi que des classes concrètes pour l'instanciation ou l'extension d'objets de base. Dans cette section annexe, nous commentons des exemples de code Java afin de préciser l'utilisation de ces composants et leur extension.

D.1.1 Instanciation et utilisation des composants graphiques

Comme nous l'avons vu dans la section 8.3.1 page 183, la boîte à outils contient plusieurs classes d'objets graphiques par défaut qui implémentent les mécanismes et comportements de base du modèle des graphes combinés. Ces classes d'objets permettent déjà la création de beaucoup d'interfaces par simple instanciation et ajout au graphe de scène.
Le code D.1 présente la création de trois objets graphiques dans une application[(1)].

Code Java D.1 – Exemple d'instanciation d'objets graphiques.

```
1  // Instanciation d'un composant graphique pliable
2  MaggLiteFoldableComponent mc = new MaggLiteFoldableComponent(new Rectangle2D.
       Double(0, 0, 200, 200), "Square");
3  // Ajout du composant sur le bureau
4  desk.add(mc);
5
6  // Création d'une fenêtre rectangulaire
7  MaggLiteWindow rectangle = new MaggLiteWindow.Rectangle("Rectangle");
8  // Modification des dimensions et placement initial de l'objet
9  rectangle.setSize(200, 150);
10 rectangle.moveTo(250, 20);
11 // Masquage du dispositif manipulateur
12 rectangle.setExternalizeManipulator(false);
13 // Ajout du composant sur le bureau
14 desk.add(rectangle);
15
16 // Création d'un slider ajouté dans la fenêtre
17 MaggLiteSlider slider = new MaggLiteSlider("Slider");
18 slider.moveTo(30,30);
19 rectangle.add(slider);
```

Le premier objet de cet exemple instancie la classe `MaggLiteFoldableComponent`. Comme

[(1)] La démarche complète de création d'une application est détaillée dans la section D.4 page 58 de cette annexe.

tout composant de base de la boîte à outils, il est nécessaire de passer en argument du constructeur la forme géométrique de l'objet à créer. Il est ajouté au bureau par la méthode `desk.add(mc)`. Cette approche de création d'objets à partir d'une forme géométrique est tout a fait adaptée à la création d'interfaces innovantes, mais aussi à l'utilisation de constructeurs d'interfaces basés sur le dessin tels que celui que nous avons proposé dans la section 8.4.2 page 199.
Le second objet créé est de la classe `MaggLiteWindow`, classe abstraite qui propose trois implémentations par défaut dans la boîte à outils, selon la géométrie désiré : `Rectangle`, `Oval` ou `Star`. Cet exemple montre l'utilisation de méthodes de manipulation de l'objet (`setSize` et `moveTo`). Les objets par défaut proposent aussi des méthodes d'accès à leur propriétés particulières (contexte graphique, état, etc.) permettant de contrôler leur apparence et leurs paramètres selon leurs capacités : `setVisible`, `setColor`, `setOpacity`, etc. Cet exemple utilise aussi la méthode `setExternalizeManipulator` qui permet de rendre disponible ou non le dispositif manipulateur de l'objet dans le configurateur graphique d'interactions. Visibles par défaut, les manipulateurs d'objets graphiques peuvent être masqués à l'initialisation de l'application. Par contre, malgré le fait que ce soit techniquement possible, la modification de cette propriété est rendue impossible dès lors que l'environnement d'ICON a été lancé, afin d'éviter la suppression de dispositifs présents dans une configuration en cours d'exécution.
Enfin, le dernier objet créé, de la classe `MaggLiteSlider`, est ajouté dans le composant précédent plutôt que sur le bureau.
La figure D.1 présente le résultat visuel de ce code.

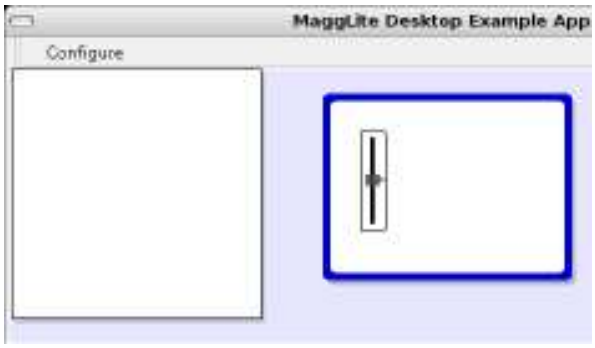


FIGURE D.1 – Objets graphiques créés par l'exemple de code D.1 page précédente.

D.1.2 Nouvelles classes de composants graphiques

Il y a deux méthodes pour introduire de nouveaux objets graphiques dans la boîte à outils. Par extension des objets déjà présents, et donc héritage et surcharge de leurs capacités, ou par composition avec de nouvelles couches, et donc la création et l'utilisation de nouveaux calques.

Par héritage

Cette première méthode est celle la plus couramment utilisée dans les boîtes à outils graphiques : par extension des classes abstraites de base ou de classes plus évoluées. L'exemple D.2 page suivante

présente une partie du code qui définit une nouvelle classe d'objets graphiques à partir de la classe abstraite de base `MaggLiteContainer`.

Code Java D.2 – Exemple de création d'une nouvelle classe d'objets graphiques : des objets de forme ovale et de couleur bleue à partir de la classe abstraite `MaggLiteContainer` et des interfaces `MaggLiteShaped` et `MaggLiteDraggable`.

```
1  public class MaggLiteBlueOval extends MaggLiteContainer implements MaggLiteShaped
       , MaggLiteDraggable {
2
3    private Shape shape;
4
5    public MaggLiteBlueOval(String name) {
6      super(name);
7      shape = new Ellipse2D.Double(0, 0, 40, 40);
8      // ajout de propriétés graphiques au contexte de l'objet
9      this.getContext().setAttribute(MaggLiteAntialiasing.id, MaggLiteAntialiasing.
           antiAliasOn);
10     this.getContext().setAttribute(MaggLiteColor.id, MaggLiteColor.Blue);
11   }
12
13 // implémentation des méthodes abstraites de la classe MaggLiteContainer
14   public void paintInside(MaggLiteGraphics2D g) {}
15   public void paintBefore(MaggLiteGraphics2D g) {
16     g.fillShape(shape);
17   }
18   public void paintAfter(MaggLiteGraphics2D g) {}
19   public void paintOnTop(MaggLiteGraphics2D g) {}
20   public void doSetSize(double width, double height) {}
21
22 // implémentation des méthodes de l'interface MaggLiteShaped
23   public Shape getShape() {
24     return shape;
25   }
26   public Shape getFlattenedShape() {...}
27   public Shape getShapeFrom(MaggLiteContainer c) {...}
28   public void setShape(Shape s) {...}
29   public void parentReshaped(Shape s) {...}
30   public Point2D getClosestPointOnShape(Point2D p) {...}
31   public Point2D getShapeCenter() {...}
32
33 // implémentation des méthodes de l'interface MaggLiteDraggable
34   public void startDrag(MaggLiteInputEvent e) {...}
35   public void endDrag(MaggLiteInputEvent e) {...}
36   public boolean isInDrag() {...}
37   public void whenDragged(MaggLiteInputEvent e) {...}
38   public boolean allowsDrag() {...}
39
40 // xmethod pour créer un slot du manipulateur permettant de controler
41 // l'anticrénelage à l'affichage de l'objet
42   @XMethodDescriptor(
43     name = "Antialiasing",
44   )
45   public void xsetAntialiasing(boolean antialias) {
46     if (antialias)
47       this.getContext().setAttribute(MaggLiteAntialiasing.id,
```

```
            MaggLiteAntialiasing.antiAliasOn);
48      else
49        this.getContext().setAttribute(MaggLiteAntialiasing.id,
              MaggLiteAntialiasing.antiAliasOff);
50      this.repaint();
51    }
52
53  }
```

Cette classe permet de construire des objets de forme ovale et de couleur bleue. Ces propriétés sont définies dans le constructeur (ligne *5*). La classe créée étend donc la classe abstraite `MaggLiteContainer` , et implémente pour cela ses méthodes abstraites (lignes *13* à *20*). Ce sont en particulier des méthodes d'affichage. Sur ce point, les lignes *9* et *10* montrent comment des propriétés graphiques sont définies dans le *contexte* de l'objet, contexte qui sera mis en place lors des opérations d'affichage de l'objet. Si certains attributs ne sont pas définis, ceux du nœud parent seront alors utilisés.
Afin de définir une forme aux objets, il faut implémenter l'interface `MaggLiteShaped` (lignes *22* à *31*). Cela permettra aux objets d'être « reconnus » par les mécanismes et interactions génériques de la boîte à outils. Il seront alors de fait compatibles avec les opérations associées. De la même manière, cette classe implémente l'interface `MaggLiteDraggable` afin de pouvoir utiliser le dispositif d'interaction `Drag` (Glisser-Déposer) avec ces objets. D'une manière générale, les interfaces proposées dans la boîte à outils permettent la compatibilité des objets avec les IAPs qui peuvent en tirer parti (nous reviendrons sur ce principe avec l'exemple de la conception du Glisser-Déposer dans la section D.2.1 page 52).

Cet exemple minimal permet déjà d'apprécier la base de la programmation d'objets graphiques avec MAGGLITE. Par contre, partir comme nous l'avons fait des classes abstraites pour créer des objets aux propriétés plus avancées nécessite de reimplémenter des méthodes déjà présentes dans des classes de plus haut niveau (gestion des formes, du glisser-déposer, des innertools, etc...) : `MaggLiteComponent` et ses dérivées (voir section 8.3.1 page 183). C'est pourquoi il est plus judicieux d'étendre ces classes de plus haut niveau implémentant déjà les interfaces et mécanismes souhaités, et de surcharger les méthodes correspondant aux modifications voulues (l'exemple précédent se résumerait alors à surcharger le constructeur pour définir une forme et une couleur).

Toutefois, l'extension des capacités des objets graphiques par héritage s'avère peu flexible et modulaire dans le cas fréquent où il est nécessaire d'associer de nouvelles possibilités graphiques à des interactions particulières. Prenons pour cela l'exemple du dessin. Il peut être utile, pour beaucoup d'interfaces avancées, de dessiner sur des objets de l'interface (reconnaissance de gestes, annotations, etc.). Dès lors, il faut spécifier l'interaction de dessin et les propriétés de l'objet graphique associé. Mais si l'on veut pouvoir dessiner sur plusieurs types d'objets, il faut :

1. soit étendre leur super classe commune, afin que tous les objets héritent de cette capacité. Cette approche est possible, mais nous semble peu élégante dans notre modèle car retombant dans la vision monolithiques des boîtes à outils standards.

2. soit étendre chacune des classes d'objets, ce qui nécessite de répéter le même code dans différentes classes, limitant alors la réutilisabilité de ce code.

Par couche (calque)

Une alternative avec MAGGLITE est d'utiliser des calques pour rajouter des couches graphiques et d'interaction aux objets. Plus flexible et modulaire, cette approche permet de composer des objets afin de leur associer de nouvelles capacités, améliorant alors la réutilisabilité des objets existants. Pour notre exemple du dessin, il suffit d'implémenter une classe de calques de dessin qui étend les calques de base (voir le code D.3).

Code Java D.3 – Exemple de création d'un nouveau calque. Le calque de dessin

```
1  public class MaggLiteDrawLayer extends MaggLiteLayer implements MaggLiteDraw {
2    //buffer de dessin
3    protected BufferedImage buffer;
4
5    public MaggLiteDrawLayer(String name) {
6      super(name);
7      // initialisation du buffer de dessin
8
9    }
10
11 // xmethods
12   @XMethodDescriptor(
13       name = "ShowGrid",
14       paramsNames = {"show", "size"}
15   )
16   public void xsetShowGrid(boolean show, int gridSize) {
17       // xmethods pour créer un slot permettant de faire afficher ou non une
             grille en fond du calque
18   }
19
20 // surcharge des méthodes graphiques
21   public void paintBefore(MaggLiteGraphics2D g) {
22       // affichage éventuel de la grille
23   }
24
25   public void paintOnTop(MaggLiteGraphics2D g) {
26       // affichage du buffer de dessin
27   }
28
29 // implémentation des méthodes de l'interface MaggLiteDraw
30   public void erase() {
31       // effacement complet
32   }
33
34   public void draw(Point2D from, Point2D to, double sizeFrom, double sizeTo,
         MaggLiteColor colorFrom, MaggLiteColor colorTo, DrawingBrush brush) {
35       // actions pendant le dessin
36   }
37
38   public void startDraw(Point2D p, double size, MaggLiteColor color, DrawingBrush
       brush) {
39       // actions au début du dessin
40   }
41
42   public void endDraw(Point2D p, double size, MaggLiteColor color, DrawingBrush
       brush) {
```

```
43          // actions à la fin du dessin
44      }
45
46      public void erase(Point2D from, Point2D to, double sizeFrom, double sizeTo,
            DrawingBrush brush) {
47          // opération d'effacement
48      }
49
50  }
```

Cette nouvelle classe de calques hérite des propriétés de la classe `MaggLiteLayer` (redimensionnement automatique, adaptation à la forme du composant conteneur, transparence, pliage, etc.) et surcharge ses méthodes graphiques pour afficher le dessin sur l'objet. Pour le dessin, elle implémente les méthodes de l'interface `MaggLiteDraw` afin d'être compatible avec les interactions de dessin qui seront définies par un nouveau dispositif d'interaction ou un nouvel outil interne (voir la section D.3.1 page 56 de cette annexe).

Il suffit alors d'associer un outil interne à une instance de ce calque et de l'ajouter dans un objet graphique pour pouvoir dessiner sur l'objet, sans avoir eu à modifier son code ou ses propriétés (voir code D.4).

Code Java D.4 – Utilisation du calque de dessin

```
1  // Création de l'objet graphique
2  MaggLiteBlueOval oval = new MaggLiteBlueOval("Oval");
3  desk.add(oval);
4
5  //Création du calque de dessin
6  MaggLiteDrawLayer drawLayer = new MaggLiteDrawLayer("draw layer");
7  //Ajout d'un outil interne de dessin
8  drawLayer.add(new DMaggLiteDrawInnerTool());
9  //Ajout du calque dans l'objet
10 oval.add(drawLayer);
```

Cette approche permet d'étendre et de réutiliser les capacités internes des objets. Elle s'avère par contre inadaptée à l'extension des capacités des objets pour une interaction directe (qui doivent agir sur l'objet). Pour cela, la méthode précédente, par héritage, est appropriée bien que favorisant la redondance du code. Il serait alors intéressant d'étudier l'utilisation de la *programmation par aspects* pour l'extension des objets graphiques. Toutefois, cette technologie nous semble trop limitée à l'heure actuelle, en particulier au niveau des performances dans le cadre d'applications graphiques et hautement interactives comme celle que nous proposons. Cela en fait tout de même une piste pour compléter les mécanismes de notre boîte à outils afin de proposer trois approches d'extension complémentaires :

1. *l'héritage*, pour créer de nouvelles classes d'objets graphiques à partir des classes de base ;
2. *les aspects*, pour étendre certaines capacités que nous qualifierons *d'internes* de classes d'objets graphiques sans avoir à étendre les classes de bas niveau ou à répéter le code dans plusieurs classes ;
3. *les calques* pour ajouter des capacités graphiques et interactives à des objets graphiques de manière modulaire et dynamique.

D.2 Les dispositifs d'interactions

La boîte à outils fournit les mécanismes de base pour créer simplement de nouveaux dispositifs d'interaction à partir de la classe abstraite `DMaggLiteGenericInteraction`. Bien qu'il puisse être nécessaire de connaître certaines bases de la programmation avec ICON pour construire des dispositifs complexes (en particulier la création de slots et la lecture de leurs valeurs), le squelette de dispositif de haut-niveau que nous proposons permet de s'en affranchir en grande partie. En effet, nous avons regroupé les mécanismes bas-niveau communs aux dispositifs d'interaction de MAGGLITE au niveau de la classe abstraite afin de permettre au programmeur de ne développer que les actions spécifiques à l'interaction qu'il veut réaliser.
Nous présenterons deux exemples : la conception du dispositif de Glisser-Déposer (*Drag*) et celle du dispositif des *Responsive Handles*.

D.2.1 Conception du dispositif de Glisser-Déposer

Tout d'abord, rappelons le principe de l'interaction de Glisser-Déposer. Elle consiste à :

1. sélectionner un objet,
2. initier l'action,
3. déplacer l'objet,
4. terminer l'action.

FIGURE D.2 – Dispositif de Glisser-Déposer.

Pour réaliser cette interaction dans le paradigme des graphes d'interaction, et donc avec un dispositif générique d'interaction, il va falloir tout d'abord spécifier les objets auxquels peut s'appliquer l'interaction et construire un dispositif recevant par ses slots d'entrée (voir figure D.2) : les objets sélectionnés (le slot *id*, la sélection des objets est confiée au dispositif `pick`), des valeurs numériques de position (slots *p.x* et *p.y*) et une commande d'utilisation (slot *use*[(2)]).
À chaque itération de la machine réactive, il faudra donc, à partir des valeurs reçues en entrée, effectuer les actions correspondantes aux étapes précédemment citées.

La classe abstraite `DMaggLiteGenericInteraction` fournit ces slots d'entrée et maintient à jour les sélections et vérifications de compatibilité des objets ainsi que les commandes d'utilisation

[(2)]Dans la figure D.2 et l'implémentation réelle de ce dispositif, il y a deux slots de commande pour que le dispositif permette de réaliser des interactions plus complexes. Nous n'exposerons pas ce principe, et détaillerons l'implémentation comme s'il n'y avait que le slot *use* unique de la classe abstraite de base des dispositifs d'interaction génériques.

de l'interaction à chaque pas de la machine réactive. Elle propose alors des méthodes de plus haut-niveau permettant de réaliser les actions sur les objets graphiques. Le code Java D.5 présente (en partie) l'implémentation du dispositif de Glisser-déposer à partir de cette classe.

Code Java D.5 – Structure du code du dispositif `Drag` (Glisser-déposer).

```
1  public class DMaggLiteDrag extends DMaggLiteGenericInteraction {
2
3    // Définition des classes d'objets compatibles avec l'interaction
4    private final static Class[] accepted = {MaggLiteMoveable.class,
         MaggLiteDraggable.class};
5    private final static Class[] excluded = {};
6
7    // Constructeur
8    public DMaggLiteDrag() {
9      super("MaggLite Drag");
10   }
11
12   // Implémentation des méthodes abstraites
13
14   // Ouverture du dispositif
15   public boolean openTasks() {
16     // réinitialiser l'interaction
17   }
18
19   // Fermeture du dispositif
20   public void closeTasks() {
21     // annuler toutes les manipulations en cours
22   }
23
24   // Retourne les classes d'objets compatibles
25   public Class[] getAcceptedTypes() {
26     return accepted;
27   }
28
29   // Retourne les classes exclues parmis les compatibles
30   public Class[] getExcludedTypes() {
31     return excluded;
32   }
33
34   // Actions exécutées à chaque pas
35   public void alwaysProcess() {}
36
37   // Actions à réaliser lors de l'entrée dans la proximité d'un objet
38   public void enteredProximity(MaggLite ml, boolean fromShape) {}
39
40   // Actions à réaliser lors d'un mouvement dans la proximité d'un objet
41   public void movedProximity(MaggLite ml) {}
42
43   // Actions à réaliser lors de la sortie de la proximité d'un objet
44   public boolean exitedProximity(MaggLite ml, boolean toShape) {return true;}
45
46   // Actions à réaliser lors de l'entrée dans un objet
47   public void entered(MaggLite ml, boolean fromProximity) {
48     // sauvegarder la position à l'intérieur de l'objet
49     // avec la méthode getPos()
50   }
```

```
51
52    // Actions à réaliser lors du mouvement dans un objet
53    public void moved(MaggLite ml) {
54      if (isInUse()) {
55        // signifier à l'objet le début ou la continuation de l'interaction
56        // réaliser les actions nécessaires (déplacement, surlignage, etc)
57      } else {
58        // si l'objet est en cours de drag lui signifier la fin de l'interaction
59      }
60    }
61
62    // Actions à réaliser lors de la sortie d'un objet
63    public boolean exited(MaggLite ml, boolean toProximity) {
64      if (isInUse())
65          return false; // continuation de l'interaction
66      // sinon, signifier à l'objet la fin de l'interaction
67      return true; // arrêt de l'interaction sur cet objet
68    }
69  }
```

Dans un premier temps, les variables de classe `accepted` et `excluded` permettent de définir les classes d'objets graphiques compatibles avec l'interaction. Elles sont retournées par l'implémentation des méthodes `getAcceptedTypes()` et `getExcludedTypes()` [(3)], lignes 25 et 30. Ces méthodes permettent à la super classe de trier les objets sélectionnés reçus, assurant que les objets passés en paramètre des méthodes d'action seront compatibles avec l'interaction. Pour notre dispositif de `Drag` , les classes compatibles doivent implémenter les interfaces `MaggLiteMoveable` et `MaggLiteDraggable`.
Ensuite, les méthodes `openTasks` et `closeTasks` sont appelées lors de l'ouverture et de la fermeture du dispositif. Elles permettent de réaliser des actions spécifiques à l'interaction à ces moments précis (initialisation du dispositif à l'ouverture ou annulation des actions en cours à la fermeture).
Enfin, les méthodes d'action permettent d'implémenter le cœur de l'interaction. Elles sont invoquées par l'algorithme principal de la classe abstraite, à chaque pas et pour chaque objet graphique compatible, selon les cas suivants :

- `alwaysProcess()` : à chaque pas, quelles que soient les valeurs des entrées du dispositif ;
- `enteredProximity(MaggLite ml, boolean fromShape)` : entrée dans la proximité de l'objet compatible `ml` . La variable booléenne `fromShape` permet de savoir si l'entrée s'est faite par l'intérieur de l'objet ou par l'extérieur ;
- `movedProximity(MaggLite ml)` : déplacement dans la proximité de l'objet `ml` ;
- `boolean exitedProximity(MaggLite ml, boolean toShape)` : sortie de la proximité de l'objet compatible `ml` . La variable booléenne `toShape` permet de savoir si la sortie se fait en direction de l'objet ou vers l'extérieur. La valeur booléenne de retour permet de signifier à l'algorithme principal si l'interaction doit continuer malgré la sortie ;
- `entered(MaggLite ml, boolean fromProximity)` : entrée dans l'objet compatible `ml` . La variable booléenne `fromProximity` permet de savoir si l'entrée s'est faite par la proximité de l'objet ;
- `moved(MaggLite ml)` : déplacement à l'intérieur de l'objet compatible `ml` ;
- `boolean exited(MaggLite ml, boolean toProximity)` : sortie de l'objet compatible `ml` . La variable booléenne `toProximity` permet de savoir si la sortie se fait vers la

[(3)] Cette méthode permet d'exclure des classes d'objets parmi celles qui sont acceptées (sous-classes ou interfaces multiples par exemple).

proximité de l'objet. La valeur booléenne de retour permet de signifier à l'algorithme principal si l'interaction doit continuer malgré la sortie.

Pour notre exemple du `Drag`, les méthodes d'action dans la proximité des objets ne réalisent aucune action. Seules les méthodes d'action dans l'objet sont utilisées. Les commentaires dans le listing D.5 page 53 détaillent brièvement les actions réalisées. Notons simplement l'appel des méthodes `isInUse()` et `getPos()` définies dans la classe abstraite. Elle permettent de savoir si le dispositif est en cours d'utilisation (ordre reçu par le slot *use*) et les positions reçues (slots *x* et *y*).

D.2.2 Exemple des *Responsive Handles*

L'implémentation des *Responsive Handles* utilise quand à elle les actions de proximité pour créer une poignée, la déplacer ou la supprimer (voir le code Java D.6). Cette implémentation est relativement simple par rapport la réalisation éventuelle de cette nouvelle interaction avec une boîte à outil standard. De plus, elle est générique, pouvant être utilisée avec de nombreuses classes d'objets graphiques. Il nous a fallu réaliser :

1. une nouvelle classe d'objets graphique, les poignées : `MaggLiteHandle`;
2. un nouveau dispositif d'interaction : `DMaggLiteHandle`;

Nous ne présenterons ici que le code du dispositif, sans détailler celui du composant de retour graphique (`MaggLiteHandle`) :

Code Java D.6 – Structure du code du dispositif `Responsive Handle`.

```
1  public class DMaggLiteHandle extends DMaggLiteGenericInteraction {
2
3    private final static Class[] accepted = {MaggLiteMoveable.class, MaggLiteSized.
         class, MaggLiteShaped.class};
4    private final static Class[] excluded = {MaggLiteHandle.class};
5
6    public DMaggLiteHandle() {
7      super("MaggLite Handle");
8    }
9
10   public boolean openTasks() {
11     // supprimer toutes les poignées en cours d'utilisation
12     return true;
13   }
14
15   public void closeTasks() {
16     // supprimer toutes les poignées en cours d'utilisation
17   }
18
19   public Class[] getAcceptedTypes() {
20     return accepted;
21   }
22
23   public Class[] getExcludedTypes() {
24     return excluded;
25   }
26
27   public void alwaysProcess() {}
28
```

```
29    public void enteredProximity(MaggLite ml, boolean fromShape) {
30      // associer une nouvelle poignée à l'objet reçu
31    }
32
33    public void movedProximity(MaggLite ml) {
34      // déplacer la poignée associée à l'objet reçu
35    }
36
37    public boolean exitedProximity(MaggLite ml, boolean toShape) {
38      // supprimer la poignée associée à l'objet reçu
39      return true;
40    }
41
42    public void entered(MaggLite ml, boolean fromProximity) {}
43    public void moved(MaggLite ml) {}
44    public boolean exited(MaggLite ml, boolean toProximity) {return true;}
45  }
```

Ce dispositif s'applique aux objets qui implémentent les interfaces `MaggLiteMoveable`, `MaggLiteSized` et `MaggLiteShaped`. En effet, il est nécessaire que les objets puissent être déplacés, redimensionnés et qu'ils aient une forme définie. Par contre, en sont évidemment exclus les objets de la classe `MaggLiteHandle`, les instruments de l'interaction.
Les actions de l'interaction sont réalisées dans les méthodes de proximité : création et association d'une poignée à un objet graphique lors de l'entrée, déplacement de la poignée associée lors d'un mouvement dans la proximité (suivi du pointeur) et suppression de la poignée lors de la sortie de la proximité.
Enfin, les actions réalisées spécifiquement par la poignée sur l'objet graphique (déplacement, redimensionnement et rotation) sont réalisée directement dans l'implémentation de la poignée, elle-même manipulée à l'aide du dispositif de Glisser-Déposer.

D.3 Les outils internes

La création de nouveaux outils internes reprend des principes proches de ceux des dispositifs d'interaction. La classe abstraite `DMaggLiteComponentInnerTool` fournit les mécanismes de base (slots communs, gestion du retour graphique, activation, mode et transmission des sorties, etc) et propose des méthodes abstraites pour réaliser les actions particulières à l'outil.

D.3.1 Exemple de l'outil interne de dessin

Le code Java suivant présente la structure de l'implémentation de l'outil interne de dessin : `DMaggLiteDrawInnerTool`.

Code Java D.7 – Structure du code de l'outil interne de dessin.

```
1  public class DMaggLiteDrawInnerTool extends DMaggLiteComponentInnerTool {
2
3    // slots
4    In width = addIn("width", SlotType.DOUBLE);
5    In r = addIn("color.r", SlotType.DOUBLE);
```

```
6     In g = addIn("color.g", SlotType.DOUBLE);
7     In b = addIn("color.b", SlotType.DOUBLE);
8     In a = addIn("color.a", SlotType.DOUBLE);
9
10    public DMaggLiteDrawInnerTool() {
11      super();
12      setName("DrawInnerTool");
13    }
14
15    //Implémentation des méthodes abstraites
16    public void doActivate() {
17      // Actions à l'activation de l'outil
18      // Initialisation du dessin
19    }
20
21    public void doDesactivate() {
22      // Actions à la désactivation de l'outil
23    }
24
25    protected void alwaysAction() {
26      // Actions réalisées à chaque pas d'exécution
27      // Traitements sur les valeurs des slots particuliers à l'outil
28    }
29
30    protected void toolActionOutside(boolean use) {
31      // Actions de l'outil en dehors des bornes du composant lié
32    }
33
34    protected void toolActionNoXY(boolean use) {
35      // Actions de l'outil si il n'y a pas eu de mouvement
36    }
37
38    protected void toolAction(boolean use, Point2D localP, Point2D screenP) {
39      // Actions de l'outil si il y a eu un mouvement dans
40      // les bornes du composant
41        if (use) {
42          // Dessin sur le composant du point précédent au point localP courant
43          // en utilisant les valeurs des slots de taille et de couleur et la
44          // méthode compatible de l'objet graphique
45          component.draw(...);
46        }
47    }
48  }
```

Cet outil, en plus des slots par défaut gérés par la super classe (*use*, *x* et *y*), propose des slots particuliers pour gérer l'épaisseur et la couleur du trait (lignes 3 à 7).
Les méthodes `doActivate` et `doDesactivate` (lignes 17 et 22) permettent de réaliser les actions propres à l'outil lors de son activation et de sa désactivation.
Enfin, les méthodes suivantes sont invoquées par l'algorithme principal du dispositif abstrait, selon les entrées reçues et le mode d'utilisation de l'outil (local ou global) :

- `alwaysAction` : cette méthode est appelée à chaque pas d'exécution, quelles que soient les valeurs des slots d'entrée ;
- `toolActionOutside(boolean use)` : actions à réaliser lorsque l'outil est en dehors des bornes graphiques du composant lié ;
- `toolActionNoXY(boolean use)` : actions à réaliser si il n'y a pas eu de mouvement de

l'outil ;

– `toolAction(boolean use, Point2D localP, Point2D screenP)` : actions à réaliser si il y a eu mouvement de l'outil dans les bornes du composant. Les paramètres `localP` et `screenP` transmettent la position actuelle de l'outil en coordonnées locales au composant et en coordonnées écran ;

Pour notre outil interne de dessin, la méthode `alwaysAction` va permettre de traiter à tout moment les entrées reçues par les slots particuliers à l'outil (taille et couleur).
La méthode `toolAction` va implémenter l'action de dessin sur le composant graphique associé à l'outil, accédé par la variable `component` . Notons que, contrairement aux dispositifs d'interaction, aucune vérification de compatibilité n'est faite au niveau de la super classe lors de l'association entre un objet graphique et un outil interne. C'est donc au niveau de l'implémentation de l'outil concret que doit être réalisée cette vérification (nous ne l'avons pas détaillé dans cet exemple de code par soucis de clarté).

D.4 Création d'une application

Finalement, le code suivant présente la création d'une application simple avec MAGGLITE, reprenant les exemples précédents. Il utilise la classe abstraite `MaggLiteApplication` qui décharge le programmeur de toute la création et l'initialisation des environnements d'exécution de Swing (ressources graphiques et fenêtre système), d'ICON et de MAGGLITE.

Code Java D.8 – Exemple de création d'une application MAGGLITE. L'abstraction d'application ne nécessite l'implémentation que de deux méthodes.

```
1  public class HelloMaggLiteWorld extends MaggLiteApplication {
2
3    private MaggLiteDrawLayer drawLayer;
4
5    public helloMaggLiteWorld(String[] argv) {
6      super("Hello MaggLite World", argv, true);
7    }
8
9    // Méthode de création et d'ajout des composants graphiques
10   public void createMaggLiteComponents(JMaggLiteDesk desk) {
11     MaggLiteOval oval = new MaggLiteOval("Hello Oval");
12
13     drawLayer = new MaggLiteDrawLayer("draw layer");
14
15     oval.add(drawLayer)
16     desk.add(oval);
17   }
18
19   // Méthode de création et d'ajout de widgets Swing
20   public void createSwingComponents(JMaggLiteFrame frame) {}
21
22   // Méthode de création et d'ajout des dispositifs spécifiques à l'application
23   public void createApplicationDevices() {
24     drawLayer.add(new DMaggLiteDrawInnerTool());
25   }
26
27   public static void main(String[] argv) {
```

```
28      HelloMaggLiteWorld app = new HelloMaggLiteWorld(argv);
29      app.start();
30    }
31
32  }
```

Les méthodes à réaliser sont :

1. `createMaggLiteComponents(JMaggLiteDesk desk)` , pour créer les objets graphiques de l'application et les ajouter au bureau (`desk`). Dans notre exemple, nous créons un objet de la classe `MaggLiteOval` auquel est ajouté un calque de dessin ;
2. `createSwingComponents(JMaggLiteFrame frame)` , pour créer et ajouter des widgets Swing standard à l'application (ces widgets peuvent aussi être insérés dans les graphes de scène et contrôlés par des interactions de MAGGLITE).
3. `createApplicationDevices()` , pour créer et ajouter au configurateur d'entrée des dispositifs spécifiques à l'application. Ici, nous associons simplement un outil interne de dessin au calque de dessin. Dans le cadre d'applications plus complexes, des dispositifs plus particuliers peuvent être réalisés et ajoutés dans cette méthode.

Enfin, la méthode standard `main(String[] argv)` permet d'exécuter l'application en créant une instance de la classe réalisée et en la démarrant par la méthode `start()` . Dès lors, l'exécution de ces quelques lignes va produire une application entièrement configurable avec ICON, où il sera possible de dessiner sur l'objet ovale, mais aussi de le manipuler (déplacement, redimensionnement, transformations, etc.) en utilisant toutes les interactions génériques de MAGGLITE.

D.5 Plus d'exemples...

Nous n'avons esquissé dans cette annexe qu'une vue globale et minimale des principes de programmation avec MAGGLITE. Elle permet toutefois de saisir la *philosophie* de la programmation modulaire et flexible que propose la boîte à outils. Nous n'y avons volontairement pas évoqué la réalisation et l'utilisation de composants tels que les transformations ou les comportements afin de ne pas la surcharger. Des exemples d'applications ou de composants plus complexes, ainsi qu'une documentation (en cours de réalisation) sont disponibles avec la distribution libre de MAGGLITE à l'adresse internet `http://www.emn.fr/x-info/magglite` .

UNE NOUVELLE APPROCHE POUR LA CONCEPTION CRÉATIVE :
DE L'INTERPRÉTATION DU DESSIN À MAIN LEVÉE
AU PROTOTYPAGE D'INTERACTIONS NON-STANDARD

Stéphane HUOT
Doctorat de l'Université de NANTES

Résumé

Il existe de nos jours beaucoup d'environnements dits de *CAO*, conçus pour assister et accompagner le travail des concepteurs tout au long d'un projet. Pourtant, ces logiciels sont peu adaptés à la première phase d'un projet : la *conception créative*. En effet, leurs approches géométriques et excessivement directrices ignorent le caractère flou et itératif de la démarche du concepteur, appuyée par un outil figuratif essentiel : le *croquis*. Cette thèse s'inscrit dans un courant de travaux axés sur l'utilisation du dessin pour la réalisation de modèles numériques 3D, dans le but de fournir des *outils* informatiques de support à la conception créative. Mais au delà de l'interprétation de croquis, nous proposons une approche originale appliquée au domaine de la conception architecturale, reposant sur un environnement instrumenté : SVALABARD, une *Table à dessin virtuelle*, basée sur un paradigme de *feuilles d'interaction*, composant des périphériques d'entrée et des techniques d'interaction non-standard.
Toutefois, la faisabilité technique d'une telle approche a rapidement été compromise de par la rigidité et le manque d'extensibilité des outils actuels pour le développement d'interfaces. C'est pourquoi nous proposons aussi dans cette thèse un nouveau modèle d'architecture logicielle plus flexible et dynamique, qui, une fois réalisé dans la boîte à outils MAGGLITE, nous a permis de faciliter et d'encourager le prototypage et la réalisation d'interactions avancées.

Mots-clés : Interaction Homme-Machine, Conception créative, Modélisation 3D, Interfaces à stylet, Périphériques d'entrée, Techniques d'interaction, Interaction multimodale, Interfaces Post-WIMP, Boîtes à outils d'IHM.

Abstract

So far, many CAD systems have aimed at assisting and supporting designers work all along project workflow. However, such tools are rather useless in the early stages of the design process, during *creative design*. Their geometrical and preconceived processes are far from the woolly and iterative aspects of the creative design process, mainly supported by a figurative tool : *sketching*. This thesis is in line with previous research works on sketching-based 3D modelling to bring a creative dimension in software tools for design. Beyond drawing interpretation, we introduce an instrumented approach for architectural design : SVALABARD, a *Virtual drawing table* relying upon the paradigm of *interaction sheets*, that associates alternative input devices and techniques.
Because the feasibility of such an approach was technically restricted by the stiffness and the lack of extensibility of current GUI tools, we also propose a new software model, more adaptable and more dynamic. With this model, we developed the MAGGLITE toolkit that makes it easier to prototype and use alternative interaction techniques.

Keywords : Human-Computer Interaction, Creative design, 3D Modelling, Pen-based interfaces, Input devices, Interaction techniques, Multimodal interaction, Post-WIMP interfaces, GUI Toolkits.