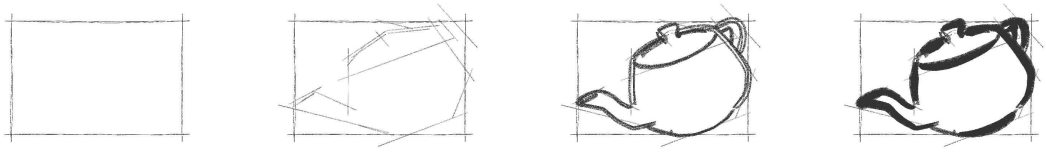
Le style dans le rendu non-photoréaliste de dessins au trait à partir de scènes 3D : une approche programmable

Stéphane Grabli

Université Joseph Fourier de Grenoble (UJF)

Le style dans le rendu non-photoréaliste de dessins au trait à partir de scènes 3D : une approche programmable



Stéphane GRABLI

Thèse présentée le 24 mars 2005
pour l'obtention du titre de
Docteur de l'Université Joseph Fourier
Spécialité Informatique
Arrêté Ministériel du 5 juillet 1984 et du 30 mars 1992
Préparée au sein du laboratoire
ARTIS-GRAVIR/IMAG-INRIA. UMR CNRS C5527.

Composition du jury :

François	SILLION	Directeur de thèse
Georges-Pierre	BONNEAU	Président du Jury
Victor	OSTROMOUKHOV	Rapporteur
Thomas	STROTHOTTE	Rapporteur
Frédo	DURAND	Examinateur

Remerciements

De nombreuses personnes ont contribué, d'une manière ou d'une autre, à cette thèse : je tiens ici à les remercier. Merci donc à François Sillion, mon directeur de thèse, pour m'avoir guidé dans ma découverte de la synthèse d'images et de la recherche (et ce en dépit des récurrentes pénuries de stylos qui compliquèrent sa tâche ;-)), à Claude Puech pour avoir rendu cette expérience possible en m'accueillant très tôt au sein de son laboratoire et de l'équipe iMAGIS, à Frédo Durand, pour m'avoir invité au MIT pendant sept mois et co-encadré de manière informelle, ce fut une formidable expérience et l'occasion d'une collaboration fructueuse, à la région Rhône-Alpes pour avoir financé ce séjour par le biais de la bourse eurodoc, à Victor Ostromouhov, Thomas Strothotte, Frédo Durand et Georges-Pierre Bonneau, pour avoir accepté de constituer mon jury et pour avoir pris le temps de venir (souvent de loin) assister à ma soutenance et plus particulièrement à Victor et à Thomas pour avoir consenti à être rapporteurs de cette thèse, à Jacques Stern pour ses conseils et son aide précieuse, je lui suis largement redevable de ce parcours, au CIES pour m'avoir permis de faire l'expérience de l'enseignement au travers du monitorat, à Sylvain Paris et Sylvain Lefebvre, les thésards de ma "promotion", pour leur présence et leurs bons conseils pendant ces trois années, à Emmanuel Turquin, pour m'avoir grandement aidé au développement du logiciel Freestyle et ce avec tant de bonne volonté, je garderai de très bons souvenirs de nos soirées de deadline, à Sylvain Paris, Samuel Hornus, Sylvain Lefebvre, Elmar Eisemann, Xavier Décoret, Cyril Soler, Joelle Thollot, et d'une manière générale, à tous les thésards, permanents et dea des équipes ARTIS et EVASION qui ont tous contribué à cette thèse par une réponse, un conseil, une relecture, etc., et auprès de qui j'ai réellement tout appris jour après jour dans une atmosphère amicale, à Gilles Debunne, pour sa librairie QGLViewer qui nous a tant servi, ainsi qu'aux développeurs des logiciels libres swig, STLPort, cygwin, et j'en oublie, qui nous ont économisé plusieurs années de développement, merci à Johan Martinsson pour avoir été un parfait colocataire, et un excellent ami pendant quatre ans à Grenoble, à Delphine Alles pour sa patience, son soutien et son aide pendant la douloureuse période de la rédaction, à mes amis Mathieu, Clotilde, Laure, et Virginie, pour avoir eu la gentillesse et le courage de subir l'épreuve de la soutenance, aux amis de Paris : Nico, Manu, Anthony, Marion, Vincent, Nabila, Charlotte, Aurélien, Elie, Judith et Frédéric, Loïc et Marie-Laure, Jean-David, les marmottons, aux amis de Grenoble : Stéphane et Fanny, Galith, Cyril, Raph, Jean-Noël, Camille, Natacha, Paola, Nathalie et le groupe des 13, et aux amis de Boston : Shelly, Elodie, Neda, Uriel, Déborah, Paul, Adnan, pour avoir partagé mes moments de détente.

Enfin, c'est à mes parents Nicole et Paul Grabli et à mon frère David que j'adresse toute mon affection et ma reconnaissance pour m'avoir permis d'en arriver là.

Je dédie cette thèse à Jean.

Table des matières

chapitre 1

Introduction

La quête du photoréalisme : cette motivation a guidé la recherche en informatique graphique depuis son origine. Aujourd'hui, les techniques de rendu utilisées pour produire des images photoréalistes à partir de scènes 3D arrivent à maturité et l'illusion est souvent parfaite. Plus récemment, le besoin de créer des images moins conformes à la réalité, plus expressives, à partir de scènes 3D s'est fait ressentir au sein d'une partie de la communauté de l'informatique graphique. Le domaine de recherche dédié au rendu non-photoréaliste (*Non-Photorealistic Rendering* ou NPR) est ainsi né il y a une dizaine d'années et ne cesse depuis de s'accroître.

1.1 Pourquoi faire du NPR ?

De la liberté dans la représentation

Choisir le NPR, c'est choisir toute la liberté offerte par le dessin ou la peinture dans la représentation et qui n'existe pas dans la photographie. Cette liberté, c'est celle d'abstraire, de simplifier, de mettre en valeur, que les artistes ont utilisée, perfectionnée au cours des siècles en jouant notamment sur les couleurs ou les traits. Pour cette raison, le dessin et la peinture sont encore aujourd'hui des moyens d'expression et de communication privilégiés.

Le dessin est un outil pédagogique

Il est également fréquent de préférer le dessin pour réaliser des illustrations à but pédagogique dans des domaines aussi variés que la médecine où la mécanique. Ce succès s'explique par une particularité du dessin : il permet de représenter de manière claire les éléments les plus complexes. En effet, le fonctionnement de notre système visuel permet de correctement appréhender les formes uniquement à partir du tracé de leurs contours. La représentation par le dessin peut ainsi se traduire par une formidable économie de moyens et donc une plus grande lisibilité de la représentation. Le dessin permet également à l'artiste de s'éloigner de la réalité à sa guise ; il est ainsi courant, en illustration technique, d'afficher les parties cachées d'un objet de manière à ce que son mécanisme ou sa structure soient mieux perçues (*cf.* figure 1.1 page suivante).

L'abstraction permet de communiquer efficacement

C'est également le dessin que l'on choisira dès qu'il s'agit de communiquer efficacement. En effet, avec le dessin on peut abstraire, et cette abstraction aboutit à une représentation simple

et plus communicative qu'une photographie. Les symboles utilisés pour les signalisations par exemple sont la plupart du temps des dessins d'une simplicité extrême, comme on peut le voir sur la figure 1.1. Cette simplicité permet en particulier une interprétation rapide et non ambiguë du message à transmettre.

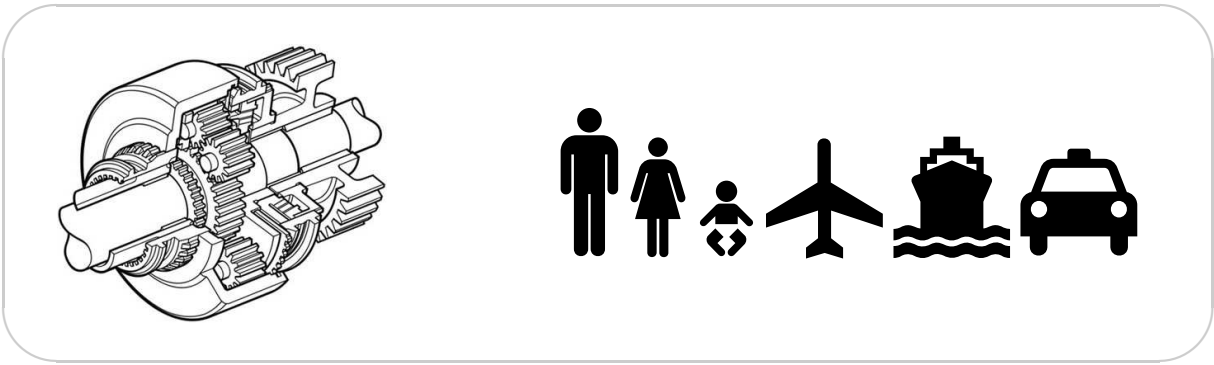


FIG. 1.1 – Le dessin comme outil pédagogique et moyen de communication

Les illustrations techniques (à gauche) démontrent l'intérêt pédagogique lié à l'utilisation du dessin. La capacité de ce dernier à représenter les objets complexes de manière minimaliste mais claire en fait un outil de communication puissant, qu'on utilise par exemple pour les signalisations (à droite).

En outre, la possibilité qui nous est ainsi offerte de représenter le monde en quelques traits ouvre des perspectives intéressantes :

■ *Les dessins restent intelligibles sur des petites surfaces d'affichage*

La représentation de scènes à l'aide de peu de lignes est particulièrement bien adaptée aux systèmes mobiles possédant de petites surfaces d'affichage, tels que les téléphones portables ou les ordinateurs de poche. L'utilisation du NPR dans ce contexte a déjà fait l'objet de plusieurs travaux de recherche [HMK02] et tend à se populariser.

■ *Un contenu simple ou incomplet peut être suggéré*

D'une part, l'art, en tolérant l'imprécision, le flou, offre un potentiel de suggestion intéressant. En illutration archéologique par exemple, les formes supposées de parties manquantes sont esquissées plutôt que tracées précisément, de manière à laisser percevoir l'incertitude relative à leur apparence exacte. Ou encore, en architecture, les croquis de futurs bâtiments conservent souvent une allure d'ébauche permettant de masquer le manque de détails inhérent à ce type de plans. En informatique graphique, la création de contenu reste à ce jour un travail long et fastidieux dans la mesure où un rendu réaliste ne peut se satisfaire de scènes trop simples ou incomplètes. Le NPR, en autorisant des contenus plus simples, ouvre la porte aux techniques de création rapide et intuitive de modèles 3D [IMT99, CHZ00].

■ *L'abstraction peut être plus esthétique que le réalisme*

D'autre part, préférer l'abstraction au réalisme peut aussi évidemment relever d'une préférence esthétique. Après tout, les peintres eux-mêmes, après être parvenus à imiter à la perfection la réalité s'en sont détournés pour faire naître le cubisme, l'impressionnisme, *etc.*. . . Dans le domaine de l'animation, malgré l'avènement récent de la synthèse d'images

avec des films tels que *Final Fantasy*, de plus en plus de studios s'intéressent à des aspects "dessinés" stylisés dans leurs productions (*cf.* figure 1.2).

(a) (b)

FIG. 1.2 – Le dessin stylisé comme choix esthétique

Malgré le récent succès des animations photoréalistes ((a) image du film Final Fantasy par Square Pictures, © H. Sakaguchi et M. Sakakibara 2001 - Square Pictures [SS01]), de nombreux réalisateurs préfèrent opter pour des aspects "dessinés" fortement stylisés ((b) à gauche, image du film Les Triplettes de Belleville par Sylvain Chomet © Sylvain Chomet 2003 - 2d.3D [Cho03], à droite, image tirée du film Waking Life de Richard Linklater, réalisé selon la technique du rotoscoping © Richard Linklater 1991 - [Lin01].)

La 3D pour factoriser

Le rendu expressif est donc, on l'a vu, un enrichissement pour l'informatique graphique. En revanche on peut se demander ce que pourrait apporter l'utilisation de données 3D au monde de l'animation traditionnelle et des illustrations.

L'automatisation permet de gagner du temps

En synthèse d'images, on distingue habituellement deux tâches, la modélisation et le rendu, complétées par une phase de définition de l'animation lorsqu'il s'agit de produire une séquence animée. La modélisation consiste à créer les modèles 3D et à les positionner dans la scène. L'animation définit les positions des cameras et des objets animables à des instants clés. Ces deux opérations concentrent l'essentiel des interactions requises auprès de l'utilisateur. Le rendu quant à lui, se charge de générer automatiquement l'image, ou les images, à partir des informations de scène.

L'apport le plus important que représente la synthèse d'images pour le monde de l'illustration traditionnel est un gain de temps à travers une plus grande automatisation.

Il est probable que, encore aujourd'hui, il soit souvent plus rapide de dessiner ou peindre directement une illustration, que de modéliser la scène correspondante. Si l'on considère, en revanche, l'exemple d'une séquence animée pour laquelle 25 dessins par seconde doivent être produits, il est sûrement plus intéressant de modéliser les objets impliqués dans la séquence, d'en définir l'animation et de générer les images, que de faire à la main chaque dessin. De même, pour la fabrication d'un manuel d'utilisation, les illustrations sont encore la plupart du temps faites à la main, alors qu'avec l'avènement de la conception assistée par ordinateur

(CAO) les objets dessinés sont de plus en plus souvent disponibles sous forme de modèles 3D. Ici encore, il paraîtrait judicieux de réutiliser les modèles existants et de générer automatiquement les dessins d'un tel manuel.

Ainsi, dans ce contexte, la 3D représente un potentiel de factorisation jusqu'à présent inexistant en illustration traditionnelle.

Tout un chacun peut devenir un "artiste"

Finalement, de façon plus triviale, le NPR met à la portée du plus grand nombre, la création d'illustrations de qualité.

- les grosses differences avec le PR : lines + mark-based rendering

1.2 Le style dans le NPR

La richesse de l'art pictural réside dans le style

Nous avons mentionné plus tôt que le dessin offrait plus de liberté dans la représentation que la photographie, notamment parce que l'artiste n'est pas obligé de se contraindre à un système de projection ou de primitives de tracé donné. C'est dans cette liberté, et donc dans les variations qu'on pourra observer entre différents artistes, que s'exprime le style. Le style est ainsi l'empreinte purement humaine et individuelle imprimée par un artiste à une représentation.

C'est cet aspect qui marque la différence majeure entre le rendu non-photoréaliste et le rendu traditionnel en synthèse d'images. En effet, alors que le premier se doit d'intégrer une composante humaine importante pour permettre la spécification du style de rendu, le deuxième relève d'un processus purement automatique : disposant de modèles pour la caméra, la lumière, et des propriétés photométriques des matériaux, les images sont calculées en simulant les interactions entre les rayons lumineux et les objets de la scène.

Une bonne compréhension des concepts manipulés en rendu non-photoréaliste passe donc par une étude de la notion de style en art pictural.

Notions de style

Dans cette section, nous reprenons une décomposition du processus de dessin qui fournit un cadre formel intéressant à l'étude de style.

L'analyse structurelle du style n'a commencé que récemment

Depuis plusieurs siècles, les historiens de l'art étudient les styles observés aux différentes époques au travers du prisme socio-culturel [Gom95b]. Ce n'est que récemment que des travaux se sont intéressés au style d'un point de vue structurel. Willats en particulier [Wil97a] s'est appuyé sur la recherche actuelle en perception visuelle et en intelligence artificielle pour développer une classification et un vocabulaire précis dédiés à la description du processus de création d'images. Nous nous appuyons sur cette classification pour spécifier le style. Par ailleurs, ses méthodes, plus proches de celles d'un linguiste moderne que de celles d'un historien de l'art, ont été une source d'inspiration pour cette thèse. Toutefois, aujourd'hui encore, il n'existe pas de véritable consensus sur les éléments constitutifs du style, et donc pas de définition précise du concept de style.

Quatre systèmes pour décrire la représentation picturale

En 2002, Durand [Dur02a] a repris un découpage proposé par Willats [Wil97a] et introduit une classification générale en quatre systèmes pour décrire les techniques de représentation, incluant les techniques artistiques classiques :

Le système spatial fait correspondre les propriétés spatiales 3D et les propriétés spatiales 2D (*e.g.* projection perspective).

Le système de primitives fait correspondre les primitives de l'espace objet (points, lignes, surfaces, volumes) avec les primitives de l'espace image (points, lignes, régions).

Le système d'attributs affecte les propriétés visuelles telles que la couleur, la texture, l'épaisseur, etc... aux primitives de l'image.

Le système de marques correspond au tracé physique des primitives à l'aide de marques dont l'aspect dépend des outils (*e.g.* pinceau, fusain), média (*e.g.* peinture à l'huile, aquarelle) et supports (*e.g.* toile, papier) employés.

Le style d'un artiste est défini par sa mise en œuvre de chacun de ces quatre systèmes.

1.3 Thèse

Comme nous l'avons expliqué plus tôt, bien que la notion de style soit centrale au rendu non-photoréaliste, elle reste floue, y compris, comme nous le verrons dans le chapitre 2, au sein de la communauté scientifique qui l'a abordée : de nombreux travaux en NPR se sont intéressés au développement de techniques de rendu stylisé et ont manipulé, de manière implicite, le concept de style ; très peu, en revanche, ont étudié le style en tant que tel et tenté d'en rendre les règles plus explicites.

Il nous a donc semblé intéressant de pousser plus loin l'étude du style et du processus de dessin dans l'espoir d'aboutir à une approche du rendu non-photoréaliste plus générale et plus complète.

1.3.1 Problématique

Comme nous l'avons souligné, la différence principale entre le rendu traditionnel et le rendu NPR, est le style, qui relève d'un processus humain et dont il est, par conséquent, essentiel que le contrôle revienne à l'utilisateur. La problématique de cette thèse tourne donc, de façon générale, autour de la question de la spécification d'un style de rendu par l'utilisateur.

Contrôle flexible du style

Le principal objectif de cette recherche est de proposer une application de NPR offrant à l'utilisateur un contrôle flexible du style du rendu. Il s'agit alors de déterminer quelles opérations permettent de décrire le style, quelles informations interviennent dans ce processus et comment structurer ces informations. Il est aussi fondamental de définir le type d'interface le plus adapté à un tel but.

Séparation du style et du contenu

Dans la mesure où l'une des motivations dominantes pour l'utilisation de la 3D est la réutilisation, il est essentiel d'assurer au maximum l'indépendance de la description de style

vis à vis du modèle. Par opposition à une application où l'utilisateur "peindrait" interactivement le style directement sur la scène, décrire le style, à la manière d'un algorithme, permet de réappliquer un même style aux rendus de différents modèles, dans l'esprit des "shaders" de *Renderman* [Ups89]. Cette séparation du style et du contenu est la clé commune à une automatisation suffisante du rendu et à un contrôle fin du style.

Pour avoir une meilleure idée de ce que cela signifie, on peut, par exemple, se figurer dans une situation où l'on doit décrire, par téléphone, un style observé sur une scène complexe, à quelqu'un qui n'a pas connaissance du dessin sur lequel ce style est observé et s'imaginer les phrases que nous prononcerions alors ; celles-ci ressembleraient probablement plus à : "les contours extérieurs de la scène sont tracés avec un poids fort" qu'à : "le trait en bas à gauche est plus épais, celui en haut à droite également, *etc.*...". Ce type de situation nous force à prendre du recul par rapport à la scène observée, et à analyser la structure même du style plutôt que ses effets. C'est à ce type de description de style que nous nous intéressons.

Introduction d'un formalisme pour la description de style

En fin de compte, s'il semble trop ambitieux de chercher à définir un *modèle* pour le style, un objectif de ce travail est de se rapprocher d'un tel modèle en introduisant un formalisme pour la description de style. Un tel formalisme peut être vue, par analogie avec la linguistique, comme la spécification d'un vocabulaire et d'une grammaire dédiés à la description de style. C'est un objectif d'autant plus important que ce formalisme manque à ce jour dans la communauté des chercheurs en NPR.

L'une des difficultés associées à une telle entreprise provient du fait que le rendu non-photoréaliste regroupe une très grande variété d'aspects visuels, provenant de multiples buts et contextes, ainsi que d'une vaste gamme de techniques, et qu'il est, par conséquent, complexe d'en trouver une formalisation unique. Nous pensons que la recherche décrite dans ce document peut constituer une base de départ solide à une formalisation plus poussée du style.

1.3.2 Approche

Dans cette thèse nous présentons une approche nouvelle du problème de la modélisation de style, qui découle d'une étude préliminaire du style dans des illustrations traditionnelles. Cette approche repose plus particulièrement sur un postulat que nous commençons par énoncer avant de préciser le cadre de travail dans lequel nous nous plaçons.

Postulat

les choix stylistiques faits par l'artiste s'appuient sur des informations extraites de la scène

Le point de départ de cette thèse est le postulat suivant lequel, en art pictural, dans de nombreux cas, les choix stylistiques faits par l'artiste ne sont pas complètement arbitraires et qu'ils reposent plus particulièrement sur des caractéristiques de la scène (*e.g.* la nature des lignes représentées) et de l'image (*e.g.* la densité des traits tracés). Cela signifie que si l'on regarde le style d'un dessin comme un ensemble d'attributs visuels (*e.g.* couleur, épaisseur des traits), alors, ce style peut être décrit comme un ensemble d'expressions mettant en relation ces attributs et des informations issues de la scène et de l'image. Une telle expression pourrait, par exemple, spécifier la manière dont l'épaisseur des traits doit varier en fonction de leur

distance au point de vue (ou plus exactement, de la distance des objets que ces traits dessinent au point de vue).

Ce postulat, établi de manière expérimentale par le biais de nombreuses observations d'illustrations traditionnelles, semble intuitivement raisonnable, comme nous le verrons dans le chapitre 3.

> **Remarque :** Il existe des illustrations pour lesquelles aucune relation liant les attributs de style et les informations ne semble se dégager, soit parce qu'aucune caractéristique visuelle ne se distingue, auquel cas la description de style est triviale, soit parce que la distribution des caractéristiques visuelles ne répond à aucune logique évidente (si ce n'est la volonté de l'artiste), du point de vue des informations considérées. Nous avons toutefois observé qu'il existe un grand nombre d'illustrations, couvrant une large variété de styles, auxquelles le postulat précédent s'applique. Selon nous, pouvoir générer tous ces styles au sein d'un même système constitue déjà une contribution significative.

Cadre de travail

Il aurait été ambitieux de vouloir aborder ces questions en traitant en même temps toutes les catégories de l'art pictural et en incluant toutes les composantes du style. Nous avons préféré restreindre notre champ d'étude, en analysant plus minitieusement un sous-espace des styles et des techniques.

Nous nous limitons aux dessins de lignes

Tout d'abord, nous avons choisi de nous limiter dans un premier temps à l'étude des dessins de lignes, c'est-à-dire ceux traçant les silhouettes, contours, *etc.*. . . d'une scène. La ligne est en effet la primitive privilégiée de nombreuses écoles graphiques, *e.g.* la bande dessinée ou l'illustration technique, pour réaliser la base minimale d'une représentation, ainsi que la caractéristique qui distingue le plus l'art plastique de la photographie. Par ailleurs, la ligne est une primitive 1-dimensionelle qui contraste avec les primitives 0-dimensionelles (les points) habituellement utilisées en rendu.

Se limiter aux dessins de lignes signifie aussi, dans notre cas, que nous ne traitons pas le remplissage de régions 2D de l'image (ce qui est indépendant du choix de la ligne comme primitive de tracé) (*cf.* figure 1.3). Bien qu'il soit courant dans le dessin au trait d'avoir recours à l'une ou l'autre des techniques de hachures pour indiquer le volume, la texture, l'ombrage des objets représentés, nous ne traçons que les lignes de "contours" habituellement utilisées, telles que les lignes de silhouette ou de crêtes par exemple. La figure 1.3 (b) illustre de telles lignes dans un dessin au trait. Quant à la raison de cette seconde limitation, c'est que le dessin de lignes, d'une part, et le dessin de hachures pour la mise en volume, d'autre part, sont deux problèmes fondamentalement différents qu'il serait difficle de vouloir traiter de manière identique. Le premier trace des traits pour représenter des limites, certes virtuelles puisque de telles lignes n'existent pas en tant que telles dans la scène réelle, mais qu'il est possible de "calculer" comme nous le verrons dans le prochain chapitre, en section 2.2. Le deuxième, quant à lui, trace également des traits, mais pour représenter des variations de ton sur des surfaces ; c'est-à-dire qu'il s'agit alors de remplir des régions 2-dimensionelles du dessin. Les problématiques soulevées par l'un et l'autre type de représentation sont très différentes et nous avons donc choisi de commencer par considérer le premier d'entre eux. Ainsi, les traits

que nous traçons forment d'une certaine manière un sous-ensemble des lignes "délimitantes" de la scène, aucun trait n'est ajouté. Le contrôle du style dans le cadre du remplissage de régions, à l'aide de hachures ou de coups de pinceaux par exemple, pour lequel des régions 2D sont tracées à l'aide de primitives 1D ou 0D, est aussi un sujet passionnant que nous laissons pour de futures recherches.

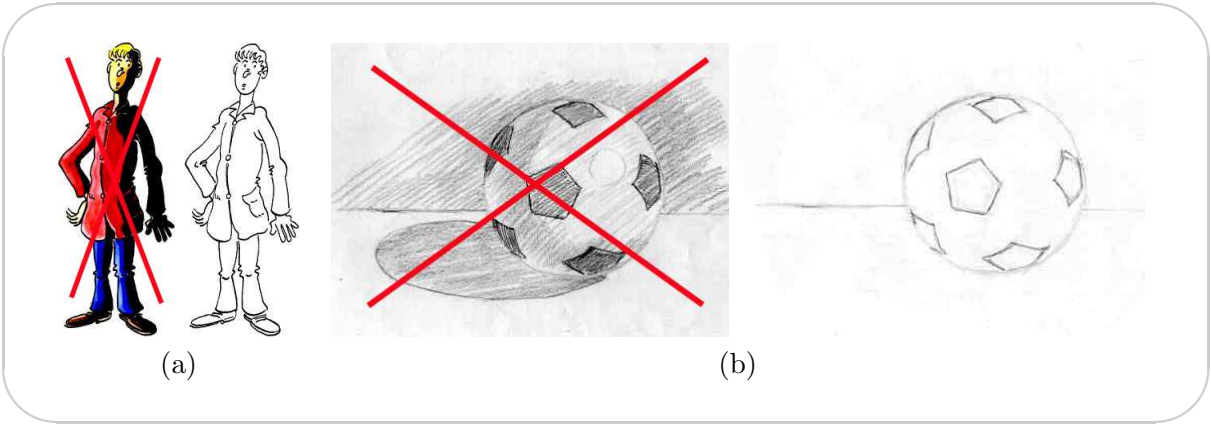


FIG. 1.3 – Dessins de lignes

Dans un premier temps, nous nous concentrons sur les dessins de lignes, c'est-à-dire que les composantes liées au remplissage de régions 2D, que ce soit par colorisation, (a), ou hachurage, (b), ne sont pas prises en compte.

■ *Le style est réduit aux seuls systèmes de primitives et d'attributs*

Nous restreignons également l'espace des styles couvert par notre approche. En nous référant à la classification introduite section 1.2, nous ne donnons de contrôle que sur deux des quatre systèmes impliqués dans la représentation graphique : les systèmes de primitives et d'attributs. La mise en œuvre du système de primitives en particulier, nous semble être une composante complexe et importante du style qui a jusqu'ici été trop souvent négligée. Nous conservons comme système spatial la projection perspective communément utilisée en synthèse d'images, interdisant ainsi, par exemple, la génération d'images cubistes à partir de scènes 3D géométriquement réalistes. Toute stylisation liée au système spatial devra avoir été préalablement intégrée à la modélisation pour pouvoir exister dans le rendu. Le contrôle sur le système de marques est, quant à lui, réduit à son strict minimum (chargement de textures) et ne fait pas l'objet d'une étude dans ce document. Offrir un contrôle flexible sur ces deux systèmes ouvrirait des perspectives artistiques excitantes et constitue également un axe de recherche très intéressant que nous laissons pour le moment en suspens.

■ *Nous nous concentrons sur la génération d'images statiques*

Finalement, nous avons choisi, dans un premier temps, de nous focaliser sur la génération d'images statiques plutôt que de séquences animées ou interactives. En effet, pour animer des rendus stylisés, il est essentiel de gérer la cohérence des marques d'une image à l'autre afin d'éviter l'inconfort visuel dû à de multiples changements intervenant sur les traits entre deux images. Les travaux portant sur ce sujet [KDMF03, Bou98] ont montré qu'il existe nécessairement une contradiction entre le respect du style à chaque image et le maintien de

la cohérence temporelle. Assurer la cohérence temporelle signifie donc réaliser un compromis entre respect du style et confort visuel. Il nous a semblé nécessaire de commencer par étudier la modélisation de style dans un cadre statique avant de l'étendre à l'animation qui pose des problèmes à la fois spécifiques et difficiles. Au chapitre 8, nous donnons de manière informelle nos idées quant à une telle extension.

Approche choisie

Nous choisissons une approche programmable

L'approche que nous avons choisie découle du postulat énoncé précédemment. L'expression d'un style comme la mise en relation d'un ensemble d'attributs avec un ensemble d'informations évoque fortement les systèmes basés sur les *shaders* programmables utilisés en synthèse d'images traditionnelle, tels que *Renderman* [Ups89, AG99, HL90, Coo84] : en rendu réaliste, les attributs (en pratique la couleur, la position, *etc.*...des points) sont calculés comme une fonction d'un certain nombre d'informations génériques communes à toute scène 3D (*e.g.* la normale au point, la position de la lumière), et, tandis que les systèmes traditionnels offrent un nombre fini de modèles pour cette fonction, la force de *Renderman* est de permettre à l'utilisateur de spécifier lui-même (sous forme d'une procédure) la fonction qu'il désire. On obtient ainsi un système parfaitement flexible, dans la mesure où les relations les plus exotiques peuvent y être programmées, et *automatique*, dans la mesure où le *shader* contenant ces relations s'appuie sur des informations génériques et peut donc être utilisé pour rendre n'importe quelle scène. Le rendu photoréaliste a énormément gagné en puissance grâce à l'introduction de telles approches et il nous semble qu'elles seraient d'autant plus profitables au rendu non-photoréaliste que celui-ci exige même davantage de flexibilité.

L'élaboration d'une telle approche pour le NPR présente plusieurs difficultés, dont la première est d'identifier un ensemble d'informations pertinentes dans le contexte de la description de style. En outre, tout comme les fonctions d'apparence ont pu être isolées du reste de l'algorithme de rendu avec *Renderman*, le processus de dessin doit être décomposé en un ensemble d'opérations élémentaires qui constitueront une interface de programmation pour la description de style. Finalement, il est indispensable, pour mener à bien ces étapes, de comprendre le processus de dessin et plus particulièrement le concept de style. Ainsi, une "formalisation" du style constitue un pré-requis à cette approche.

Contributions

Notre contribution générale est de proposer une approche programmable pour la modélisation de style dans le rendu non-photoréaliste de dessins au trait.

Plus précisément, nos contributions portent sur :

- la validation du postulat selon lequel le style d'un dessin dépend des informations contenues dans la scène
- l'ajout d'attributs de style essentiels, à la base communément utilisée en NPR
- l'identification d'un ensemble d'informations utiles pour la modélisation de style
- la spécification de mesures de la densité de lignes permettant d'élaborer des stratégies complexes de simplification automatique de dessin au trait
- une "formalisation" du processus de dessin et de mise en style sous la forme d'un pipeline d'opérations dédiées à la description de style

1.4 Organisation du document

Le prochain chapitre passe en revue un certain nombre de travaux présentant le domaine de recherche du NPR et plus particulièrement ceux traitant du style.

Dans le chapitre 3, nous étudions des illustrations réelles et tâchons d'exprimer le style de chacune en terme de mise en relation entre attributs de style et informations. Ceci permet de démontrer le bien fondé de notre postulat de départ et sert également d'introduction informelle à l'approche que nous avons développée.

Le chapitre 4 liste l'ensemble des informations que nous avons identifiées comme utiles dans le cadre de la description de style. En outre, nous y décrivons le *graphe de vue*, qui est la structure au sein de laquelle les informations sont organisées et au travers de laquelle elles sont accessibles.

Le chapitre 5 présente les outils de description de style. Nous y proposons une décomposition du processus de dessin sous forme d'une série d'opérations (*e.g.* sélection, chaînage, découpage, *shading*). Ces opérations servent à construire les traits qui constitueront le dessin final à partir des arêtes du graphe de vue. L'utilisateur programme un style à l'aide de ces opérations.

Dans le chapitre 6, nous examinons plus spécifiquement un problème classique du rendu de dessin au trait : le trop grand nombre de lignes. Ce problème, inhérent à la nature même de la technique, se présente dès que des objets, dont la taille de projection dans le dessin est trop petite, sont dessinés. De nombreuses stratégies de simplification ont été introduites par les artistes et nous servent d'exemples. Nous utilisons plus particulièrement une information pour ce traitement : la densité du dessin. Cette information ainsi que les différents éléments qui y sont liés apparaissent brièvement dans les chapitres précédents, mais sont illustrés et présentés de manière plus détaillée dans le chapitre 6.

Dans le chapitre 7, nous présentons les résultats obtenus avec le système développé en suivant notre approche. Ces résultats consistent, d'une part, en des exemples de code de descriptions de style, qui permettent de concrétiser les idées quant au travail à fournir par l'utilisateur pour spécifier un style, et d'autre part, en des illustrations générées par le système.

Finalement, nous concluons et discutons les décisions relatives à notre approche dans le chapitre 8.

1.5 Vocabulaire

Certains des termes que nous emploierons souvent dans ce manuscrit peuvent revêtir différentes significations selon le contexte dans lequel ils sont rencontrés et, comme nous les utilisons parfois dans un sens précis, il est préférable de les définir dès le début.

> **Trait :** C'est une marque laissée par l'outil (*e.g.* pinceau, crayon) sur le support (*e.g.* toile) en un seul geste de l'artiste. Un trait commence là où l'artiste pose son outil et s'arrête là où il le lève.

> **Ligne de silhouette :** Pour un point de vue donné, c'est la ligne sur une surface où l'orientation de cette surface par rapport au point de vue (vers le point de vue ou dans la direction opposée au point de vue) change. On peut avoir plusieurs lignes de silhouette par objet.

Contour : C'est le sous-ensemble de la silhouette qui marque la limite de l'objet. Dans le plan image, exactement une des deux régions séparées par un contour n'est pas incluse dans la projection de l'objet délimité par ce contour.

Lignes caractéristiques : Ce sont l'ensemble des lignes habituellement dessinées pour représenter une scène donnée et qui n'incluent pas les lignes d'ombrage (*e.g.* hachures). Parmi les lignes caractéristiques, on trouve par exemple les lignes de silhouette ou les arêtes vives.

Dessins de lignes : Ce sont des dessins contenant uniquement des lignes caractéristiques

chapitre 2

État de l'art

Dans ce chapitre, nous présentons les travaux antérieurs associés à notre recherche. La première partie est consacrée à donner un aperçu global du domaine de recherche du NPR. Les articles majeurs illustrant les différents aspects du NPR y sont cités et brièvement décrits. En outre, cette partie nous donne l'occasion d'indiquer où s'inscrit notre recherche dans le contexte global du NPR. Les deux parties suivantes décrivent des techniques de base nécessaires au développement de notre application. Finalement, dans la dernière partie, nous étudions des travaux de recherche s'étant fixés le même but que nous, en insistant sur leurs points forts et leurs points faibles ainsi que les similarités et les différences avec notre approche.

2.1 Contexte général

Le* NPR *est vaste et englobe même le rendu photoréaliste

La multitude de techniques et de styles qui existent en art pictural font du NPR un domaine de recherche aussi vaste que varié. En effet, comme nous l'avons souligné dans le chapitre précédent, par opposition à la photographie, l'extrême liberté qu'offre le dessin dans la représentation se traduit en une très grande variété d'aspects (incluant notamment l'aspect photoréaliste !). Pour inclure le NPR dans la synthèse d'images, il est donc nécessaire d'étendre les techniques de cette dernière : la projection perspective comme système spatial ou l'utilisation du point comme système de marques, qui sont caractéristiques du rendu photoréaliste, peuvent être vus comme des cas particuliers des quatre systèmes généraux introduits par Willats et Durand [WD05, Dur02a, Wil97a]. Le NPR, défini à tort comme l'opposé du rendu photoréaliste, est en réalité un domaine de recherche plus vaste qui inclut même le photoréalisme.

Toutefois, depuis sa création, le NPR s'est principalement concentré sur le développement de techniques de rendu permettant de produire des images imitant l'aspect de dessins ou de peintures faites à la main. Ainsi, les articles qui y sont catégorisés peuvent, par exemple, traiter de la simulation du tracé d'un couple outil/medium sur un support, de la génération d'images évoquant des peintures ou des dessins, à partir de scènes 3D ou de photographies. On y trouve également des publications traitant des questions relatives à l'animation de telles illustrations, et qui s'intéressent en particulier au développement de techniques permettant d'assurer la cohérence temporelle des traits dessinés [KDMF03]. D'une manière générale, pour retranscrire la liberté caractéristique des arts plastiques, les applications de NPR doivent

offrir à l'utilisateur un contrôle plus important que celui généralement disponible dans les applications de rendu photoréaliste. Deux ouvrages, l'un par Gooch et Gooch [GG01] et l'autre par Strothotte et Schlechtweg [SS02], fourniront au lecteur intéressé une vision globale du domaine du NPR.

Les régions et les lignes sont les deux grandes familles d'éléments constitutifs d'un dessin

En NPR, plus qu'en art traditionnel, on fait la distinction entre les différents éléments constitutifs du dessin, tels que les lignes caractéristiques, d'un côté, et les régions, de l'autre. En effet, pour la génération d'un dessin au trait par exemple, les techniques impliquées dans la détection et le dessin des lignes caractéristiques sont très différentes de celles employées pour la détection et le remplissage de régions, bien que dans les deux cas, la même primitive de trait soit utilisée pour le tracé (il s'agit de alors de traits de hachures pour les régions). On distingue justement ces deux types d'éléments (régions et lignes caractéristiques) pour organiser la suite de notre état de l'art.

2.1.1 Les régions

Dans un dessin, les régions correspondent aux projections des surfaces dans l'image, selon le point de vue considéré. La problématique réside alors dans le "remplissage" de telles régions. À la différence du rendu photoréaliste pour lequel l'unique primitive de marque est le point, le NPR varie les primitives de marques avec les styles. Un style imitant la peinture à l'huile utilisera des marques simulant des coups de pinceau pour remplir les régions, tandis qu'un autre, imitant le fusain, emploiera la technique des hachures et tracera des lignes. Contrairement à la synthèse d'images réalistes et à l'utilisation de points comme système de marques unique, la taille ou l'orientation des marques dans un rendu NPR contribuent pleinement à l'aspect de l'image finale.

Les difficultés relatives au remplissage de régions peuvent par exemple justement concerner la détermination des valeurs adéquates pour la taille et l'orientation des marques, en fonction des zones représentées, ou encore, le respect d'un ton dans le cas d'un remplissage à l'aide de hachures. La cohérence temporelle des marques dans le cadre d'une animation fait également partie des complications induites par cette nouvelle dimensionnalité des marques (1 et non plus 0 comme en rendu photoréaliste).

De nombreux travaux de recherches ont porté sur le rendu de régions dans un style ou dans l'autre, et nous en présentons ici quelques uns de manière à donner une idée du type de difficultés associées à cet objectif. Nous présentons en premier les travaux considérant des images en entrée, puis ceux utilisant des scènes 3D.

À partir d'images

Parmi les différentes techniques artistiques de remplissage de régions, la peinture (à l'huile notamment) et les hachures ont été les plus souvent imitées en NPR.

Style peint (*painterly*) En 1990, Haeberli *et al.* [Hae90] proposent un premier système interactif de rendu non-photoréaliste de peintures à partir d'images sources. L'utilisateur peint directement sur l'image et peut être assisté par le système à plusieurs niveaux. La couleur, par exemple est déterminée automatiquement en fonction de l'image source. La direction des coups de pinceau peut également être laissée à la charge du système, qui s'appuiera par

exemple sur le gradient de l'image source. L'épaisseur des marques est réglée par l'utilisateur de manière à à permettre le remplissage de fonds aussi bien que l'ajout de détails. La figure 2.1 (a) montre un exemple de résultat généré par cette technique.

En 1998, Hertzmann *et al.* [Her98] proposent un système complètement automatique pour générer des illustrations imitant la technique de la peinture à l'huile, à partir de photographies. L'utilisateur spécifie une série de tailles de pinceaux qui seront chacune utilisée dans une couche différente du rendu, les couches étant affichées séquentiellement, en commençant par celle correspondant à la taille de pinceau la plus importante. Les couches correspondant aux pinceaux plus fins sont donc superposées aux couches plus grossières, permettant ainsi l'ajout naturel de détails. Les supports des coups de pinceau sont des B-splines dont la direction est définie par le gradient de l'image originale. Ici encore, la couleur des coups de pinceau est extraite de l'image originale. Un exemple de résultat est montré sur la figure 2.1 (b).

(a) (b)

FIG. 2.1 – Générations de peintures à partir d'images

(a) : Cette image a été obtenue à partir d'une photographie sur laquelle l'utilisateur a "peint" interactivement. Un certain nombre d'attributs des coups de pinceau (e.g. leur couleur) sont automatiquement déterminés à partir des informations de la photographie. © *Haeberli et al. [Hae90]. (b) : Cette image a été générée automatiquement à partir d'une photographie : plusieurs calques sont superposés, chacun reproduisant la photographie originale à l'aide de coups de pinceau d'une certaine taille. On utilise des tailles de pinceaux décroissantes de manière à ajouter des détails par dessus les couches plus grossières.* © *Hertzmann et al. [Her98].*

Hachures Salisbury et Salesin ont énormément contribué au développement de techniques permettant la génération de dessins à base de hachures à partir d'images, en publiant trois articles sur le sujet dès 1994. Nous présentons donc brièvement ces publications avant de décrire deux autres techniques originales plus récentes.

En 1994, Salisbury *et al.* [SABS94] présentent un système interactif permettant de créer des dessins au trait à base de hachures à partir d'images en niveaux de gris. L'utilisateur dispose de "pinceaux" de haut niveau avec lesquels il applique des textures de traits. Le système assiste l'utilisateur en dessinant chaque trait et peut également les orienter en fonction du gradient de l'image de référence. La correspondance de tons est en revanche entièrement laissée à la charge de l'utilisateur qui, en choisissant une texture de trait, choisit aussi un ton. La figure 2.2 (a) montre un exemple de résultat obtenu avec cette approche.

Salisbury *et al.* améliorent ce système en 1996 [SALS96] en assurant un ton cohérent quelle que soit l'échelle à laquelle est générée l'illustration. Le système devient par ailleurs complètement automatique : les hachures sont maintenant automatiquement positionnées sur l'ensemble de l'illustration.

Finalement en 1997 [SWHS97], Salisbury *et al.* publient une nouvelle version de leur système dans laquelle l'utilisateur dispose d'une interface lui permettant de spécifier des champs de direction sur l'image originale à l'aide de laquelle les directions de courbures des objets présents peuvent par exemple être données. À partir d'un exemple de texture de traits et de l'image de référence en niveaux de gris, les hachures sont générées de manière à respecter les orientations définies et à correspondre aux tons initiaux. La figure 2.2 (b) illustre le principe de fonctionnement de ce système.

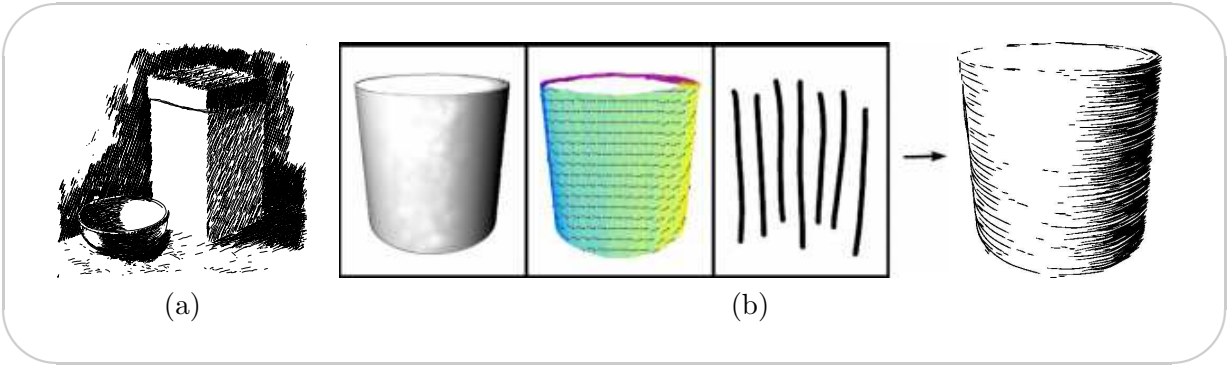


FIG. 2.2 – Génération d'illustrations à base de hachures à partir d'images (Salisbury *et al.*)

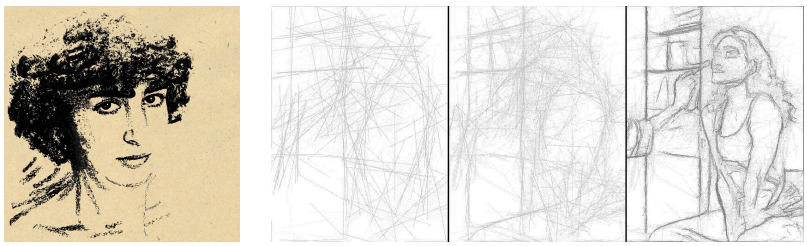
(a) : Exemple de résultat produit avec le système interactif de hachurage de Salisbury et al.. © Salisbury et al. [SABS94]. (b) : Principe de fonctionnement du nouveau système de Salisbury et al. : à partir d'une image de référence, d'une carte d'orientations ("peinte" par l'utilisateur) et du motif de hachures, l'illustration est automatiquement générée. © Salisbury et al. [SWHS97].

En 2001, Durand *et al.* [DOM+01] proposent également un système interactif permettant de produire une illustration à partir d'une photographie source. L'utilisateur dessine directement les traits sur l'image de référence et le système se charge d'attribuer à chaque trait la texture appropriée au niveau de gris cible. Par opposition avec le système de Salisbury *et al.* [SABS94], l'utilisateur peut ainsi se concentrer sur le positionnement des primitives sans se soucier de respecter le ton. Le résultat d'un dessin réalisé avec ce système est montré sur la figure 2.3 (a).

Plus récemment, dans un travail innovant, Semet *et al.* [SOD04] combinent des techniques d'intelligence artificielle avec des techniques de NPR pour générer (notamment) des dessins au trait à partir de photographies. La figure 2.3 (b) montre un résultat obtenu avec cette technique.

À partir de scènes 3D

Pour toutes ces techniques, il existe bien sûr des équivalents travaillant sur des scènes 3D. En pratique, les possibilités offertes par une approche utilisant la 3D sont plus grandes dans la mesure où la quantité d'informations disponibles y est plus importante.



(a) (b)

FIG. 2.3 – Génération d'illustrations à base de hachures à partir d'images

(a) : Image produite à partir d'une photographie avec le système de Durand et al. dans lequel, le placement des traits est laissé à la charge de l'utilisateur, tandis que la texture de ces traits est automatiquement déterminée par le système de manière à respecter les niveaux de gris de l'image initiale. © Durand et al. [DOM+01]. (b) : Semet et al. proposent de combiner les techniques d'intelligence artificielle avec celles de NPR *pour générer des illustrations : Cette séquence d'images montre le processus de création du dessin à l'aide d'une colonie de fourmis, chacune traçant un trait en s'appuyant sur l'information de gradient calculée dans l'image pour déterminer son chemin. © Semet et al. [SOD04].*

Style peint On peut noter que les techniques de Haeberli *et al.* ou Hertzmann *et al.* vues précédemment pourraient être utilisées avec des scènes 3D : il suffirait pour cela de faire un rendu normal de la scène et d'utiliser leurs algorithmes sur cette image. Il peut cependant être plus intéressant d'exploiter mieux l'information 3D. En particulier, dans le cadre d'une animation, les techniques précédentes ne permettent pas d'assurer la cohérence temporelle et souffrent notamment de l'effet "rideau de douche", où la stabilité, dans l'espace image, des coups de pinceau, d'une image à l'autre, donne l'impression que la scène est filmée à travers, justement, un rideau de douche fixe. En 1996, Meier [Mei96] résout ce problème en s'appuyant complètement sur la 3D pour produire un rendu imitant une peinture exécutée à l'aide de coups de pinceau : les supports des coups de pinceau 2D sont des particules 3D régulièrement réparties sur les maillages triangulaires. Ainsi lorsque le point de vue change, les coups de pinceau "collent" aux objets, offrant de cette manière une impression de cohérence temporelle plus naturelle. La figure 2.4 montre une image tirée d'une animation réalisée avec cette technique.

Hachures Pour le rendu à base de hachures, l'utilisation d'une scène 3D en entrée offre des possibilités plus nombreuses qu'avec une image : la plus grande quantité d'informations disponibles permet un contrôle plus poussé des attributs des traits de hachure (il est par exemple possible de s'appuyer sur les courbures 3D pour les orienter). En outre, la quantité de hachures à dessiner peut être déduite des calculs d'illumination classiques menés lors de rendus standard.

L'idée de base de ces approches est justement d'adpater les techniques traditionnelles de synthèse d'image au rendu par traits, en créant les hachures par le biais de textures (éventuellement procédurales). Les deux articles suivants utilisent cette approche, l'un dans le cadre d'un rendu interactif, l'autre pour la génération d'images fixes, et servent de références à ce type d'approches.

FIG. 2.4 – Rendu de peintures à partir de scènes 3D

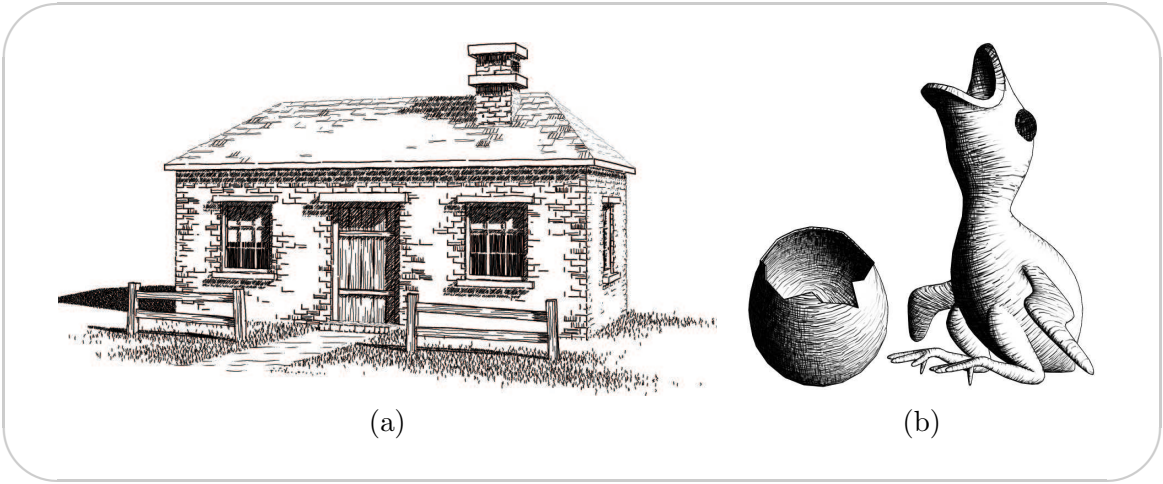
Cette image a été générée à partir d'une scène 3D. Les positions des coups de pinceau s'appuie sur des particules distribuées dans la scène 3D. Cette technique permet en particulier d'assurer la cohérence temporelle des marques au cours d'une animation. © Meier et al. [Mei96].

En 1994, Winkenbach *et al.* [WS94] sont les premiers à s'intéresser au rendu artistique de scènes 3D. Ils se focalisent plus spécifiquement sur la technique du dessin à l'encre et à la plume et travaillent sur des maillages polygonaux. Leur approche s'appuie sur des textures procédurales de traits qui permettent de générer des hachures dans les régions correspondant aux projections des surfaces de la scène, de manière à indiquer à la fois le ton, pour la mise en volume et l'ombrage, et le matériau (*e.g.* bois, briques, tuiles). Ils introduisent notamment l'idée de priorité de traits dans leur texture : les traits les plus visuellement importants sont dessinés en premier et les autres sont rajoutés de manière à assombrir le ton si besoin est. Un résultat obtenu à l'aide de leur approche est montré sur la figure 2.5 (a).

En 2001, Praun *et al.* [PHMF01] s'inspirent de ce travail et en réalisent une implémentation temps réel qui s'appuie sur les capacités du matériel graphique. Ils introduisent la notion de *tonal art maps (TAM)* qui sont les familles de textures de traits servant à représenter les différents tons à différentes échelles à l'aide de hachures. Contrairement au travail de Winkenbach *et al.* [WS94], les textures ne sont pas générées procéduralement pendant le rendu, mais au préalable, de manière à être stockées sur la carte graphique. Sur chaque triangle du maillage, six textures de la *TAM* sont mélangées de manière à assurer à la fois la cohérence spatiale (d'une face à l'autre) et la cohérence temporelle. L'utilisation du matériel graphique leur permet une visualisation interactive. On peut voir le résultat de leur approche sur la figure 2.5 (b).

2.1.2 Les lignes caractéristiques

Les travaux que nous avons cités jusqu'ici traitent du rendu de régions (zones 2D de l'image) dans un style non-photoréaliste à partir de scènes 3D ou d'images. Les lignes (éléments 1D) sont l'autre grand type d'éléments employés dans la représentation picturale. Comme nous l'avons expliqué dans le premier chapitre (section 1.3.2), c'est à ce type d'éléments que nous nous limiterons dans cette thèse. Bien que les lignes caractéristiques soient des lignes imaginaires, dans la mesure où elles n'ont pas d'existence physique dans la scène réelle, il



(a) (b)

FIG. 2.5 – Rendu de scènes 3D à l'aide de hachures

Ces deux illustrations ont été générées à partir de scènes 3D et s'appuient sur l'utilisation de textures pour le dessin des hachures. (a) : Winkenbach et al. utilisent ici des textures de traits procédurales. © Winkenbach et al. [WS94]. (b) : Grâce à un plaquage multiple de textures, Praun et al. parviennent à simuler les tons à l'aide de hachures tout en assurant une cohérence spatiale des traits. Ces techniques standard étant accélérées par les cartes graphiques, leur système permet un rendu interactif. © Praun et al. [PHMF01].

reste très naturel d'y avoir recours dans un dessin. En effet, dans la nature, nous percevons les formes par le biais de variations de texture et de ton et les limites des objets correspondent alors aux seuils de forts contrastes en couleur ou en intensité. Ce sont ces limites qui sont habituellement représentées par les artistes [Kø84]. De nombreuses recherches se sont penchées sur l'identification mathématique de ces lignes et on distingue aujourd'hui quatre familles de lignes qui, pour des raisons purement géométriques ou parce qu'elles sont le seuil de phénomènes lumineux, sont enclines à apparaître dans les dessins.

Les différents types de lignes caractéristiques

Caractériser les lignes signifie identifier de manière mathématique les points d'une scène 3D qui se projettent sur les lignes habituellement dessinées par un artiste. On peut noter l'incertitude inhérente à cette définition même, dans la mesure où des artistes différents ne tracent a priori pas exactement les mêmes lignes. Toutefois, un certain nombre de ces lignes semblent apparaître dans la plupart des dessins et la question est alors de savoir si ces dernières répondent à des critères (notamment géométriques) communs.

Aujourd'hui, l'identification précise des lignes caractéristiques dessinées par les artistes reste un sujet ouvert. S'il est généralement admis que ces lignes correspondent à des limites perçues par l'oeil, et pouvant être de nature purement géométrique (silhouette) ou incluant des considérations d'interactions lumineuses (limites des ombres), il n'existe qu'un faible consensus sur une telle caractérisation. Nous présentons ici les trois types de lignes qui sont le plus communément utilisés en NPR : les silhouettes, les crêtes (et vallées) et les contours suggestifs. Parmi ces trois familles, les silhouettes sont probablement celle pour laquelle la caractérisation est la plus objective.

Les lignes de silhouette Ce sont les lignes les plus communément représentées dans les dessins ; la figure 2.7 (a) illustre cette catégorie de lignes en les isolant d'un ensemble de lignes caractéristiques. La silhouette d'une surface correspond aux endroits de cette surface où celle-ci passe d'une position où elle est orientée face au point de vue à une position où elle est orientée dans l'autre sens. Mathématiquement, pour une surface vue depuis V, un point P de cette surface appartient à la silhouette si et seulement si :

$$(P - V).N = 0 \tag{2.1}$$

N étant la normale à cette surface au point P. L'ensemble des points P tels que $(P-V).N > 0$ correspond aux points où la surface est orientée face au point de vue tandis que l'ensemble des points P tels que $(P - V).N < 0$ correspond aux points où la surface est orientée dans l'autre sens. La figure 2.6 illustre ces grandeurs. On peut remarquer que la silhouette change avec le point de vue.

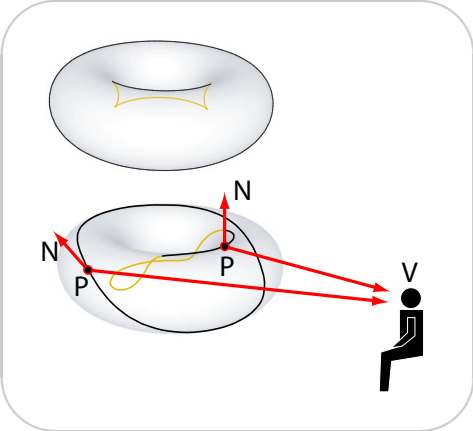


FIG. 2.6 – Natures des lignes : définition d'une silhouette

La silhouette d'une surface est définie comme l'ensemble des points P de cette surface, où la normale à la surface N est perpendiculaire à la direction de vue $(P - V)$, V étant le point de vue. En haut : pour le point de vue considéré, les silhouettes visibles de cet objet sont les lignes tracées en noir et les silhouettes cachées, celles tracées en orange. En bas : On regarde la scène d'un point de vue différent sans mettre à jour les silhouettes. Les lignes noires et oranges désignent respectivement les lignes de silhouette visibles et invisibles du point de vue précédent. On illustre la normale N et la direction de vue $(P - V)$ pour deux points quelconques de cette silhouette.

Remarque : Les définitions des termes "silhouette" et "contour" changent d'un article à l'autre dans la littérature. Lorsque nous parlons de nature des lignes, le contour désigne les lignes d'un objet qui, dans le plan, séparent l'intérieur de cet objet d'un autre objet ou du vide. La silhouette correspond aux lignes définies ci-dessus. Les lignes de contour sont un sous-ensemble des lignes de silhouette. Les figures 2.7 (b) et (c) illustrent la différence entre ces deux types de lignes. Les définitions des notions de "contour" et de "silhouette" ainsi que d'autres termes régulièrement utilisés dans cette thèse se trouvent dans la section 1.5 du chapitre 1

.

Remarque : Dans la plupart des cas, les silhouettes forment, sur la surface, des boucles 1D fermées. Ce n'est plus le cas dès qu'une surface plane est tangente au point de vue. Dans ce cas, on a toute une région 2D pour laquelle, en chaque point, $(P - V).N = 0$.

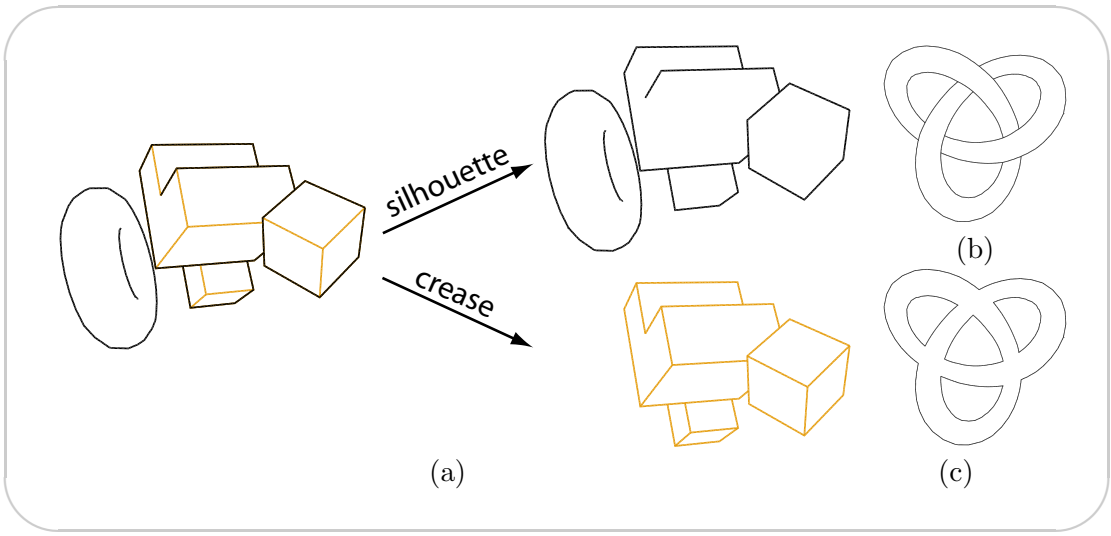


FIG. 2.7 – Natures des lignes : contour et silhouette

(a) : Les lignes caractéristiques de cette scène (à gauche) sont constituées des arêtes vives (à droite, en haut) et des silhouettes (à droite, en bas).
Sur une autre scène, les lignes de silhouette (b), et les lignes de contours (c).

Les crêtes et les vallées Les crêtes et les vallées sont définies de manière purement géométrique, indépendamment du point de vue. Elles correspondent aux points où les courbures principales passent par un extremum en suivant leur ligne de courbure. Pour les crêtes, les courbures sont positives et passent par un maximum, pour les vallées, elles sont négatives et passent par un minimum. La figure 2.8 (a) montre un exemple de telles lignes.

Courbures principales : Considérons un point P sur une surface $\mathcal{S}$. Soit N la normale à $\mathcal{S}$ en P et $\mathbf{t}$ un vecteur défini dans le plan tangent à $\mathcal{S}$ en P. La valeur κ de la courbure de $\mathcal{S}$ en P dans la direction $\mathbf{t}$, est égale à celle de la courbe définie comme l'intersection entre $\mathcal{S}$ et le plan contenant $\mathbf{t}$ et N au point P. $\kappa(\mathbf{t})$ est appelée la *courbure normale*. Les courbures principales maximum κ_{max} et minimum κ_{min} correspondent aux maximum et minimum de κ lorsqu'on balaye la famille de plans. Ces courbures principales sont définies respectivement par rapport aux directions $\mathbf{t_{max}}$ et $\mathbf{t_{min}}$.

Les contours suggestifs Les contours suggestifs ont été introduits en 2003 par DeCarlo *et al.* [DFRS03] et correspondent intuitivement aux points qui sont des "presque silhouettes", c'est-à-dire des points qui appartiendront à la silhouette sous des points de vue voisins. Les contours suggestifs peuvent être combinés aux contours de silhouette dans la mesure où ils les étendent ou les complètent. Ils sont définis comme l'ensemble des points de la surface pour

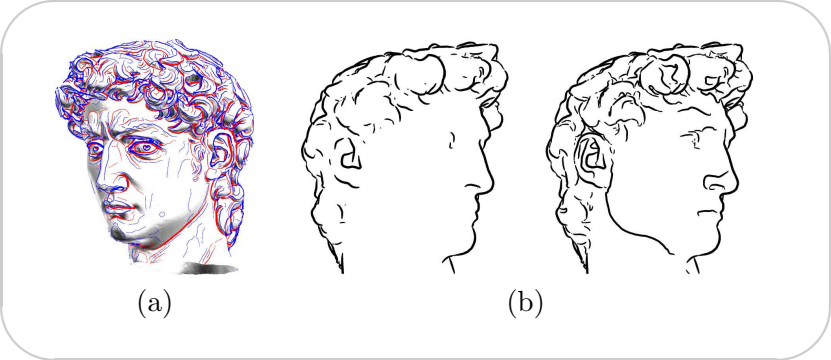


FIG. 2.8 – Types des lignes : crêtes et vallées

(a) : Les lignes de crête (bleu) et de vallée (rouge). © Ohtake et al. [OBS04]. (b) : à gauche, sans contour suggestif et avec, à droite. © DeCarlo et al. [DFRS03].

lesquels la courbure radiale change de signe et augmente en se rapprochant du point de vue. Les contours suggestifs sont illustrés par la figure 2.8 (b).

> **Courbure radiale :** La courbure radiale en P est la courbure normale de la surface en P dans la direction $\mathbf{w}$, $\mathbf{w}$ étant définie comme la projection du vecteur vue $(V - P)$ sur le plan tangent à la surface en P (*cf.* figure 2.9). La courbure radiale (et donc les contours suggestifs) dépend du point de vue.

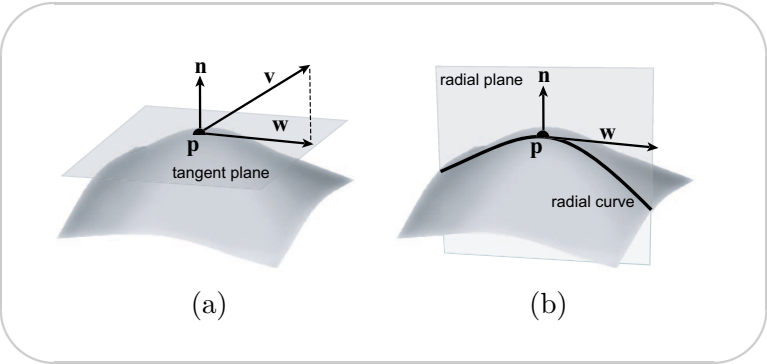


FIG. 2.9 – Courbure radiale

(a) : w est obtenu en projetant le vecteur de vue v sur le plan tangent. (b) : Le plan radial est formé par p, n et w et coupe la surface le long de la courbe radiale, dont la courbure en chaque point est la courbure radiale. © DeCarlo et al. [DFRS03].

Remarque : On remarque que les lignes de crêtes et de vallées et les contours suggestifs (ainsi que les lignes paraboliques) sont toujours des lignes proches les unes des autres qu'il est parfois difficile de différencier. Dans le cas d'une animation, les lignes de crêtes et de vallées (dont le calcul dépend uniquement de la géométrie) ont un aspect trop statique, qui évoque le plaquage de texture et qui nuit à la qualité visuelle du résultat. Il semble intéressant de définir des critères d'apparition et de disparition de ces lignes en fonction des positions du point de vue et des sources lumineuses.

Rendu direct de lignes caractéristiques

On appelle rendu direct de lignes caractéristiques l'ensemble des techniques qui permettent d'afficher une ou plusieurs familles de lignes caractéristiques à partir de scènes 3D, sans que ces lignes ne soient jamais calculées analytiquement ni explicitement manipulées par le programme. Ces techniques s'appuient la plupart du temps sur les fonctionnalités de la carte graphique.

Par opposition au rendu direct, le rendu analytique de lignes, que nous verrons en détail dans la suite de ce chapitre, calcule et manipule explicitement les lignes dans le programme.

Les techniques de rendu direct sont, dans l'ensemble, des variations autours d'un algorithme de base introduit sous une forme proche par Rossignac en 1992 [RvE92] pour les silhouettes, qui s'appuie fortement sur l'utilisation du *Z-Buffer* combinée avec un rendu multi-passes :

Algorithme 2.1 Rendu direct de silhouettes

Fonction *RenderSilhouette*
1. depth function ←*Less than*
2. **render** front facing polygons in the *Z-Buffer* only
3. depth function ←*Equal to*
4. **render** back facing polygons

Cet algorithme affiche une approximation polygonale des silhouettes au travers du sous ensemble des arêtes du maillage qui ont la propriété de partager une face orientée vers la caméra et une face orientée dans la direction opposée (cette propriété sera détaillée dans le prochain paragraphe). Cet algorithme a l'avantage d'être simple à implémenter et permet d'afficher les silhouettes de la plupart des modèles 3D en temps interactif. Il présente néanmoins de nombreux artefacts notamment dûs aux imprécisions numériques du *Z-Buffer*. Il est ainsi courant de déplacer légèrement les polygones vers la caméra avant la deuxième passe de rendu, comme le montre la figure 2.10 (a).

En 1999, Raskar *et al.* [RC99] améliorent la qualité du rendu en agrandissant les polygones orientés dans la direction opposée à la direction de vue (*backfacing*) (*cf.* figure 2.10 (b)).

L'algorithme est à nouveau modifié en 2001 par Raskar *et al.* [Ras01] de manière à afficher les arêtes et les vallées en plus des silhouettes. La technique s'appuie ici sur l'ajout de polygones pertinemment orientés au bout de certaines faces.

En 1999, Gooch *et al.* [GSG+99] proposent une autre méthode de rendu de silhouettes également complètement exécutée sur la carte graphique. Ils suggèrent qu'en utilisant une texture d'environnement dont le périmètre est colorié en noir, les silhouettes apparaîtront

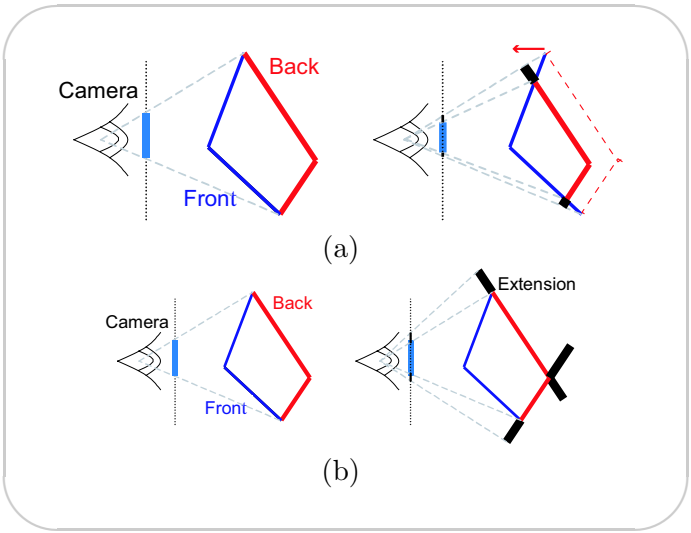


FIG. 2.10 – Rendu direct de silhouettes

(a) : Une première technique d'affichage des silhouettes consiste à avancer légèrement les faces orientées vers l'arrière (en rouge). Les portions de ces faces (en noir) qui passent devant les faces orientées dans l'autre direction (en bleu), deviennent visibles et dessinent les silhouettes. (b) : L'autre technique, introduite par Raskar et al. [RC99] consiste à agrandir ces mêmes faces plutôt que de les avancer. On obtient ainsi des lignes d'épaisseurs plus homogènes. © *Raskar et al. [RC99].*

naturellement, comme on peut le voir sur la figure 2.11. Cette technique offre un rendu intéressant mais avec un contrôle trop faible sur la régularité de l'épaisseur des lignes.

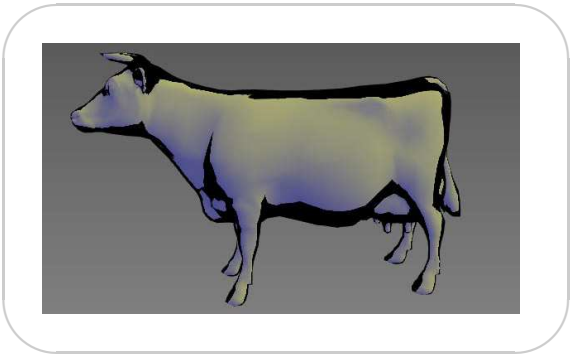


FIG. 2.11 – Rendu direct de silhouettes : utilisation d'une texture d'environnement

On utilise une texture d'environnement colorée avec des contours noirs pour calculer la couleur de chaque point du modèle. Cette technique permet d'obtenir des zones ombrées aux silhouettes. Son inconvénient réside dans l'irrégularité des silhouettes ainsi obtenues. © *Gooch et al. [GSG+99].*

D'une manière générale ces méthodes sont limitées quant au style qu'il est possible d'appliquer aux lignes. En effet, comme nous l'avons mentionné, la géométrie des traits n'est jamais explicitement manipulée par le programme et ceux-ci ne peuvent donc pas être altérés dans un objectif de stylisation. Pour cette raison les applications destinées à afficher les lignes de

manière stylisée utilisent une approche analytique dans laquelle le calcul des lignes est explicitement réalisé dans le programme et qui offre ainsi un meilleur contrôle au programmeur.

Jusqu'à récemment, le principal intérêt des approches directes résidaient dans l'utilisation de la carte graphique qui permettait un affichage interactif des silhouettes. Aujourd'hui, les cartes graphiques programmables permettent de déporter un calcul analytique des silhouettes (ou d'autres lignes d'ailleurs) du processeur central (*CPU*) vers le processeur graphique (*GPU*) et ainsi de mettre les approches analytiques à la portée du temps réel [MH04] et les approches directes entre parenthèses.

Rendu analytique de lignes caractéristiques

Dès que l'on veut afficher les lignes de modèles 3D de manière réellement stylisée, l'approche classique consiste à calculer les arêtes caractéristiques de manière analytique en s'appuyant sur les formules correspondant à chaque type de lignes (*cf.* section 2.2), à déterminer le sous-ensemble de ces arêtes qui sont visibles, à fabriquer des traits stylisés à partir de ces arêtes (en les chaînant et en affectant des attributs, de couleur ou d'épaisseur par exemple, aux traits ainsi construits) (*cf.* section 2.3) puis à afficher ces traits (*cf.* section 2.3.3). La figure 2.12 montre le pipeline commun à l'ensemble des applications ayant choisi cette approche.

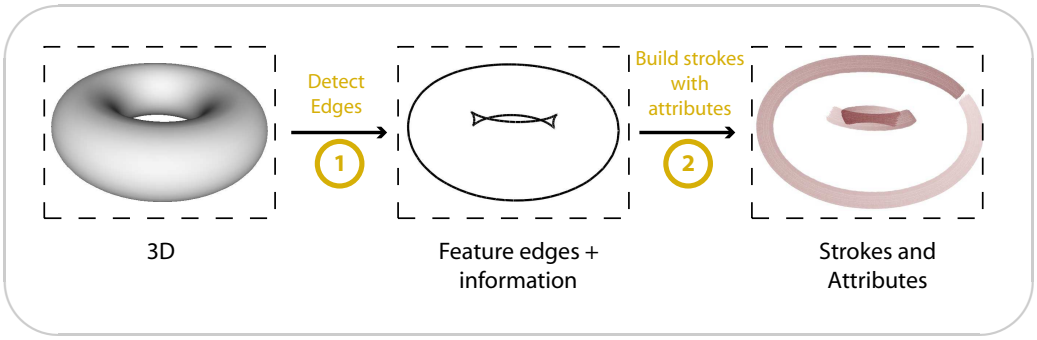


FIG. 2.12 – Rendu analytique de lignes : schéma de fonctionnement général

Les étapes standard d'une approche analytique du rendu de lignes caractéristiques sont la détection d'arêtes caractéristiques sur le modèle 3D, et la construction de traits stylisés (géométrie et attributs des traits).

Avec cette approche, la liberté de stylisation offerte au programmeur est théoriquement infinie : les traits étant explicitement manipulés dans le programme, toute opération (*e.g.* déformation, coloration), aussi complexe soit-elle, peut leur être appliquée. Comme nous le verrons, la littérature regorge de nombreux exemples de styles visuels réalisés en suivant cette approche. L'objectif d'une application NPR est, d'une part de produire des rendus stylisés, mais également, d'autre part, d'offrir à l'utilisateur un maximum de contrôle sur les styles des rendus. La difficulté réside alors dans le transfert du contrôle offert au programmeur vers l'utilisateur.

Si l'on se réfère au schéma de la figure 2.12, l'étage de stylisation porte le label 2. C'est donc à cet étage que l'utilisateur doit avoir un accès flexible pour pouvoir contrôler le style du rendu.

Nous utilisons ce schéma de fonctionnement comme cadre de travail pour notre approche et tâchons donc d'exposer, dans notre système, le maximum de fonctionnalités du composant de construction des traits. Dans la suite de ce chapitre, nous décrivons plus en détail les travaux relatifs à ce schéma de fonctionnement : les prochaines sections présentent les nombreux travaux relatifs aux deux composants de ce pipeline avant de se pencher plus précisément sur la recherche antérieure traitant du contrôle sur la stylisation des lignes en NPR, c'est-à-dire du contrôle sur l'étage 2.

Résumé: Les techniques de NPR sont extrêmement variées : elles peuvent prendre en entrée des images ou des scènes 3D, et produire en sortie des images stylisées, imitant par exemple, la peinture ou le dessin au trait. On distingue deux grands types d'éléments dans les rendus NPR : les régions et les lignes (*e.g.* silhouette, vallées). Notre application travaille sur une scène 3D et se focalise sur le rendu stylisé de lignes. Pour ce faire, nous choisissons une approche analytique de calcul des traits, qui permet une totale liberté dans leur stylisation.

2.2 Détection analytique de lignes caractéristiques

La détection des lignes caractéristiques correspond au premier étage du pipeline représenté sur la figure 2.12 et constitue un passage obligé pour toute application de rendu non-photoréaliste. Pour cette raison, c'est l'un des champs de recherche les plus actifs du NPR. Nous avons vu précédemment les grandes familles (identifiées) de lignes caractéristiques habituellement représentées par les artistes, dans cette section nous passons en revue les principales techniques dédiées à la détection de telles lignes sur des modèles 3D. Nous commençons par présenter les approches travaillant dans l'espace image puis nous décrivons celles employant des techniques purement géométriques appliquées dans l'espace objet. Il est important de remarquer que, bien que l'on parle d'"espace image" pour la première catégorie, les données d'entrée sont 3D : les "images" sont obtenues en rendant la scène d'une manière ou d'une autre.

2.2.1 Espace image

Détecter les lignes d'un modèle 3D dans l'espace image signifie qu'on effectue un ou plusieurs rendus de la scène et que le travail de détection à proprement parler se fait à l'aide d'outils issus du traitement d'images, sur les images dans lesquelles sont stockés ces rendus.

En 1990, Saito *et al.* [ST90] introduisent les *G-buffers (Geometric Buffers)* : ce sont des images dans lesquelles des informations géométriques ponctuelles (*e.g.* profondeur, normales) relatives à la projection de la scène 3D sont accessibles. Saito *et al.* suggèrent que les lignes caractéristiques des objets sont plus aisément détectables dans certains de ces *G-buffers*. En particulier, ils travaillent sur la carte des profondeurs dans laquelle les lignes de silhouette d'une part, et les lignes de crêtes et de vallées d'autre part, sont respectivement des discontinuités d'ordre 0 et d'ordre 1 (*cf.* figure 2.13 (a)).

Curtis [Cur98] travaille également sur la carte des profondeurs pour détecter les lignes de silhouette à l'aide d'un filtre de type gradient. La carte de gradients obtenue sert par ailleurs à

diriger des particules d'encre émanant des zones de silhouette. En contrôlant la perturbation subie par ces particules on obtient différentes gestuelles de tracé artistique.

En 1996, Decaudin [Dec96a] remarque que la détermination des discontinuités de premier ordre peut se révéler instable et propose de travailler sur la carte des normales (figure 2.13 (b)) pour détecter les lignes de crêtes et de vallées, celles-ci y correspondant en effet à des discontinuités d'ordre 0.

Finalement, en 1998, Corrêa *et al.* [CJTF98] utilisent la *carte des uv* (figure 2.13 (c)), qui stocke sous forme de couleur les coordonnées paramétriques de chaque sommet, pour détecter les silhouettes : en combinant les informations de couleur et de profondeur, on peut déterminer la nature (*e.g.* silhouette, bord) de chaque arête séparant deux pixels et ainsi construire un graphe connectant les bords de pixel de même nature.

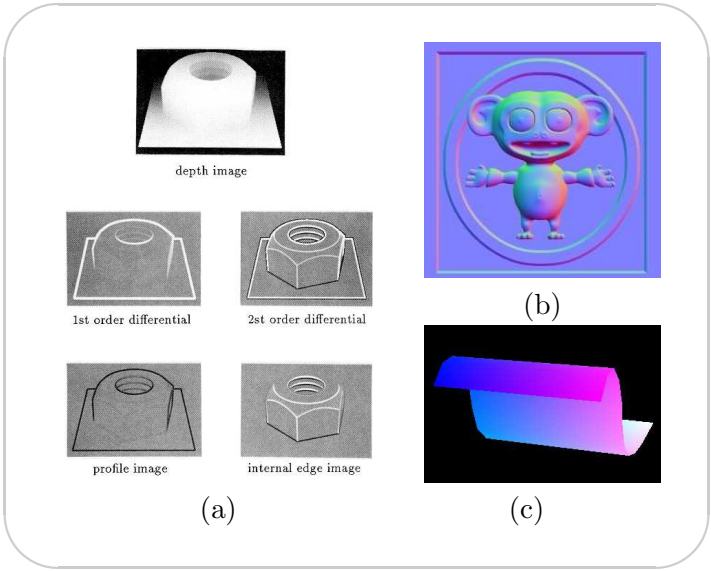


Fig. 2.13 – Détection de lignes caractéristiques : *G-buffers*

(a) : Saito et Takahashi utilisent la carte de profondeur pour calculer les arêtes caractéristiques : des filtres différentiels d'ordres 1 *et* 2 *permettent respectivement de déterminer les lignes de silhouette et de crêtes (et vallées). © Saito et Takahashi [ST90]. (b) : Une carte de normales. Les lignes de crêtes et de vallées peuvent être calculées sur ce type d'image avec des filtres dérivatifs d'ordre* 1. *© Blender 2004 - www.blender.org. (c) : La carte des* uv *d'un modèle de tapis. Cette carte permet de détecter les lignes de silhouette. © Corrêa et al. [CJTF98].*

Il existe plusieurs avantages à détecter les lignes dans l'espace image. Premièrement, le domaine de la vision par ordinateur nous fournit quantité de détecteurs de lignes, souvent aisés à implémenter, qu'il suffit de réutiliser. Ensuite, d'un point de vue perceptuel, il semble plus pertinent de travailler après projection, c'est-à-dire sur ce que l'on voit vraiment, que dans l'espace objet. Finalement, comme nous le verrons dans la prochaine section, la stylisation de lignes nécessite d'abord d'être capable de construire de longues chaînes à partir de l'ensemble des morceaux de lignes détectées et ensuite de paramétrer régulièrement cette chaîne. Or, contrairement à l'espace objet, la question de la paramétrisation ne pose pas de problème particulier dans l'espace image.

La détection de lignes dans l'espace image présente également plusieurs défauts. La précision, par exemple, est limitée à la résolution des images, et cela peut se révéler insuffisant dans le cas d'objets complexes. Ensuite, si l'on veut associer aux lignes des informations extraites de la scène 3D, il faut passer par une projection inverse coûteuse et imprécise. Pour ces raisons, il nous semble plus intéressant de détecter les lignes à l'aide d'outils géométriques travaillant dans l'espace objet 3D.

2.2.2 Espace objet

Nous avons vu que les techniques de détection de lignes caractéristiques opérant dans l'espace image manipulent des pixels, c'est-à-dire qu'elles s'appliquent dans un espace discret. Par contraste, travailler dans l'espace objet signifie que l'on travaille dans un espace 3D continu. Le type de ces éléments 3D et les opérations qu'on leur applique varient en fonction de la nature de la représentation utilisée pour la scène 3D. On identifie quatre types principaux de représentation surfacique pour les modèles 3D :

Les maillages polygonaux Dans cette représentation, une surface est approchée par un ensemble de polygones (*e.g.* triangles, quadrilatères) connectés. Cette représentation est en particulier adaptée à un affichage utilisant la carte graphique.

Les surfaces paramétriques Une surface paramétrique est définie par des points de contrôle (de manière similaire à des courbes paramétriques, telles que les *splines* par exemple) : tout point de la représentation est défini par interpolation polynomiale d'un sous-ensemble des points de contrôle. Ce type de représentation permet notamment une édition interactive aisée et une obtention directe des propriétés différentielles (*e.g.* normale, courbure). Il est courant de convertir les surfaces paramétriques en maillages polygonaux pour le rendu.

Les surfaces de subdivision À mi-chemin entre les maillages polygonaux et les surfaces paramétriques, les surfaces de subdivision sont définies de manière itérative à partir d'un maillage polygonal de contrôle : ce dernier est une version grossière de l'objet et le schéma d'itération est un schéma de raffinement qui permet d'en obtenir une forme lisse. Les surfaces de subdivision permettent une représentation régulière (à la limite de l'itération), sont bien adaptées à une édition interactive ainsi qu'au rendu (dans la mesure où il s'agit toujours d'un maillage polygonal).

Les surfaces implicites Les surfaces implicites sont une représentation indirecte d'une surface : elles sont définies comme la surface iso-potentielle d'une fonction de potentiel : $S = \{P(x, y, z) | f(x, y, z) = 0\}$. Cette représentation est, en particulier, bien adaptée à l'utilisation d'opérations CSG et à l'animation par déformation. En revanche, elle est peu adaptée à un affichage interactif dans la mesure où le passage d'une surface implicite à un maillage polygonal est fastidieux.

Les maillages polygonaux constituent la représentation la plus populaire à une application de rendu et sont, pour cette raison, la représentation la plus courante en synthèse d'images ; nous nous focaliserons donc sur celle-ci. Les méthodes applicables dans ce cas de figure restent valables pour les surfaces de subdivision. Le lecteur pourra se référer aux travaux de Bremmer *et al.* [BH98] et de Elber [Elb98] pour la détection de lignes dans le cas de surface de subdivision et de surface paramétrique. Il existe d'autres types de représentations (*e.g.* par points, par voxels) que nous n'avons pas cités ici. Toutefois, pour la plupart des représentations, des

travaux s'intéressant à la détection des lignes caractéristiques existent. Xu *et al.* [XC04], par exemple, détectent les silhouettes dans le cas d'une représentation par nuage de points.

Pour beaucoup des techniques que nous présentons, il est nécessaire que les surfaces soient des *variétés* (*manifold* en Anglais) de dimension 2. Il faut également pouvoir disposer de l'information de connectivité du maillage (*e.g.* la listes des arêtes issues d'un sommet donné). On pourra par exemple construire une structure *Winged-Edge* [Gla91] à partir d'une "soupe" de polygones dans une phase de pré-traitement.

> **Rappel sur les variétés :** Intuitivement, une surface est une variété de dimension 2 si en tout point elle est localement topologiquement équivalente à un disque. Pour un maillage polygonal, cela revient à dire que chaque arête est bordée par exactement deux faces.

Nous décrivons maintenant plusieurs techniques géométriques permettant de détecter les différents types de contours vus dans la sous-section 2.1.2.

Lignes de bord

Les arêtes de bord correspondent aux arêtes du maillage qui ne sont bordées que par une seule face (au lieu de deux). Ce type de lignes n'existe pas dans le monde réel (et n'a pour cette raison pas été introduit dans la classification) dans la mesure où il implique l'existence d'objets d'épaisseur nulle. On le retrouve cependant souvent en informatique graphique où des surfaces ouvertes sont souvent utilisées pour modéliser des objets de fine épaisseur (*e.g.* une feuille de papier).

D'une certaine manière ces arêtes sont un cas particulier d'arêtes de silhouette, puisqu'elles correspondent à un changement infiniment rapide d'orentation de la surface par rapport au point de vue, et doivent être affichées. Pour les détecter, il suffit de parcourir l'ensemble des arêtes du maillage et de repérer celles qui ne sont bordées que par une seule face :

Algorithme 2.2 Détection des arêtes de bords

Fonction *DetectBorders*
Entrée: L'ensemble $\mathcal{A}$ des arêtes du maillage
Sortie: Le sous-ensemble $\mathcal{B}$ de $\mathcal{A}$ correspondant aux arêtes de bord
1. **pour** $a \in \mathcal{A}$
2. **si** le nombre de faces adjacentes à a est égal à 1
3. **alors** $a \in \mathcal{B}$

> **Remarque :** Les surfaces comportant des arêtes de bord ne sont pas des variétés. Les techniques de détection de lignes ainsi que les structures associées (*e.g. Winged-Edge*) peuvent cependant être facilement adaptées pour les prendre en compte. Dans la suite, les surfaces considérées sont donc des variétés ou des surfaces comportant des bords (*i.e.* les représentations polygonales associées possèdent donc des arêtes ayant une ou deux faces adjacentes.)

Lignes de silhouette

Nous avons vu précédemment (*cf.* sous-section 2.1.2) que la silhouette d'une surface passe par les points où la normale est perpendiculaire à la direction de vue. Avec les maillages polygonaux on ne dispose que d'une discrétisation surface exacte, et donc, que d'une approximation linéaire par morceaux de cette dernière. La détection des points de silhouette n'est donc pas directe dans la mesure où il est très peu probable que celle-ci passe exactement par les sommets du maillage (*i.e.* les échantillons de la surface). La majorité des techniques se contentent toutefois de calculer une approximation grossière de la silhouette en déterminant l'ensemble des arêtes du maillage qui sont bordées par une face vue de devant et une face vue de derrière (*cf.* figure 2.14). Malgré sa grande imprécision, cette approximation se révèle souvent suffisante dans la mesure où les plus grands écarts, par rapport à la silhouette exacte, se trouvent dans des plans tangents à la direction de vue, et disparaissent donc lors de la projection dans l'image (nous verrons un peu plus loin qu'il existe tout de même une technique [HZ00] permettant de calculer une meilleure approximation de la silhouette). L'algorithme de base (approche *force brute*) permettant la détermination des arêtes de cette approximation polygonale de la silhouette consiste à parcourir dans un premier temps l'ensemble des faces du maillage afin de les marquer *avant* ou *arrière* selon leur orientation par rapport au point de vue, puis, dans un deuxième temps, à parcourir l'ensemble des arêtes du maillage afin de détecter celles bordées à la fois par une face *avant* et une face *arrière*.

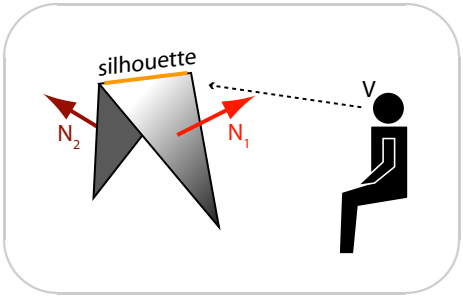


Fig. 2.14 – Approximation polygonale des silhouettes

L'approximation polygonale d'une silhouette est constituée des arêtes du maillage qui sont adjacentes à la fois à une face orientée vers le point de vue et à une face orientée dans la direction opposée. En suivant cette définition, sur le schéma, l'arête orange est une arête de silhouette.

Cet algorithme nécessite le parcours de l'ensemble des faces et des arêtes du modèle pour que la silhouette soit complètement détectée et a donc une complexité linéaire. Pour les applications visant le temps réel, il peut se révéler un goulot d'étranglement dans le cas de gros modèles. De nombreuses recherches ont porté sur son accélération. Ces travaux adoptent soit des approches "avec perte", avec lesquelles la silhouette obtenue est potentiellement incomplète, soit des approches "sans perte", qui assurent un résultat équivalent à la méthode *force brute*,e mais dont l'implémentation est plus complexe. Pour une application ne visant pas le temps réel, il est acceptable d'utiliser l'approche *force brute* dont la complexité linéaire garantit tout de même des temps de calcul raisonnables. Bien que nous ayons fait ce choix, il nous a semblé intéressant de passer ici en revue quelques unes des techniques d'accélération pour la détection de silhouette.

Algorithme 2.3 Détection de silhouette

Fonction *DetectSilhouette*

Entrée: V le point de vue, L'ensemble $\mathcal{F}$ des faces du maillage, L'ensemble $\mathcal{A}$ des arêtes du maillage

Sortie: Le sous-ensemble $\mathcal{S}$ de $\mathcal{A}$ appartenant à l'approximation de la silhouette

1. **pour** $f \in \mathcal{F}$
2. $N \leftarrow$la normale à f
3. $P \leftarrow$un point de f
4. **si** $(P - V).N > 0$
5. **alors** l'orientation de f est *avant*
6. **sinon** l'orientation de f est *arrièrre*
7. **pour** $a \in \mathcal{A}$
8. f_a, $f_b \leftarrow$les faces adjacentes à a
9. **si** les orientations de f_a et f_b sont opposées
10. **alors** $a \in \mathcal{S}$

Markosian *et al.* [MKT+97] proposent une approche "avec perte" : ils déterminent les arêtes de silhouette en temps interactif à l'aide d'une approche probabiliste. Ils s'appuient sur le fait que seul un faible pourcentage des arêtes du maillage appartient à la silhouette, et que celle-ci est le plus souvent constituée de longues chaînes d'arêtes : en examinant une petite fraction des arêtes du maillage et en prolongeant la recherche le long des arêtes adjacentes à celles identifiées comme silhouette, il est probable qu'une grande partie des arêtes de silhouette a été détectée. Il s'agit alors de bien choisir les arêtes initiales de la recherche. Markosian *et al.* exploitent dans ce but la forte cohérence entre les silhouettes de deux images successives d'une animation : on choisira donc préférentiellement les arêtes qui appartenaient à la silhouette dans l'image précédente pour débuter la recherche.

Buchanan *et al.* [BS00] proposent une structure permettant d'améliorer et d'accélérer l'implémentation de la détection *force brute* des arêtes de silhouettes. Ils maintiennent, pendant le rendu des faces, un tableau des arêtes du maillage indiquant si elles sont silhouettes ou non : lorsqu'une face est rendue, son orientation (*avant* ou *arrière*) est stockée sur un bit, et les entrées du tableau correspondant aux arêtes de cette face sont mises à jour à l'aide de l'opération XOR. En définitive, seules les arêtes dont les deux faces adjacentes sont d'orientations opposées, ont une entrée de valeur 1 dans le tableau. On peut alors, lors d'une seconde passe, rendre ces arêtes.

Benichou *et al.* [BE99] et Gooch *et al.* [GSG+99] s'appuient sur une représentation alternative pour implémenter une détection exhaustive des arêtes de silhouette. Ils utilisent la fonction de Gauss (qui à tout point d'une surface associe sa normale) pour accélérer la détection de silhouette sur des maillages polygonaux sous projection orthogonale : sur la sphère des directions, les normales aux faces sont des points et les arêtes du maillage sont des arcs (dont les deux points extrêmes sont les normales aux deux faces adjacentes). La sphère de Gauss est illustrée par la figure 2.15. Sous projection orthogonale le point de vue est défini par un plan passant par l'origine de cette sphère. Tout arc intersectant ce plan correspond alors à une arête de silhouette. La détermination des arêtes de silhouette se résume donc à un problème d'intersection entre des arcs sphériques et un plan, problème pour lequel les deux papiers proposent des solutions rapides (Benichou *et al.*, par exemple, projettent la sphère de Gauss sur un cube unitaire de manière à ramener ce problème à des calculs d'intersections

entre segments dans le plan). L'inconvénient de cette approche est qu'elle est difficilement extensible aux projections perspectives.

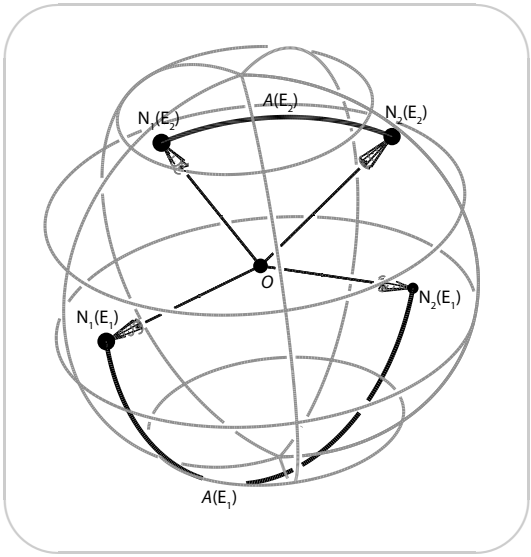


FIG. 2.15 – Sphère de Gauss

La sphère de Gauss est la sphère des directions sur laquelle les normales sont représentées par des points et les arêtes par des arcs. Sur la figure, on a par exemple dessiné l'arête E_1, qui est représentée par un arc $A(E_1)$ reliant les points $N_1(E_1)$ et $N_2(E_1)$ qui correspondent aux normales respectives des deux faces adjacentes à l'arête E_1. © Benichou et al. [BE99].

Dans le même esprit, Barequet *et al.* [BDG+99] proposent une méthode de mise à jour des arêtes de silhouette, travaillant dans l'espace dual et permettant d'accélérer la détermination des silhouettes dans le cadre d'une animation. Dans l'espace dual, les plans supports des faces du maillage sont représentés par des points et le point de vue par un plan. Si l'on considère, dans cet espace, les duaux de deux faces adjacentes du maillage, on peut représenter le segment les joignant par l'espace des plans parcourus par ces faces alors qu'on les fait tourner autour de leur arête commune pour aller d'un plan support à l'autre. Si une arête appartient à la silhouette, ses deux faces adjacentes ont des orientations différentes par rapport au point de vue. Ainsi, si l'on fait tourner le plan support d'une des faces jusqu'au plan support de l'autre face, on croise nécessairement le point de vue. Par définition, un point P appartient à un plan π dans l'espace si et seulement si le point $dual(\pi)$ appartient au plan $dual(P)$. On en déduit que sous le point de vue v, une arête e est une silhouette si et seulement si le plan $dual(v)$ intersecte le segment $dual(e)$. Le problème de la mise à jour des silhouettes d'une image à l'autre se réduit donc à la détermination de l'ensemble des points de l'espace dual intersectés par le plan correspondant au point de vue lorsque celui-ci se déplace de sa position initiale vers sa position courante (lorsqu'un point est rencontré par ce plan, cela signifie qu'une face change d'orientation et que l'on a apparition ou disparition de silhouette). Barequet *et al.* implémentent cette approche à l'aide d'un *octree* qui permet de déterminer rapidement les arêtes de l'espace dual qui intersectent les plans du *wedge* défini (dans l'espace dual) par les deux positions prises par le point de vue à l'image précédente et à l'image courante.

Sander *et al.* [SGG+00] introduisent la notion de *cônes ancrés* (*anchored cones*) qui, couplés avec un arbre de recherche, permettent de calculer la silhouette en temps réel. L'idée est de représenter, pour un ensemble d'arêtes du maillage, le sous-espace des points de vue pour lesquels toutes les faces adjacentes à ces arêtes ont la même orientation (*avant* ou *arrière*) par un cône. Il suffit alors de tester si le point de vue courant est inclus dans un cône (test pouvant être implémenté de manière relativement rapide) pour savoir si tout un groupe de faces sont de même orientation, auquel cas l'ensemble des arêtes associées n'appartiennent pas à la silhouette. La figure 2.16 illustre un tel cône pour un ensemble de faces. Sander *et al.* rangent l'ensemble des arêtes du maillage dans un arbre de manière à accélérer la recherche de celles dont les faces adjacentes ont des orientations différentes : à chaque nœud est associé un ensemble d'arêtes du maillage ainsi que le cône ancré correspondant à l'ensemble de leurs faces adjacentes. Le stockage des arêtes dans les nœuds est fait de manière à ce que chaque nœud contienne des arêtes dont les faces sont d'orientations proches, et l'arbre est construit de telle sorte que l'ensemble des cônes ancrés des nœuds enfants soient inclus dans le cône ancré du nœud parent.

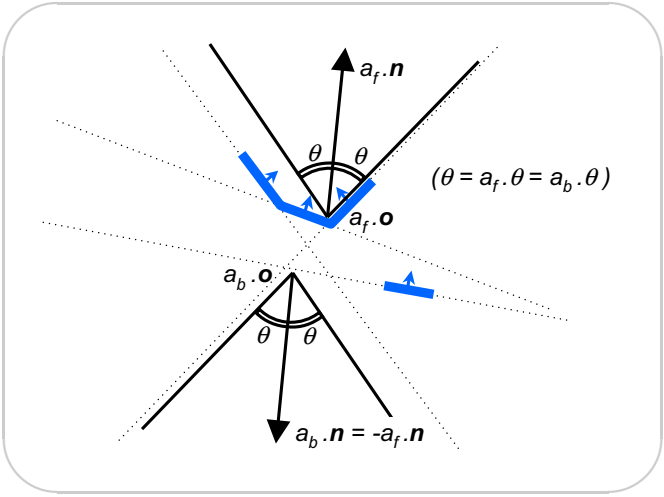


FIG. 2.16 – Cônes ancrés

Illustration des cônes ancrés en 2D : les quatre faces dessinées en bleu sont toutes orientées de la même manière par rapport au point de vue, dès que celui-ci est inclus dans le cône d'angle θ. © *Sander et al. [SGG+00].*

Toutes les techniques que nous avons vues dans cette section permettent la détermination d'une approximation (imprécise) de la silhouette sous la forme d'un sous-ensemble des arêtes du maillage. Ce résultat présente en particulier des problèmes topologiques (*cf.* section 2.3.2). En 2000, Hertzmann *et al.* [HZ00] proposent une approche permettant de calculer une meilleure approximation de la silhouette à partir d'un maillage polygonal, de manière à éviter au maximum les problèmes topologiques. L'idée de base, inspirée par la technique du *Marching Cubes* [LC87] est de construire un ensemble de segments, sur le maillage, qui sont une meilleure approximation des lieux de silhouette : l'algorithme consiste à calculer en chaque sommet le produit scalaire de la normale par le vecteur direction de vue, à détecter les faces pour lesquels ces produits scalaires ne sont pas de même signe pour tous les sommets (ce qui signifie que la silhouette passe *par* ces faces). On construit alors le segment qui traverse chaque face et

pour lequel le produit scalaire vaut exactement 0 (en première approximation ceci peut être déterminé en interpolant linéairement les valeurs des produits scalaires obtenus en chaque sommet). La figure 2.17 montre la différence de qualité entre ces silhouettes et leur approximation polygonale. Contrairement aux approches précédentes, ces segments de silhouette ne sont a priori pas des arêtes du maillage et, la plupart du temps, ils traversent les faces. Le pseudo-code pour cet algorithme est donné, dans le cas de faces triangulaires, algorithme 2.4 :

Algorithme 2.4 Construction de silhouette exacte

Fonction *BuildExactSilhouette*
Entrée: V le point de vue, L'ensemble $\mathcal{V}$ des sommets du maillage, L'ensemble $\mathcal{F}$ des faces du maillage
Sortie: L'ensemble $\mathcal{S}$ des segments constituant la silhouette

```
1.  pour v ∈ V
2.          N ←la normale à v
3.          On associe à v la valeur du produit scalaire d = (v − V).N
4.  pour f ∈ F
5.          v1, v2, v3 ←les sommets de f
6.          d1, d2, d3 ←les valeurs du produit scalaire (vi − V).N pour i = 1, 2, 3
7.          k ←0
8.          pour i ∈ 1, 2, 3 et j = (i + 1)(mod3)
9.                  si di × dj < 0
10.                    alors
11.                        (∗ la silhouette passe entre vi et vj ∗)
12.                        sk = dj/(dj−di) vi + di/(di−dj) vj
13.                        k ←k + 1
14.         si k ≠ 0
15.           alors
16.                 (∗ une silhouette passe par la face f ∗)
17.                 on ajoute le segment sk sk+1 à S
```

Comme nous le verrons en section 2.3, lorsque la surface sous-jacente au maillage est lisse, l'approximation polygonale de la silhouette, telle que les précédentes techniques la calculent, peut poser des problèmes dans le cas de rendus stylisés et la solution de Hertzmann *et al.* est une bonne manière d'éviter ces difficultés. Pour cette raison, nous avons choisi de nous appuyer sur cette technique pour déterminer les lignes de silhouette.

> **Remarque :** Il est suffisant d'utiliser une interpolation linéaire (au lieu d'une interpolation d'ordre au moins égal à deux) pour déterminer les points des faces pour lesquels le produit scalaire s'annule, dans la mesure où les faces portant les silhouettes sont observées selon une direction de vue qui leur est quasi-tangente.

Cet algorithme peut par ailleurs être accéléré en travaillant dans l'espace dual avec une approche similaire à celle de Barequet *et al.* [BDG+99] présentée plus haut.

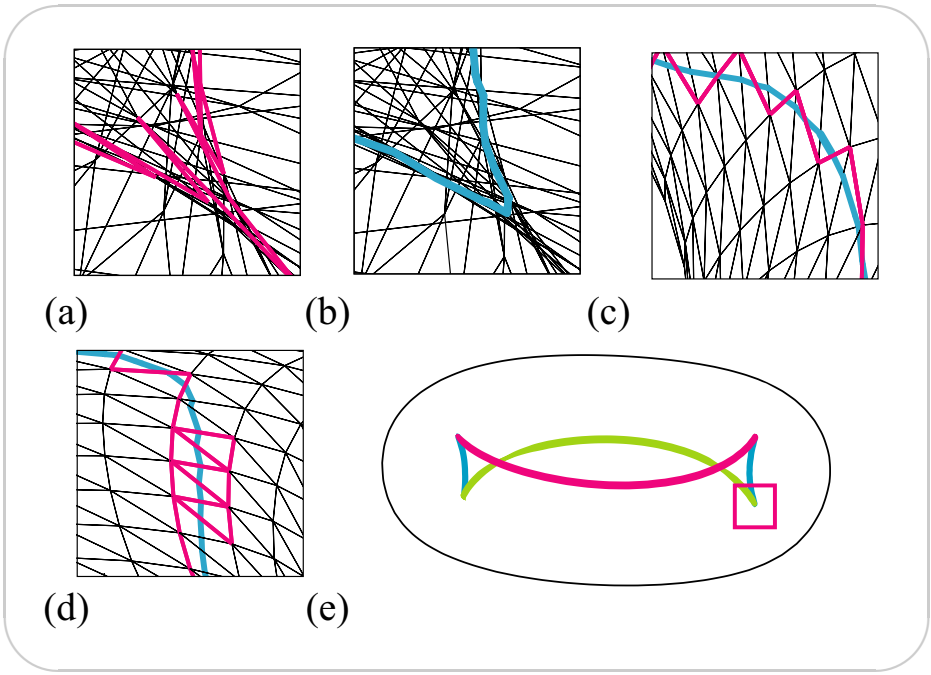


FIG. 2.17 – Silhouettes lisses *vs.* approximation polygonale

(a) : L'approximation polygonale des silhouettes produit des courbes irrégulières. (b) : La méthode de Hertzmann et al. produit des courbes de silhouette lisses en inférant des informations sur la surface lisse à partir de son approximation polygonale. (c) : Les mêmes courbes vues sous un autre point de vue et superposées. (d) : Avec les silhouettes lisses, il n'y a pas de superposition d'arêtes de silhouette pour une surface convexe lorsque celle-ci est quasi-parallèle à la direction de vue. (e) : Lignes de silhouette du "tore souriant". Le cadre rouge montre la position des courbes pour les illustrations de (a) à (c). © *Hertzmann et al. [HZ00].*

Contours suggestifs

Le calcul des contours suggestifs [DFRS03] se fait selon un algorithme similaire à celui utilisé par Hertzmann et Zorin [HZ00] (*cf.* algorithme 2.4). Au lieu d'un produit scalaire, c'est ici la fonction donnant la courbure radiale dont on va chercher les zéros sur la surface. Ainsi, après avoir calculé κ_r en chaque sommet du maillage, les faces pour lesquelles on observe des signes opposés sur les différents sommets sont identifiées, et les segments réalisant le zéro de κ_r sont construits par interpolation linéaire. La courbure radiale peut être calculée en chaque sommet à partir des valeurs des deux courbures principales à l'aide de la formule suivante [dC76] :

$$\kappa_r(P) = \kappa_1(P)cos^2(\phi) + \kappa_2(P)sin^2(\phi) \tag{2.2}$$

κ_1 et κ_2 étant les courbures principales, et ϕ l'angle entre $\mathbf{w}(P)$ et la direction de la courbure principale correspondant à κ_1. Pour le calcul des courbures principale sur un maillage polygonal, le lecteur pourra par exemple se référer au travail de Alliez *et al.* [ACSD+03]. Cette construction produit un ensemble de courbes fermées sur la surface. Il suffit alors de calculer la dérivée directionnelle de κ_r de manière à ne garder que les morceaux de courbes pour lesquels elle est positive. Ces opérations sont à répéter chaque fois que le point de vue change. La construction des contours suggestifs exigeant un calcul différentiel (courbures), elle

est souvent bruitée sur les maillages polygonaux et nécessite l'introduction de seuils filtrants. En particulier, les lignes obtenus à partir de valeurs de courbures trop faibles seront éliminés.

Crêtes et vallées

Les crêtes et les vallées correspondent aux extrema des courbures principales. Théoriquement, il suffit donc de calculer les dérivées des courbures principales en chaque sommet et de construire les polylignes réalisant les zéros de ces dérivées en utilisant à nouveau un algorithme similaire à celui de Hertzmann et Zorin (algorithme 2.4). En pratique, de tels calculs différentiels d'ordre 2 donnent des résultats très bruités sur des maillages polygonaux et rendent la construction des lignes de crête et de vallée difficile. De nombreux travaux [RKS00, NH01] s'affranchissent de ce problème en calculant, de manière grossière, sur le maillage, les *zones* de crêtes et de vallées puis en en extrayant le squelette à l'aide d'opérations morphologiques. Plus récemment, Ohtake *et al.* [OBS04] ont obtenu de bons résultats en calculant une approximation du maillage polygonal à l'aide de fonctions à base radiale. De cette manière, les dérivées de courbure peuvent être exprimées et calculées en chaque sommet, puis les lignes de crêtes et de vallées peuvent en être déduites.

Résumé: Dans cette section, nous avons vu comment calculer efficacement les différents types de lignes caractéristiques, à savoir les silhouettes, les contours suggestifs, les arêtes vives et les bords. En particulier, la littérature traitant de la détection des silhouettes (et de l'accélération d'une telle détection) est vaste. Nous choisissons pour notre système, de calculer les lignes de silhouette en suivant l'approche de Hertzmann *et al.* [HZ00], qui fournit des lignes adaptées à la stylisation.

2.2.3 Informations : visibilité

À ce stade de l'état de l'art, nous avons vu diverses techniques permettant d'extraire d'une surface l'ensemble des segments 3D qui constituent les lignes caractéristiques de cette surface, observée sous un point de vue donné. Un certain nombre d'étapes restent encore à franchir pour passer de cet ensemble de segments à un rendu stylisé qui donne l'illusion d'un dessin fait à la main. En pratique, une certain nombre d'informations sont calculées et associées aux segments caractéristiques ainsi détectées, au tout premier rang desquelles la plus communément utilisée est l'information de visibilité. En effet, parmi ces segments, certains sont entièrement ou partiellement invisibles sous le point de vue considéré. Excepté dans des cas particuliers tels que l'illustration technique, ces parties cachées ne sont pas dessinées par un artiste. Il est donc essentiel d'être capable de calculer la visibilité de ces lignes caractéristiques, c'est-à-dire pour chaque point des lignes, de déterminer s'il est visible ou caché.

On distingue deux types d'approches :

Les approches exactes ces approches s'appuient sur des propriétés géométriques mettant en relation les surfaces et le point de vue pour identifier les points auxquels on observe des changements de visibilité. Une fois les lignes caractéristiques segmentées par la création de sommets en ces points, il ne reste plus qu'à déterminer la visibilité de

chaque segment de courbe. Ce dernier point peut, par exemple, être implémenté à l'aide d'une technique de lancer de rayons.

Les approches discrètes Ces approches procèdent à une première passe de rendu de la scène et utilisent l'image du *Z-Buffer* ainsi générée pour déterminer la visibilité en chaque point des lignes caractéristiques.

Les approches exactes

L'étude de la visibilité des lignes caractéristiques selon l'approche exacte est liée aux travaux sur les graphes d'aspect [PD90b] dans lesquels on peut trouver une caractérisation des éléments géométriques jouant un rôle dans les changements de visibilité observés sur les lignes. En pratique, étant donnée une ligne de nature quelconque, on peut observer un changement de visibilité aux points suivants :

Intersections en T. Ces points correspondent aux intersections (dans la plan image) des projections de la ligne à tracer et d'une silhouette : potentiellement, le contour passe derrière un objet ou ressort de derrière un objet. (*cf.* figure 2.18 (a)).

Jonctions en Y. Ces points correspondent aux points 3D où une silhouette débute (ou s'achève) sur notre ligne. (*cf.* figure 2.18 (a)).

Points de rebroussement (*Cusps* en Anglais). Ils correspondent aux points de la silhouette où sa tangente est colinéaire à la direction de vue. (*cf.* figure 2.18 (b)).

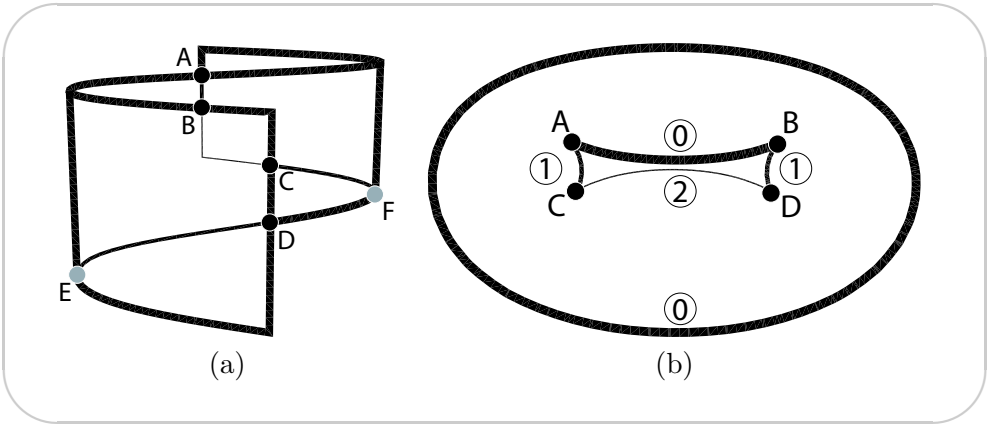


FIG. 2.18 – Points de changement de visibilité : jonctions en T et en Y et *cusps*

*(a) Jonctions en T et en Y : Les points A, B, C et D sont des jonctions en T. Ils correspondent à des intersections entre les projections d'une arête de silhouette et d'une autre arête. Les points E et F (en gris) sont des jonctions en Y. Ils correspondent à des jonctions (en 3D) entre des arêtes caractéristiques et des arêtes de silhouette. (b) : Points de rebroussement (*cusps *: les points A, B, C et D sont des points de rebroussement. Ils marquent des changements d'invisibilité quantitative (dont les valeurs sont indiquées à côté des arêtes). Sur ces deux illustrations, les arêtes visibles sont dessinées avec le poids le plus important, les arêtes cachées par une seule surface sont dessinées avec un poids moyen et les arêtes cachées par deux surfaces sont dessinées avec un poids faible.*

C'est Appel [App67a] qui le premier utilise ces points pour déterminer la visibilité des lignes caractéristiques. Il introduit, pour ce faire, *l'invisibilité quantitative*, qui donne, en

chaque point, le nombre de surfaces qui le cachent. S'il observe des singularités aux points de rebroussement, Appel ne les caractérise pas encore et se limite aux intersections en T. Plutôt que d'évaluer l'invisibilité quantitative de chaque segment par un coûteux lancer de rayons, Appel montre qu'à l'aide de propriétés géométriques simples, il est possible de connaître l'évolution de l'invisibilité quantitative le long d'une ligne. En effet, grâce à l'orientation des faces il est possible de déterminer si la projection d'un point se trouve à l'intérieur ou à l'extérieur de la projection d'une face ; Appel utilise cette propriété pour savoir si une ligne passe derrière une face ou en ressort. L'algorithme consiste alors à calculer l'ensemble des intersections des projections des lignes, à déterminer l'invisibilité quantitative d'un certain nombre de points initiaux (à l'aide d'un lancer de rayons par exemple), puis à mettre à jour cette invisibilité quantitative le long des lignes en la décrémentant ou en l'incrémentant à chaque intersection avec une silhouette, selon que la ligne passe derrière une surface ou en ressort.

Markosian [MKT+97] reprend la technique d'Appel et propose d'exploiter un certain nombre de propriétés de manière à l'accélérer. En particulier, les points de rebroussement sont caractérisés de manière à limiter les tests de visibilité. Ainsi, pour une approximation polygonale de la silhouette, un sommet est un point de rebroussement si :

1. il est adjacent à exactement deux arêtes de silhouette, l'une orientée *avant* et l'autre orientée *arrière* : une arête de silhouette est dite orientée *avant* lorsque sa face adjacente la plus proche du point de vue est elle-même orientée vers le point de vue et *arrière* dans le cas contraire.
2. il est adjacent à plus de deux arêtes de silhouette.

Si l'on suit l'approche exacte, le calcul de visibilité peut poser des problèmes lorsque l'on travaille avec les silhouettes exactes introduites par Hertzmann *et al.* [HZ00] (*cf.* section 2.2.2). En effet, dans ce cas, les silhouettes obtenues approchent celles de la surface lisse approximée par le maillage, et, à moins de raffiner le maillage dans les zones de silhouette lors du calcul de visibilité, il est donc possible d'observer des contradictions entre la visibilité attendue et celle calculée à partir du maillage. En pratique, pour chaque portion de contour de visibilité constante on aura recours à plusieurs tests de visibilité de manière à choisir celle donnée par la majorité. Hertzmann *et al.* proposent également une technique pour détecter les points de rebroussement, l'utilisation de la propriété de colinéarité de la tangente et de la direction de vue n'étant pas numériquement fiable. Ils montrent que ces points correspondent aux intersections sur la surface de la silhouette avec la courbe d'équation :

$$\kappa_1((P-V)\cdot\mathbf{w_1})^2 + \kappa_2((P-V)\cdot\mathbf{w_2})^2 \tag{2.3}$$

κ_1 et κ_2 étant les valeurs des courbures principales dans leurs directions respectives $\mathbf{w_1}$ et $\mathbf{w_2}$. La figure 2.19 montre ces deux courbes.

Nous avons choisi de déterminer l'invisibilité quantitative des arêtes caractéristiques à l'aide d'une approche exacte. En raison des imprécisions dues au problème d'incompatibilité entre la silhouette (que nous calculons d'après la méthode de Hertzmann et Zorin) et le modèle polygonal, nous effectuons plusieurs lancers de rayons par segment de visibilité constante et avons recours à un vote pour déterminer la valeur de l'ensemble du segment.

Pour déterminer les points de rebroussements, nous avons choisi de détecter les points pour lesquels la courbe de silhouette est colinéaire à la direction de vue : pour chaque arête de la silhouette, on calcule son produit vectoriel avec la normale à la surface en son milieu ; lorsque le produit scalaire entre ce vecteur et la direction de vue est nul, la silhouette est colinéaire

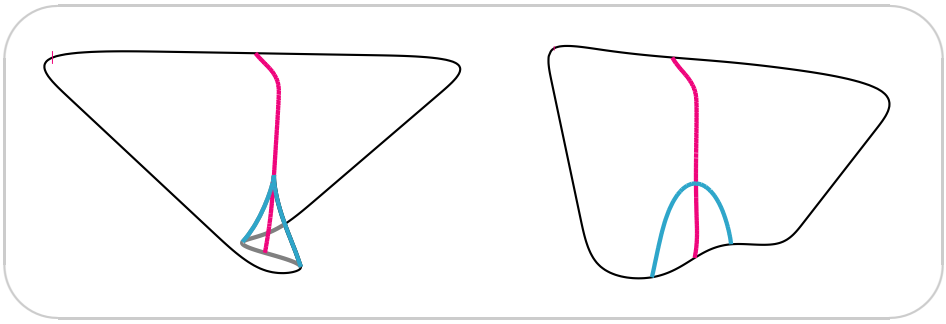


FIG. 2.19 – Détection des points de rebroussement

Gauche : les points de rebroussement sont les intersections des zéros de deux fonctions définies sur la surface, le produit scalaire de la normale avec la direction de vue (silhouette) et la fonction d'équation 2.3. La silhouette est déssinée en bleu, l'autre fonction en rouge. Droite : les mêmes courbes, vues sous un point de vue différent. © *Hertzmann et al. [HZ00].*

à la direction de vue. Ce produit scalaire change également de signe de part et d'autre d'un point de rebroussement. Nous marquons donc chaque arête de silhouette en fonction du signe de ce produit scalaire et créons un point de rebroussement lorsqu'il y a changement de signe. De plus, on utilise une fonction hystérésis de manière à amoindrir les conséquences du bruit lié à ce calcul. Cette approche s'est révélée simple à implémenter et efficace.

Les approches discrètes

La principale difficulté avec les approches discrètes et l'utilisation du *Z-Buffer* pour déterminer la visibilité de lignes, est l'imprécision du *Z-Buffer*.

Northrup *et al.* [NM00] et Markosian *et al.* [MMK+00] utilisent une approche discrète et s'appuient sur le *Z-Buffer* pour déterminer la visibilité des lignes. Dans une première passe, une image référence d'identité (*ID reference image*) est construite : chaque face et chaque arête sont rendues avec une couleur unique. Les arêtes n'ayant pas contribué à cette image sont totalement cachées. Pour déterminer les segments visibles des arêtes restantes, celles-ci sont rasterisées et, pour chaque pixel, l'image de référence permet de savoir s'il est visible ou non.

Isenberg *et al.* [IHS02] étendent cette approche et proposent une méthode similaire pour laquelle le recours à une image référence d'identité n'est plus nécessaire. En effet, Isenberg *et al.* s'appuient uniquement sur le *Z-Buffer* pour déterminer si un pixel donné est visible. Afin de contourner les problèmes de précision, pour chaque pixel on teste les valeurs de z des 8 pixels voisins, si l'un d'entre eux est plus loin que le pixel concerné, ce dernier est déclaré visible.

Résumé: Les méthodes de calcul de visibilité pour les lignes caractéristiques peuvent être exactes (et s'appuyer sur des outils géométriques) ou discrètes (et utiliser le *Z-Buffer*).

2.3 Construction de traits et affectation d'attributs

Cette section décrit les opérations qui se déroulent au sein du deuxième étage du pipeline de la figure 2.12 et qui sont responsables de la stylisation. Dans la section suivante, nous verrons quels types de contrôles sont offerts par les techniques actuelles sur ces opérations pour mener à bien cette stylisation.

2.3.1 Chaînage des arêtes caractéristiques

Quelle que soit la technique employée pour détecter les lignes caractéristiques, celle-ci produit un ensemble de segments 3D dont la taille est du même ordre de grandeur que celle de l'ensemble des arêtes du maillage. La figure 2.20 montre les segments de silhouette obtenus sur un tore par la méthode de Hertzmann et Zorin [HZ00]. Une fois la visibilité prise en compte, on est donc capable de produire un dessin en affichant la projection des arêtes à l'aide de traits monochromes d'épaisseur constante.

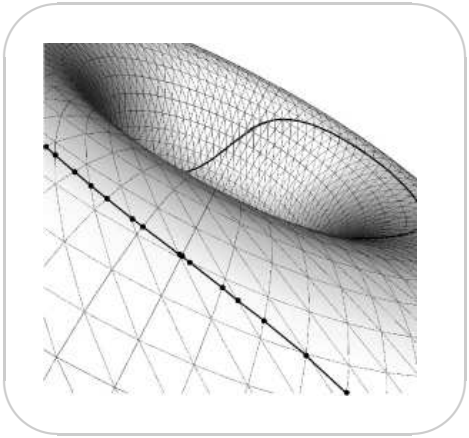


FIG. 2.20 – Segments de silhouette

Sur cette figure, on peut voir les segments de silhouette obtenus par la méthode de Hertzmann et Zorin [HZ00] : sur chaque triangle du maillage par lequel passe la silhouette, un segment approxime cette silhouette. Ces segments doivent être chaînés de manière à construire des traits plus longs, plus conforme à des traits dessinés par un artiste.

Toutefois, un tel dessin ne sera pas satisfaisant. En effet, pour qu'un rendu NPR puisse créer, de manière plus convaincante, l'illusion d'un dessin fait à la main, la couleur, la texture et la géométrie des traits doivent pouvoir imiter les interactions de l'outil (*e.g.* pinceau) et du medium (*e.g.* huile) sur le support (*e.g.* toile) ainsi que le tracé lisse ou irrégulier de l'artiste. On s'intéresse dans cette section à ce dernier point qui concerne la définition de la "topologie" des traits. On commence par définir un *trait* comme la marque laissée par l'outil au cours d'un geste unique de l'artiste (*i.e.* sans que l'outil ait quitté le support). Il est alors indispensable de construire de longues chaînes à partir des arêtes caractéristiques de manière à imiter les longs traits généralement tracés par les artistes. Nous verrons dans les prochaines sections qu'une fois la topologie des traits définie, il existe des techniques permettant de leur affecter des attributs et d'en faire un rendu qui parvient à simuler un véritable tracé de trait.

La construction de ces chaînes est, dans la plupart des travaux antérieurs, réduite à sa plus simple version, qui consiste, en partant d'une arête caractéristique, à suivre la ligne à laquelle cette arête appartient le long de la surface jusqu'à ce qu'on revienne au point de départ ou jusqu'au prochain changement de visibilité.

Sousa *et al.* [SP03b] adoptent toutefois un schéma plus élaboré que celui que nous venons d'énoncer, dans lequel les adjacences du graphe d'aspect de la scène sont considérées de manière à pouvoir chaîner sans respecter la topologie des surfaces 3D.

2.3.2 Artefacts de paramétrisation

D'une manière générale, les traits sont construits en chaînant des segments de l'espace 3D puis en projetant ces chaînes dans le plan image. Chaque chaîne 3D est, excepté en ses extrémités, localement topologiquement équivalente à une droite. Il est essentiel que cette propriété soit conservée lors de la projection pour disposer d'une paramétrisation correcte des chaînes 2D, indispensable à leur stylisation. Or, il existe de nombreux cas problématiques, en particulier dans le cas des approximations polygonales des silhouettes.

Northrup *et al.* [NM00] proposent une catégorisation de ces problèmes et un ensemble de solutions qui y sont dédiées. Un premier artefact qu'ils observent est la superposition des projections de plusieurs segments de silhouette (dans le cas d'un morceau de surface plane tangent à la direction de vue) (*cf.* figure 2.21 (a)). L'autre artefact fréquemment rencontré lorsque les silhouettes sont approximées polygonalement est l'effet de zigzags observable sur les chaînes 3D une fois qu'elles sont formées. Le résultat de ces zigzags est une paramétrisation incohérente de la projection de la chaîne (*cf.* figure 2.21 (b)).

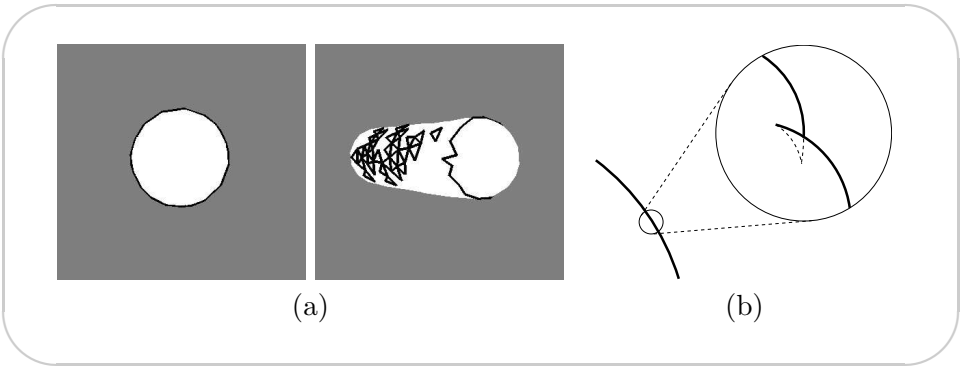


Fig. 2.21 – Rendu de traits : artefacts des chaînes polygonales de silhouette

(a) : Une silhouette simple en apparence (à gauche) peut être le résultat de la superposition d'un ensemble complexe d'arêtes (à droite). (b) : Avec les approximations polygonales de silhouette, on observe souvent des zigzags le long des chaînes. © Northrup et al. [NM00]

Northrup *et al.* repèrent, avant de chaîner, les segments qui seront à l'origine de ces artefacts. Ainsi, les segments qui, dans l'espace image, se recouvrent et sont presque parallèles sont détectés : dans le cas où deux segments se recouvrent partiellement, leur géométrie est modifiée de manière à ce qu'ils soient connectés (*cf.* figure 2.22 (a)) ; lorsque l'un des deux segments est complètement recouvert par un segment plus grand, il est supprimé (*cf.* figure 2.22 (b)).

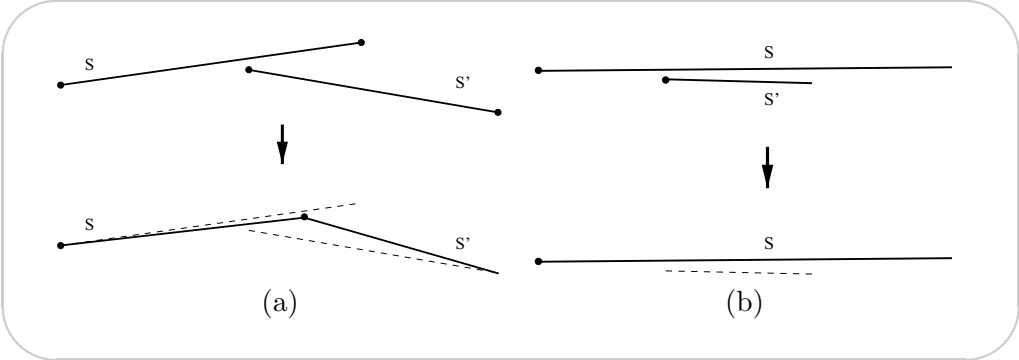


FIG. 2.22 – Rendu de traits : correction des recouvrements de segments

(a) : Les segments S et S' sont corrigés en redéfinissant leurs sommets superposés par le point équidistant. (b) : S' est complètement recouvert par un segment plus grand, il est donc éliminé. © *Northrup et al. [NM00]*

Isenberg *et al.* [IHS02] identifient en outre quelques artefacts supplémentaires, dûs à des imprécisions du maillage, qui résultent en des alternances de faces *avant* et *arrière* :

Triangles avec deux arêtes caractéristiques Cette situation est illustrée par la figure 2.23 gauche. Pour la corriger, Isenberg *et al.* suggèrent de remplacer la paire d'arêtes concernée par l'arête du triangle restante, comme illustré par la figure 2.23 droite.

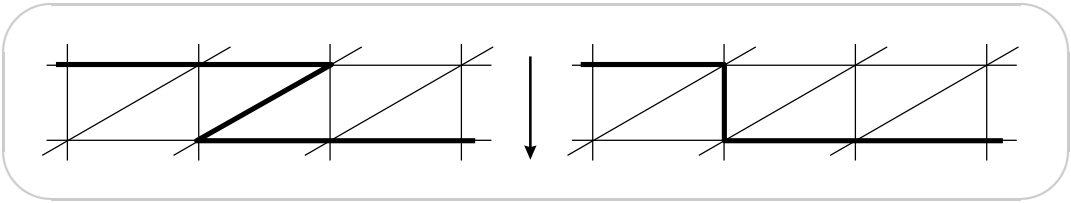


FIG. 2.23 – Rendu de traits : double arête de silhouette pour un triangle

Cette figure illustre une manière de corriger les silhouettes dont deux arêtes appartiennent au même triangle. Après traitement, la silhouette ne passe plus que par la troisième arête de ce triangle. © *Isenberg et al. [IHS02].*

Groupes d'arêtes de silhouette Lorsqu'un morceau de surface à faible courbure est tangent à la direction de vue, les imprécisions dues à l'approximation polygonale de cette surface deviennent plus évidents. On observe notamment souvent une alternance entre les faces détectées *avant* et celles détectées *arrière*. Lors d'une détection de silhouette polygonale, la conséquence est que l'on obtient un groupe d'arêtes de silhouette au lieu d'une seule, comme illustré par la figure 2.24 (a). La solution de Isenberg *et al.* à ce problème consiste à choisir le plus court chemin passant par la zone à problème et à supprimer les autres arêtes. La figure 2.24 (b) illustre un résultat obtenu avec cette technique.

L'utilisation d'approximations plus exactes pour les lignes de silhouette, à l'aide d'une méthode telle que celle de Hertzmann et Zorin [HZ00], permet d'éviter ces artefacts.

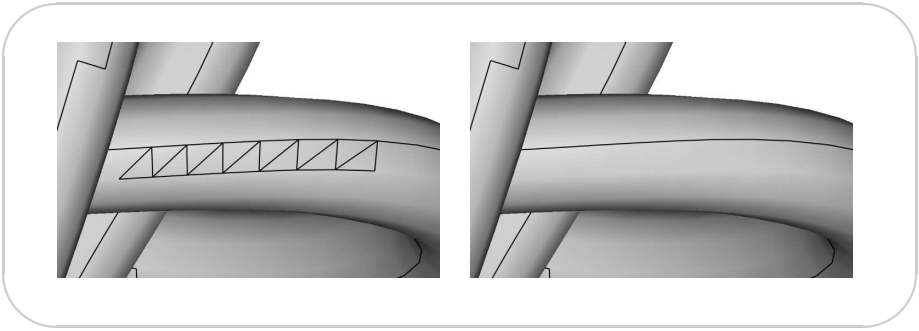


FIG. 2.24 – Rendu de traits : groupes d'arêtes de silhouette

(a) : L'approximation polygonale d'une surface peut résulter en un groupe de triangles détectés alternativement avant *et* arrière *et donc en un groupe d'arêtes de silhouette. (b) : La solution consiste à choisir le plus court chemin de silhouette passant par la zone problématique, les autres arêtes étant alors supprimées.* © *Isenberg et al. [IHS02].*

Résumé: Afin de produire des traits convaincants (*i.e.* imitant de manière crédible des traits réels), il faut construire des chaînes à partir des segments caractéristiques détectés sur les surfaces. Ce chaînage définit la paramétrisation des traits et peut être problématique, en particulier dans le cas d'une approximation polygonale de la silhouette dont le rendu engendre de nombreux artefacts qu'il faut gérer. Il est préférable de travailler avec une approximation plus exacte de silhouette.

2.3.3 Rendu des marques

Finalement, une fois les traits construits et leurs attributs spécifiés, il ne reste qu'à en faire le rendu. La difficulté consiste alors à imiter de la manière la plus réaliste possible l'apparence d'un trait réel. Celle-ci dépend d'un certain nombre de paramètres, tels que l'outil, le medium et le support utilisés.

Il existe deux méthodes pour produire une image à partir des traits munis de leurs attributs : la simulation ou l'utilisation de textures.

Les rendus de traits basés sur la simulation s'appuient sur des modèles physiques (*e.g.* de diffusion) pour calculer la couleur déposée par l'outil en chaque pixel, à partir des attributs des traits (*e.g.* outil, medium).

Curtis *et al.* [CAS+97] simulent par exemple des tracés à l'aquarelle, en s'appuyant sur des techniques de simulation d'écoulement des fluides pour calculer les mouvements d'eau et de pigments lorsque le pinceau est passé sur la feuille. La feuille, modélisée comme un champ de hauteurs, peut être vue comme une grille pour laquelle chaque case stocke finalement la densité d'eau et de pigment. Le rendu se fait alors en utilisant le modèle de Kubelka-Munk [HM92] pour simuler la composition optique de tracés multi-couches. Les résultats obtenus par cette approche sont particulièrement réalistes, comme on peut le voir sur la figure 2.25. En revanche, les temps de calcul ne permettent pas une utilisation interactive.

D'autres travaux, tels que ceux de Strassmann [Str86], se sont également intéressés à la modélisation physique du pinceau pour simuler une répartition spatiale réaliste des pigments

FIG. 2.25 – Système d'attributs et de marques : simulation

En haut : effets réels de peinture aquarelle. En bas : les mêmes effets obtenus par simulation. © Curtis et al. [CAS+97].

sur la feuille. Le lecteur trouvera dans la littérature [Lee99, YLLC03, WHS03, BWL04] plusieurs articles, traitant notamment de simulation de peintures orientales à l'encre, décrivant des méthodes similaires.

De même, Sousa *et al.* [SB99b] ont proposé un système pour simuler le tracé du crayon sur du papier dans le cadre d'un système complet de rendu de scènes 3D.

Dans notre cas, le but est d'offrir le maximum de contrôle et de précision sur la forme et les attributs des traits, et peut potentiellement entrer en conflit avec une approche par simulation. L'alternative à la simulation, adoptée dans la grande majorité des cas et en particulier dans le rendu interactif, est le plaquage de texture. Ce dernier copie la réalité "microscopique" (sous la forme d'une image de trait) au lieu de la simuler tout en laissant un contrôle potentiel sur les attributs "macroscopiques", tels que la couleur et l'épaisseur. De plus, cette technique permet d'exploiter les ressources des cartes graphiques pour accélérer l'affichage. Dans ce contexte, un trait est rendu sous forme d'une bande de triangles (*triangle strip*) texturée. Plusieurs articles proposent des méthodes destinées à construire de telles structures à partir de la géométrie du trait et de ses attributs d'épaisseur. Ces articles décrivent en particulier la gestion des cas problématiques pour lesquels le trait présente une courbure importante.

Hsu *et al.* [HL94] proposent, dans un tel cas, de subdiviser la géométrie du trait d'autant plus que la courbure est importante. Virtuellement, on ne peut donc pas avoir de repliement d'un triangle sur son voisin.

Northrup *et al.* [NM00] quant à eux proposent simplement de modifier l'épaisseur du trait en fonction de l'angle formé par deux segments consécutifs du trait : plus cet angle est faible, plus l'épaisseur sera multipliée par un facteur important (*cf.* figure 2.26). Ceci permet d'éviter les rétrécissements aux tournants d'un trait dont l'échantillonnage est insuffisant.

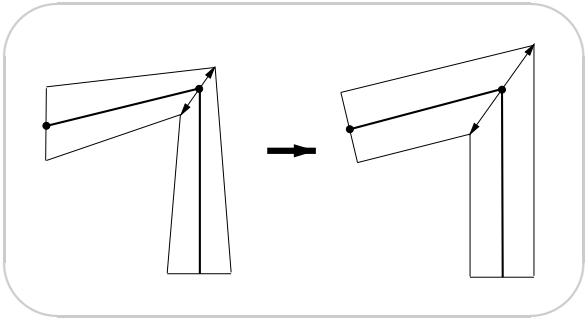


FIG. 2.26 – Rendu de traits : *skeletal strokes*
L'épaisseur est augmentée aux points marquant un angle important entre deux segments consécutifs du trait de manière à conserver une épaisseur constante. © *Northrup et al. [NM00]*

Résumé: Le rendu des traits stylisés peut se faire soit par simulation physique, soit par plaquage d'une texture de trait réel sur une bande de triangles reproduisant les variations géométriques des traits construits.

2.4 Contrôle du style en rendu non-photoréaliste

Avant de parler des travaux de recherches s'intéressant au contrôle du style dans le rendu non-photoréaliste, nous listons brièvement les solutions commerciales dédiées au rendu stylisé dont nous avons pu avoir connaissance.

2.4.1 Les solutions commerciales

Le récent engouement pour le NPR dans la communauté scientifique a très rapidement été suivi par l'industrie. Ainsi, aujourd'hui, la plupart des grandes solutions de graphisme 3D intègrent un composant, plus ou moins développé, de rendu NPR.

Le style de rendu NPR le plus populaire est de loin le *Cartoon Shading* [Dec96b, LMHB00], qui consiste en la combinaison du tracé de lignes (de manière relativement uniforme) et de la coloration des régions à l'aide d'à-plats de couleurs. Plusieurs clips vidéos (*e.g.Respire* [AC03b], *Yalil* [AC03c] et *Deux pieds* [AC03a]) et au moins un long métrage (*Apple Seed* [SA04]) ont déjà été réalisés dans ce style (*cf.* figure 2.27) à l'aide d'ordinateurs.

Dans leur grande majorité, les composants NPR commerciaux se sont focalisés sur ce rendu et de nombreux plugins (*e.g. Illustrate !*, *CartoonReyes*, *Penguin*) y sont dédiés. Le contrôle offert par ces logiciels sur les attributs des lignes est en général relativement pauvre et se limite à la spécification de poids et de couleur. Il existe également un *shader Renderman* [Ups89] permettant de faire ce type de rendu. Cependant, *Renderman* n'étant pas prévu pour la détection et l'analyse d'éléments 1D, les possibilités sont très limitées.

Le moteur de rendu *MentalRay*, notamment intégré aux logiciels *Maya* et *Softimage|XSI*, intègre un *Toon Shader* [Ari02] plus riche, qui offre un contrôle fin sur de nombreux attributs

(a) (b)

FIG. 2.27 – *Cartoon Shading*

Deux images rendues dans le style "dessin animé". (a) : Une image du clip "Deux pieds" (Thomas Fersen), réalisée à l'aide du logiciel CartoonReyes. © *Jérome Combe et Stéphane Hamache 2003 - Deux pieds (Thomas Fersen) [AC03a]. (b) : Une image du long-métrage* Appleseed *[SA04].*

des traits. Cependant, ce contrôle reste de type paramétrique et ne permet, par exemple, pas de s'appuyer sur des informations de la scène.

Récemment, Disney a communiqué des informations au sujet de leur système interne de rendu de lignes, *Inka* [Tee03]. Il permet un rendu flexible à l'encre de modèles 3D et s'appuie sur une philosophie de *shaders*. De même que pour *MentalRay*, le contrôle sur les attributs est paramétrique et ne peut s'appuyer sur des informations de la scène.

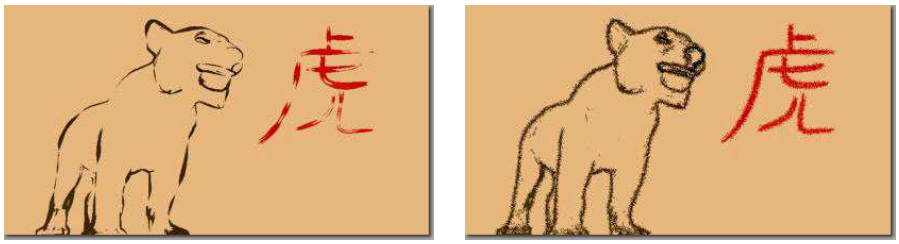


FIG. 2.28 – *Toon Shader* de *MentalRay*

Deux rendus différents du même modèle faits dans le logiciel Softimage|XSI.

D'autres logiciels proposent également des rendus NPR plus variés, mais pour lesquels le contrôle offert dans chaque cas se borne au choix de l'un ou l'autre des styles prédéfinis (ou à une combinaison de tels styles). *NPR1Reyes*, *Penguin* ou *Eva* par exemple offrent quelques styles de traits intéressants (*cf.* figure 2.29).

Notre but, à terme, est de permettre le développement d'une application dédiée à la modélisation de style et offrant la même souplesse que celle offerte par *Renderman* en rendu

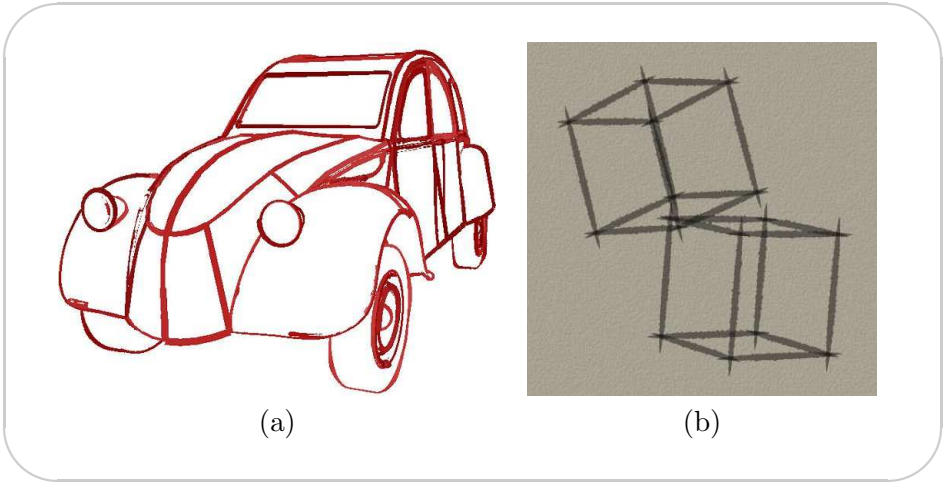


FIG. 2.29 – Des *shaders* de traits plus variés

Certains logiciels proposent quelques shaders *de style prédéfinis plus variés que ceux des* Toon Shaders. *(a) : Une image produite à l'aide de* Penguin © *McNeel - www.penguin3d.com. (b) : Une image produite à l'aide de* Eva © *Algorithmic - www.algorithmic.com/eva_e.html .*

photoréaliste. En particulier, on veut pouvoir s'appuyer sur des informations issues de la scène pour exprimer un style.

2.4.2 Les travaux de recherche

Les techniques de recherche que nous avons vues jusqu'à présent constituent les briques de base pour un rendu stylisé. Dans le pipeline général que nous avons identifié pour ce type de rendu, la mise en style intervient plus particulièrement à deux niveaux : la construction des traits à partir des arêtes caractéristiques et l'affectation des attributs à ces traits. C'est pendant la construction des traits que l'on décide de leur topologie, que l'on décide également d'en omettre certains ; et c'est avec l'affectation des attributs que ces traits sont pourvus d'une apparence visuelle particulière.

Les travaux que nous présentons dans cette section s'intéressent au style dans le rendu NPR. La plupart de ces travaux peuvent être regroupés en deux catégories : celle des approches automatiques et celle des approches interactives. Après avoir décrit chacune de ces catégories, nous présenterons une alternative à cette approche et expliquerons où nous voulons que notre propre application se situe.

Nous rappelons que la notion de style englobe ici uniquement les systèmes de primitives et d'attributs (*cf.* section 1.3). Nous renvoyons le lecteur intéressé par la stylisation du système spatial aux travaux de Singh *et al.* [Sin02, SGS04, CS04] et de Yu *et al.* [YM04].

Les approches automatiques

Avec une approche automatique, l'utilisateur dispose d'une "boite noire" qui lui permet de fabriquer une image dans un style donné à partir d'une scène 3D quelconque. La figure 2.30 montre le deuxième étage du pipeline complet correspondant aux approches automatiques. En pratique, cette boite noire est une implémentation du pipeline de rendu analytique de

lignes (*cf.* figure 2.12), dans laquelle des choix de mise en style ont été faits, c'est-à-dire que les approches automatiques mêlent choix techniques et choix artistiques. L'intérêt d'une telle approche est que ce style est générique, dans la mesure où n'importe quelle scène 3D peut être utilisée en entrée. L'inconvénient réside dans le peu de contrôle fourni à l'utilisateur. En effet, ce dernier n'a en général, pour varier ses styles de rendu, que le choix de modifier quelques paramètres ou de changer de "boite noire".

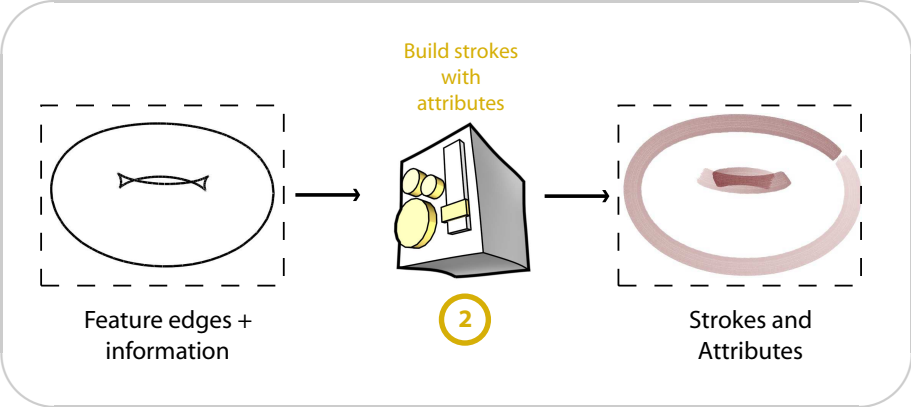


Fig. 2.30 – Étage de stylisation : approches automatiques

Dans les approches automatiques, une "boite noire" permet de rendre n'importe quel scène 3D dans un style donné. Ces approches mêlent ainsi choix techniques et artistiques et offrent peu de contrôle à l'utilisateur sur le style de rendu.

De très nombreux articles présentent ce type de moteur de rendu dans un style donné, celui-ci pouvant concerner les traits [SP03b, WS94, JWY02, WLS02] ou le remplissage de régions [GSG+99, GGSC98, Her98, Mei96]. Sousa *et al.* [SP03b], par exemple, programment un rendu de traits pour lequel l'épaisseur est d'autant plus importante que la courbure 3D l'est aussi, de manière à imiter les variations dans la quantité d'encre déposée, qui dépendent du geste de l'artiste.

Dans cette catégorie, l'article de Kowalski *et al.* [KMN+99] a plus particulièrement retenu notre attention et nous a inspiré. Il présente une technique de rendu NPR d'objets complexes touffus, tels que des feuillages (*cf.* figure 2.31 (a)), à l'aide de textures de traits procédurales. Si on l'illustre sur un arbre par exemple, leur approche consiste à démarrer avec une forme approchée de son feuillage (un ensemble de sphères par exemple, chaque sphère représentant un groupe de feuilles), et à rajouter des motifs de détails aux endroits pertinents lors du rendu. On peut voir ces formes approchées ainsi que le rendu final sur la figure 2.31. Dans leur système, l'ajout de détails est implémenté de manière procédurale, sous la forme de *graftals*. Cette procédure s'appuie par exemple sur la normale à la primitive ou sur la position du point de vue pour décider de tracer le motif ou non.

L'idée de pouvoir spécifier procéduralement les détails semble particulièrement intéressante et séduisante. En effet, on perçoit ici une liberté potentiellement infinie dans la représentation. Malgré l'accent mis sur cet aspect procédural, dans leurs travaux, Kowalski *et al.* ne permettent pas à l'utilisateur de définir ses propres procédures de rendu de *graftals*, renvoyant ainsi leur technique au groupe des approches automatiques standard.

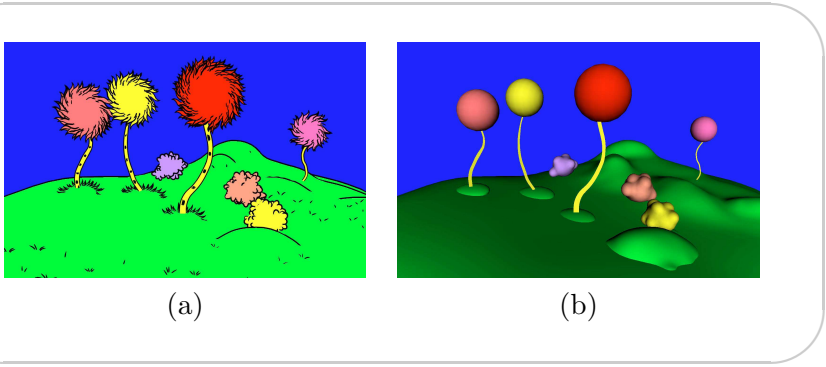


FIG. 2.31 – Rendu NPR procédural de feuillages à l'aide de *graftals*

(a) : Rendu NPR*. Les détails sont ajoutés de manière procédurale lors du rendu des primitives plus simples. (b) : Primitives de départ.* © *Kowalski et al. [KMN+99]*

Nous adoptons une approche similaire à l'approche procédurale mise en avant par Kowalski *et al.* en tâchant de donner à l'utilisateur le contrôle nécessaire sur l'aspect procédural de l'approche.

Les approches interactives

L'approche interactive consiste à modéliser un style en interagissant directement avec la visualisation de la scène, à la manière d'un logiciel de dessin tel que Photoshop ou Illustrator. Le style est alors construit par la suite des interactions effectuées par l'utilisateur. Si le principal inconvénient des approches automatiques résidait dans le manque de contrôle offert à l'utilisateur, c'est cet aspect qui constitue la principale force des approches interactives.

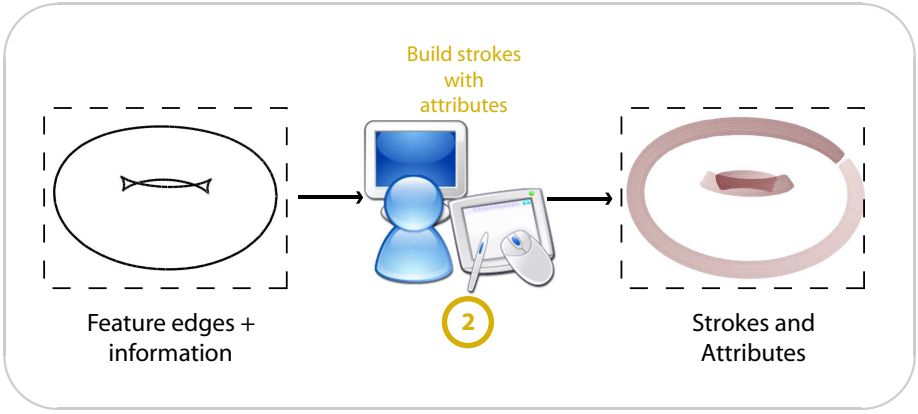


FIG. 2.32 – Étage de stylisation : approches interactives

Dans les approches interactives, l'utilisateur dessine directement sur la scène.

Kalnins *et al.* [KMM+02] par exemple présentent le système *WYSIWYG NPR*, qui permet de peindre interactivement des traits sur le modèle 3D, en plus des lignes de silhouettes déjà

présentes. Dans ce système il est également possible de modifier les attributs des traits de manière interactive : l'utilisateur peut dessiner un trait exemple dont les attributs seront reproduits sur l'ensemble des traits de la scène à l'aide de techniques d'*apprentissage* (*Machine Learning*).

L'inconvénient de ces approches est que le style ainsi généré est spécifique à la scène. Par exemple, si l'on considère l'illustration de la figure 2.33, les couleurs des traits ont été choisies interactivement et donc spécifiquement pour cette scène. Pour cette raison le style peut éventuellement être réutilisé pour l'ensemble des images d'une même animation mais son transfert à une autre scène ne peut pas se faire automatiquement.

FIG. 2.33 – Rendu stylisé : les approches interactives

Une image produite avec le système interactif de modélisation de style : WYSIWYG NPR. © Kalnins et al. [KMM+02]

Par contraste, notre approche est programmable et le style du dessin est controlé au travers de procédures. Ce type de contrôle requiert des capacités de programmation similaires à celles d'un directeur technique dans le rendu de productions, mais il fournit un contrôle plus puissant sur le style. Cela inclut en particulier l'utilisation d'informations génériques de manière à produire une description indépendante du modèle 3D. En terme de modélisation traditionnelle, la différence entre leur système et le nôtre est la même qu'entre la peinture interactive sur modèles 3D et les *shaders* procéduraux. Aujourd'hui les deux approches sont utilisées conjointement et nous pensons qu'une approche programmable est un complément similaire à des techniques interactives telles que *WYSIWYG NPR*.

Les approches mixtes

Vue d'ensemble On appelle "approches mixtes", les approches qui combinent un aspect interactif, au travers d'une phase d'édition, et un aspect automatique au travers du moteur de rendu. La figure 2.34 reprend l'étage de stylisation de la figure 2.12, sous la forme d'une approche mixte. Durant la phase d'édition, l'utilisateur spécifie de manière interactive un modèle de style. Cette phase d'édition présente les avantages des approches interactives. Ce modèle de style sera ensuite utilisé durant la phase de rendu pour spécifier le style à utiliser au moteur de rendu. Cette étape de rendu est généralement indépendante de la scène considérée et est en ce sens automatique.

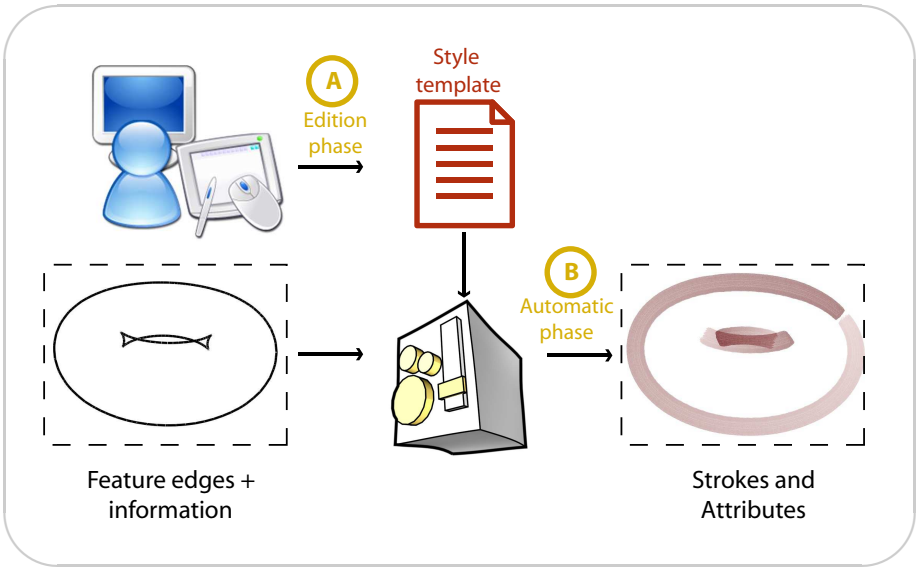


FIG. 2.34 – Rendu stylisé : les approches mixtes

Les approches mixtes combinent à la fois les approches interactives au travers d'une phase d'édition (A) et les approches automatiques à travers une phase de rendu automatique (B).

Dans ce type d'approches, il faut, premièrement, déterminer la forme sous laquelle le modèle de style sera stocké, et deuxièmement, décider de comment ce modèle de style sera spécifié par l'utilisateur. Nous nous intéressons aux différentes techniques existantes et envisageables pour la phase d'édition.

Phase d'édition Dans le cadre du NPR, une seule approche a jusqu'ici été envisagée pour cette phase d'édition, l'approche statistique.

Approches statistiques Le domaine de recherche de l'*apprentissage* (*Machine Learning*), qui s'est considérablement développé ces dernières années est particulièrement populaire en NPR et semble s'imposer comme la technique appropriée pour un certain nombre d'applications. En particulier, les recherches dans le domaine de la capture de style fondent l'espoir que le *Machine Learning* saura capturer, mieux même que l'humain, l'essence d'un style. Avec ce type d'approches, l'utilisateur fournit un certain nombre d'*exemples* qui permettent au système *d'apprendre* et de *reproduire*. Dans le contexte du rendu stylisé suivant une approche mixte, durant la phase d'édition, l'utilisateur aurait à fournir un exemple de style (selon une méthode interactive par exemple) et le système serait ensuite capable de réappliquer ce style (qu'il aurait appris) à d'autres modèles 3D. Un tel système combinerait à la fois contrôle flexible et intuitif du style (au travers de l'exemple) et automatisation, dans la mesure où le style appris serait réapplicable à n'importe quel modèle.

Hamel *et al.* [HS99b] sont les premiers à utiliser une approche de type mixte pour capturer et réappliquer un style de rendu de scènes 3D. Dans la phase d'édition, un utilisateur dessine interactivement directement sur la scène de manière à obtenir un ensemble de traits stylisés. Ils suggèrent de stocker un style sous forme d'un *patron*, qui met en relation les attributs de

style avec des informations génériques de la scène (*e.g.* courbures) sous forme d'histogrammes. Ainsi, si les relations attributs/informations ne sont jamais explicitement exprimées, en s'appuyant sur ces histogrammes, on est capable d'extraire un style et de l'appliquer à un modèle différent. En arrière-plan de cet article se dessinent les idées d'un moteur de rendu générique capable de rendre des styles variés, de stockage d'un style sous forme d'une feuille de style et de mise en relation des attributs de style avec des informations de la scène. Notre approche s'efforce de combiner les concepts de feuille de style et d'utilisation de l'information partiellement introduits par Hamel *et al.*.

Un certain nombre d'articles [TF97, FTP99, HS99b, HJO+01, HOCS02, Her03, KMM+02, JEGPO02] ont utilisé cette approche dans le cadre du rendu NPR stylisé. Ces techniques se focalisent habituellement sur les aspects statistiques et bas niveau des styles en cherchant à transférer des variations d'attributs (*e.g.* géométrie ou épaisseur) d'un trait exemple à un trait cible.

Pourtant, de nombreux travaux [WS94, DOM+01, Her01a, DS02a] ont montré que des variations subtiles plus globales pouvaient changer complètement le style d'une image ou permettre de focaliser le regard sur un point donné. À ce jour, les techniques basées sur le *Machine Learning* échouent à capturer des attributs de style de plus haut niveau. La difficulté est que, pour apprendre un style complexe, il est nécessaire de disposer d'un modèle de style, et qu'un tel modèle n'existe pas à ce jour.

Dans notre approche, les règles de style sont, au contraire du *Machine Learning*, explicitement manipulées par l'utilisateur. Le couplage du *Machine Learning* avec des méthodes dans lesquelles la connaissance est explicite nous semble très prometteur, en particulier pour la capture de style.

Approches programmables Existe-t-il une approche plus adaptée à la phase d'édition ?

Le travail qui a le plus inspiré notre approche est l'interface de programmation *Renderman* [Ups89, AG99, HL90, Coo84], qui est, de manière paradoxale, dédiée au rendu photoréaliste. *Renderman* est la spécification d'une interface standard entre des programmes de modélisation et des programmes de rendu capables de produire des images de qualité photoréaliste. *Renderman* est, d'une certaine manière, similaire au langage Postscript, mais pour des primitives 3D. Un aspect important de cette interface est le langage de *shading* : il casse le concept restrictif de surfaces décrites par un petit nombre de paramètres, en autorisant l'utilisateur à écrire ses propres descriptions, arbitrairement complexes, de comment l'éclairage et l'ombrage doivent être calculés. *Renderman*, et en particulier son langage de *shading*, ont apporté au rendu photoréaliste une flexibilité essentielle et ont grandement contribué à la qualité des rendus actuels. Comme nous l'avons mentionné au chapitre précédent, nous pensons qu'une telle approche pourrait d'autant plus profiter au rendu non-photoréaliste, que celui-ci exige encore plus de flexibilité dans la spécification du rendu.

Bien que notre approche s'inspire de systèmes tels que *Renderman* et ait, à ce titre, une philosophie qui en est proche, elle comporte des spécificités qui la différencient nettement de ces derniers : en premier lieu, l'utilisation de lignes comme éléments de dessin atomiques, auxquels un certain nombre de procédures sont appliquées, signifie que l'on travaille avec des objets ayant une étendue significative dans l'image, par opposition, par exemple, aux pixels pour *Renderman*. Deux propriétés additionnelles contribuent également à cette non localité du rendu. Premièrement, les propriétés du dessin à une certaine échelle, telles que sa densité

globale, peuvent affecter les lignes ou les traits individuels. Une autre différence avec les *shaders* procéduraux existant est que le dessin est créé par l'accumulation de marques dans l'image et est, par conséquent, produit de manière séquentielle : l'ordre des opérations, et la séquence de traits dessinés en résultant, influent sur le résultat final.

Nous nous sommes intéressés aux approches programmables existant dans le domaine du NPR. Il existe une approche programmable pour NPR, il s'agit du système *OpenNPAR* [HIR+03, Ope02, HSS02], qui est une API pour le développement de logiciels NPR temps réel. Notre approche est plus spécialisée que la leur puisque nous nous focalisons sur le dessin au trait. Cette spécialisation nous permet de fournir des outils plus puissants pour le développement de styles plus complexes. En pratique, leur système est une boite à outils pour la programmation permettant d'aider au développement de nouvelles applications NPR, dans l'esprit de *OpenInventor* [SC92], tandis que notre approche est davantage inspirée des *shaders* procéduraux de *Renderman*, et est dédiée à la production d'images. Le tableau suivant résume les principales différences entre l'approche de *OpenNPAR* et la nôtre :

	OpenNPAR	**Notre approche**
Comparable à	OpenInventor	Renderman
Objectif	Développement d'applications	Fabrication d'images
Utilisateurs	Développeurs	Directeurs techniques

Les auteurs ont en outre utilisé leur système pour développer une interface graphique impressionante pour l'exploration de styles simples [HSS02].

Résumé: Les travaux de recherche s'étant intéressés au style en NPR appartiennent majoritairement à deux catégories : d'un côté les approches automatiques, qui permettent une réutilisation du style mais offrent à l'utilisateur un faible contrôle sur le style, et de l'autre côté, les approches interactives, qui, par contraste, offrent un excellent contrôle mais sont peu adaptées à la réutilisation. Une troisième catégorie tente de concilier le meilleur des deux approches, celle des approches mixtes qui combinent une approche interactive, au travers d'une phase d'édition et une approche automatique, au travers de la phase de rendu. Le contrôle du style s'effectue lors de la phase d'édition. Cette dernière a jusqu'ici été envisagée uniquement sous l'angle des techniques d'apprentissage qui échouent à capturer les composantes haut-niveau du style. Nous choisissons d'adopter une approche programmable, inspirée de *Renderman*, pour mettre en œuvre cette phase d'édition de manière plus complète et performante.

---

chapitre 3

Modélisation de style

Le but de cette thèse, on l'a vu, est d'offrir, dans le cadre du rendu de dessins au trait stylisés à partir de scènes 3D, un contrôle flexible sur le style de l'image produite et, d'assurer que ce style sera réutilisable sur n'importe quelle scène 3D. Il s'agit de fournir des outils permettant de spécifier un style. Mais qu'est-ce que le style ? Comment le décrire de façon générique (*i.e.* non spécifique à la scène considérée pour sa modélisation) ? Avec quels outils ?

On a vu en introduction que nous limitons la notion de style aux seuls systèmes de primitives et d'attributs, écartant ainsi toute stylisation pouvant intervenir dans le système spatial (*e.g.* projections cubistes) et limitant celles relatives au système de marques. Dans la suite du document, le terme "style" ne concernera que ces deux seuls systèmes.

Dans ce chapitre, nous commençons par donner l'intuition de ce qu'est le style et une méthodologie pour sa description en s'appuyant sur des illustrations faites par des artistes. Ensuite nous identifions les attributs de style ainsi que les informations génériques pertinentes dans le cadre d'une description de style. Enfin, nous motivons notre choix d'une approche programmable et donnons un aperçu du système dans son ensemble.

3.1 Observation

Le style des traits joue un rôle prépondérant dans l'aspect visuel d'une illustration

La première question qu'il est légitime de se poser au sujet de ce travail est de savoir si le style, tel qu'on l'entend ici, représente un élément important de l'aspect visuel d'un dessin ou non. En réalité, bien que l'on en ait peu conscience, l'aspect visuel d'un dessin, résultant des attributs des traits, de leur manière d'être organisés dans le dessin, est souvent suffisamment spécifique pour que l'auteur puisse être reconnu sans ambiguïté, et ceci indépendamment de la scène représentée et de la colorisation. De même, deux représentations au trait d'une même scène par deux artistes différents seront aisément distinguables, comme on peut le voir sur la figure 3.1.

Ainsi, le style des lignes est un élément suffisamment caractéristique d'une illustration pour permettre d'attribuer une illustration à son auteur. Comment alors caractériser ce style ?

On commence par regarder des exemples d'illustrations réelles (dessins de lignes) en tâchant de décrire pour chacune d'elles le style en s'appuyant sur des informations génériques, comme si un interlocuteur n'ayant pas l'illustration sous les yeux devait en reproduire le style sous notre dictée.

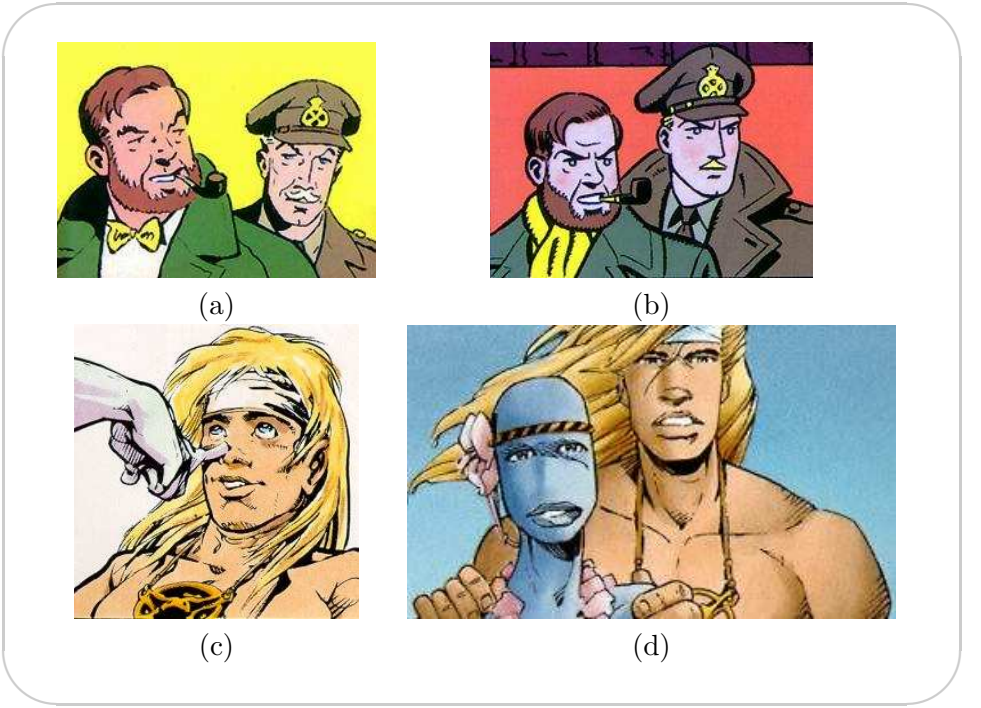


FIG. 3.1 – L'importance du style des lignes dans l'aspect visuel d'un dessin

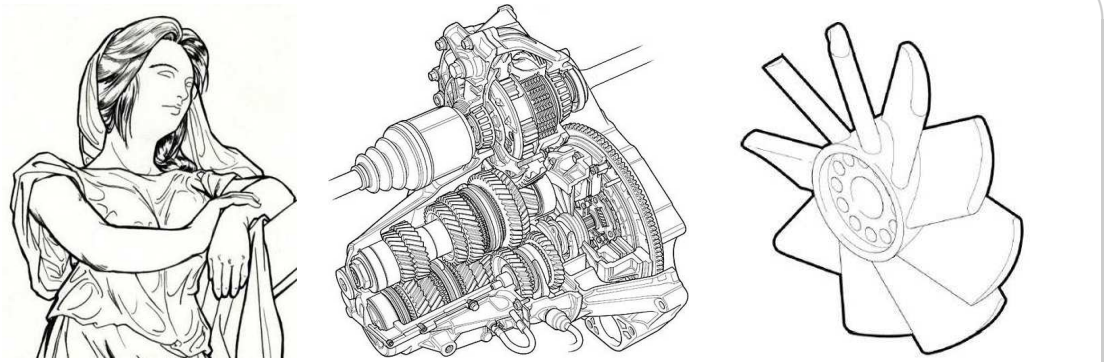
(a) et (b) : Blake et Mortimer dessinés respectivement par Edgar P. Jacobs et Ted Benoit. Bien que les mêmes personnages soient représentés, on voit aisément qu'ils ont été réalisés par deux artistes différents. On peut entre autres remarquer que Jacobs utilise plus de traits que Benoit, et que ses traits sont légèrement moins précis et plus fins. © *Edgar P. Jacobs 1947-1977 - Blake et Mortimer [Jac77] et* © *Ted Benoit 1996 - Blake et Mortimer [Ben96]. (c) et (d) : Le personnage Nao de la série Aquablue dessiné respectivement par Vatine et Tota. Ici aussi, bien que le même personnage soit représenté dans les deux illustrations, ces dernières ont des aspects très différents. Le trait de Vatine est plus sûr et appuyé que celui de Tota, résultant en des traits de poids globalement plus importants. Par ailleurs, chez Vatine on constate de grandes différences d'épaisseur d'un trait à l'autre, tandis qu'ils sont d'épaisseur uniforme chez Tota.* © *Cailleteau et Vatine 1989 - Delcourt [CV89a] et* © *Cailleteau et Tota 1998 - Delcourt [CT98].*

On observera en particulier les *attributs* de style (*e.g.* épaisseur, couleur des lignes) et leur relation aux *informations* de la scène représentée (*e.g.* nature des lignes, profondeur).

■ *Le poids des lignes est lié à leur nature*

Les trois illustrations de la figure 3.2 ont en commun un certain nombre de caractéristiques visuelles. En particulier, les lignes délimitent les objets avec précision et chacune d'elle est monochrome et d'épaisseur constante. Nous sommes ici très proche du style du courant de la *ligne claire*, symbolisé par Hergé. Pourtant, elles présentent toutes trois une caractéristique supplémentaire prédominante : le poids des lignes varie d'une ligne à l'autre dans le dessin.

Y-a-t'il une raison pour le choix de ces différents poids ? Peut-on relier le choix de chaque poids à une information de la scène ? On distingue en fait dans les trois images deux poids différents, un faible et un fort. Dans les deux premières illustrations, ce sont les lignes ap-



(a) (b) (c)

FIG. 3.2 – Poids des lignes liée à leur nature

(a) et (b) : les lignes de silhouette sont dessinées avec un poids plus important que les autres lignes. (c) : les lignes du contour extérieur sont tracées avec un poids important. Dans tous les cas, l'attribut de poids des lignes peut être mis en relation avec l'information de nature de ces lignes. (b) et (c) © *Kevin Hulsey Illustration Inc. - www.khulsey.com/.*

partenant à la silhouette des objets qui sont tracées avec le poids le plus fort, de manière à améliorer la perception de ces objets.

À partir de ces observations on peut tenter de décrire le style de ces deux illustrations sous une forme pesudo-algorithmique :

```
pour chaque ligne l à dessiner
  si nature(l) = silhouette
    dessiner l avec un poids fort
  sinon
    dessiner l avec un poids faible
```

Dans la troisième illustration, le poids fort est utilisé pour les lignes se trouvant sur le contour extérieur de l'objet. C'est la forme globale de l'objet qui est ainsi soulignée. Comme précédemment on peut ici encore tenter d'exprimer ce style sous une forme peudo-algorithmique :

```
pour chaque ligne l à dessiner
  si nature(l) = contour extérieur
    dessiner l avec un poids fort
  sinon
    dessiner l avec un poids faible
```

Pour décrire ces deux styles nous avons mis en relation le poids des lignes avec leur nature (*e.g.* silhouette ou contour extérieur). L'attribut caractéristique du style est ici le poids des lignes tandis que l'information est la nature de ces lignes. Ces deux descriptions de style s'appuyant sur l'information générique de nature des lignes, elles sont elles-mêmes génériques. On pourrait donc envisager d'appliquer ces deux styles à un ensemble de lignes provenant d'une scène quelconque.

Remarque : Dans ces illustrations et dans les suivantes, les dépendances entre les attributs de style et les informations résultent d'une démarche plus ou moins consciente de l'artiste. En règle générale, ces dépendances ne sont pas formulées explicitement par l'artiste lors de la réalisation du dessin. Toutefois, il existe des cas pour lesquels on trouve des règles spécifiant explicitement la valeur des attributs de style en fonction d'informations. Ainsi, en illustration technique les poids des traits y sont explicitement codifiés en fonction de la nature des lignes [J.M89]

.

■ *Les variations d'épaisseur ou la couleur des lignes sont liées au matériau*

(a) (b)

FIG. 3.3 – Attribut de couleur des lignes lié au matériau

(a) et (b) : La couleur des traits dessinant un objet est reliée au matériau de ce dernier. © Disney 1940 - Pinocchio [Dis40] et © Disney 1937 - Blanche neige et les sept nains [Dis37]. Pour ces illustrations, l'attribut est la couleur et l'information générique est le matériau.

Les illustrations de la figure 3.3 se caractérisent surtout par le fait que différentes couleurs ont été utilisées pour tracer les lignes. On peut à nouveau tenter de relier le choix des différentes couleurs à une information de la scène : on remarque facilement que la couleur choisie par l'artiste pour chaque ligne d'un objet est une variation de la couleur de cet objet. Par exemple, les traits de la cape de couleur bleue sont tracés en bleu plus foncé. D'une manière générale, lorsque le matériau est de couleur foncée, les lignes sont tracées avec un ton plus clair et inversement lorsque le matériau est clair. Avec cet exemple, on peut imaginer proposer une formule permettant de calculer automatiquement la couleur des lignes en fonction de celle du matériau[1], et exprimer à nouveau le style sous forme pseudo-algorithmique :

[1]Par exemple, en passant dans l'espace LUV [WS82b] et en translatant la composante L dans une direction ou une autre selon que la couleur est sombre ou claire.

```
pour chaque ligne l à dessiner
  obj ← objet(l) // on récupère l'objet auquel l appartient
  mat ← materiau(obj) // on récupère le matériau de cet objet
  c   ← couleur(mat) // et la couleur de ce matériau
  c'  ← f(c) // on calcule la variation de couleur
  dessiner l avec la couleur c'
```

(a) (b)

FIG. 3.4 – Attribut d'épaisseur des lignes lié au matériau

(a) et (b) : ce sont ici les variations d'épaisseur qui dépendent du matériau. Pour ces illustrations, l'attribut est l'épaisseur et l'information générique est le matériau.

Les illustrations de la figure 3.4 se caractérisent, quant à elles, par d'importantes variations d'épaisseur au sein des traits. On remarque également que certaines lignes sont de poids moyen plus important que les autres et que l'on peut ainsi séparer l'ensemble des traits en deux groupes, un groupe de poids fort et l'autre de poids faible. Pour ces deux illustrations, les lignes dont l'amplitude des variations d'épaisseur est la plus importante correspondent aux vêtements, tandis que les lignes dessinant le visage et les autres parties du corps sont globalement plus fines. On peut donc ici aussi relier ce choix d'amplitude de variations au matériau, celle-ci étant par exemple d'autant plus importante que le matériau est rugueux. Le peudo-algorithme décrivant ce style est alors :

```
pour chaque ligne l à dessiner
  obj ← objet(l) // on récupère l'objet auquel l appartient
  mat ← materiau(obj) // on récupère le matériau de cet objet
  r   ← rugosité(mat) // et la rugosité de ce matériau
  si r > seuil
    dessiner l avec d'importantes variations d'épaisseur
  sinon
    dessiner l avec de faibles variations d'épaisseur
```

Pour ces deux styles on a mis en relation les attributs respectifs de couleur et d'épaisseur avec l'information de matériau. À nouveau, ces deux descriptions de style sont génériques dans la mesure où l'information de matériau est elle-même générique. On peut donc imaginer de transposer ces styles à n'importe quelle autre scène.

L'épaisseur des lignes est liée à la discontinuité en profondeur

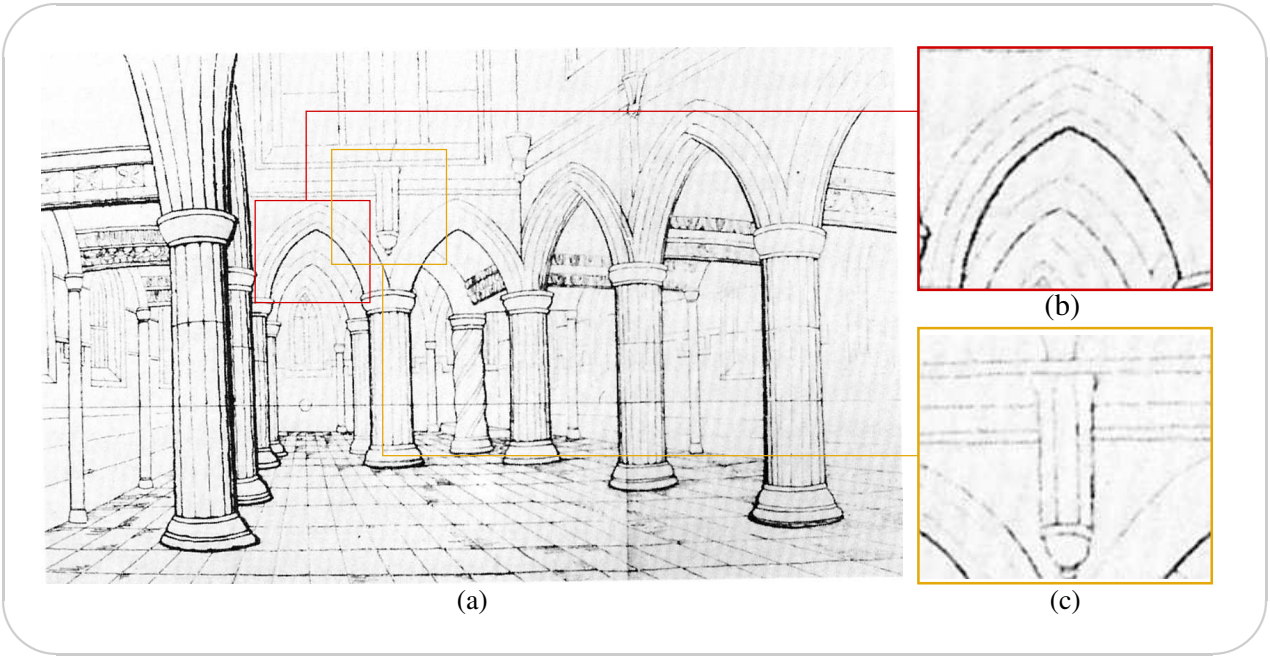


FIG. 3.5 – Épaisseur des lignes liée à la discontinuité de profondeur

(a) : Dans cette illustration, les lignes de silhouette marquant de fortes discontinuités de profondeur sont tracées avec un poids plus important. (b) : gros plan sur un élément dont la silhouette marque une forte discontinuité en profondeur. (c) : dans ce détail, la silhouette de la colonne ne correspond pas une forte discontinuité en profondeur, elle est donc tracée avec un poids faible. © William Herdman 1850 - [Her50]

Le style de l'illustration de la figure 3.5 est ici encore dû à l'utilisation de poids différents pour les différentes lignes du dessin. Cette fois, on peut remarquer que les lignes de poids le plus fort correspondent aux silhouettes marquant de fortes discontinuités en profondeur (la discontinuité en profondeur est définie précisément dans la section 3.3. La figure 3.21 en est une illustration). Si la distance entre l'objet à dessiner et le plus proche objet situé derrière est importante, les lignes de silhouette seront tracées avec un poids important. Ainsi, en chaque point d'une ligne de silhouette, l'épaisseur de la ligne est proportionnelle à la discontinuité en profondeur. On peut donc définir le style de cette illustration de la manière suivante :

```
pour chaque ligne l à dessiner
  pour chaque point p de l
    d ← discontinuité en profondeur en p // on calcule la discontinuité
                                         // en profondeur en p
    épaisseur(l) en p ← f(d) // on assigne à la ligne l au point p une
                             // épaisseur proportionnelle à d
```

L'omission des lignes est liée à la densité du dessin

FIG. 3.6 – Omission de lignes liée à la densité du dessin

Dans ce dessin, l'artiste a omis de nombreuses lignes de manière à conserver une densité "constante" sur l'ensemble du dessin. L'omission de lignes dépend donc de la densité constatée sur le dessin.

Jusqu'ici, nous avions concentré nos observations sur les aspects de style "bas niveau", c'est-à-dire la couleur ou l'épaisseur des traits. Pour l'illustration de la figure 3.6, nous allons laisser ces aspects de côté et nous concentrer sur un élément de style de plus haut niveau : l'omission de lignes.

Il est très courant dans l'illustration au trait, d'omettre un certain nombre de détails, en particulier lorsque ceux-ci sont trop petits, de manière à ne pas surcharger le dessin. Ce type de simplification est notamment inévitable dès que la perspective est importante dans la représentation de la scène. La quantité de détails représentés, le choix dans les lignes à omettre, *etc.*... varient d'un artiste à l'autre et font partie intégrante de son style.

On peut ici se poser la question de savoir s'il est possible de trouver, dans la scène, une raison pour l'omission de telle ligne plutôt que d'une autre. Puisque le but d'une telle simplification est de finalement obtenir un dessin de densité raisonnable, on peut imaginer que l'omission de lignes dans une zone donnée du dessin doit être d'autant plus importante que la densité de lignes y est déjà grande. On peut ainsi tenter de relier l'omission de lignes à une information extraite, non pas de la scène, mais du dessin lui-même : la densité de lignes déja tracées. Les pavés du sol, par exemple, sont représentés avec de moins en moins de traits

au fur et à mesure que l'on s'éloigne du point de vue et que la densité de traits s'accentue sous l'effet de la perspective. On peut alors proposer une interprétation pseudo-algorithmique de ce style :

```
pour chaque ligne l à dessiner
  d = densité du dessin en l
  si d < seuil
    dessiner l
  sinon
    omettre l
```

Le chapitre 6 étudie en détail l'information de densité et propose différents scénarios de simplification basés sur cette information.

Bien sûr, cette information seule ne suffit pas à expliquer complètement les choix qui ont été faits dans cette illustration concernant l'omission de lignes. Si on regarde le dessin du feuillage de l'arbre de gauche par exemple, on remarque que certaines zones sont complètement vides alors que d'autres sont représentées en détail. On verra dans le chapitre 6 que ce type de simplification est tout de même rattaché à l'information de densité.

■ *Ces descriptions de styles s'appuient sur des informations génériques et sont donc réutilisables*

Dans tous ces exemples, on a caractérisé les attributs spécifiques à chaque style, mis en évidence des dépendances entre ces attributs et des informations générique de la scène (*e.g.* nature des lignes, discontinuité en profondeur) ou du dessin (dans le cas de la densité), et finalement décrit chaque style, au moins en partie, sous une forme pseudo-algorithmique en explicitant ces dépendances. Une telle description de style, utilisant des informations génériques (*i.e.* non spécifiques à la scène), possède l'avantage de pouvoir être réutilisée d'une scène à l'autre.

Notre approche du style pour le NPR est née de ces observations et de l'hypothèse qui en découle naturellement : le style d'un dessin peut être décrit de manière non spécifique à la scène représentée. Si l'on est capable d'identifier, d'un côté les attributs qui contribuent au style d'un dessin et les informations qui interviennent dans les choix faits pour ces attributs, et d'un autre côté les opérateurs qui les lient, il est envisageable de proposer des outils de description de style dans le cadre du rendu non-photoréaliste de dessins au trait. Ces outils s'appuyant sur des informations génériques, les descriptions de style pourront être réutilisées d'une scène à l'autre ou d'une image à l'autre dans le cadre d'une animation.

Remarque : Il arrive d'observer des styles s'appuyant fortement sur des informations de type sémantique. S'il est souvent possible d'interpréter cette sémantique comme une combinaison d'informations génériques, dans certains cas il sera impossible de formuler un style sous la forme d'un algorithme générique.

Résumé: Le style d'un dessin peut être formulé de manière pseudo algorithmique, comme une mise en relation d'attributs de style avec des informations extraites de la scène.

Dans la suite de ce chapitre, nous abordons les questions suivantes :
– “Quels attributs composent le style d’un dessin ?”
– “Quelles sont les informations (génériques) pertinentes dans le cadre de la mise en style d’un dessin ?”
– “Quelle est l’approche la plus adaptée à la modélisation de style ?”

3.2 Attributs de style

La section précédente nous a donné un aperçu de ce que pouvaient être les attributs de style, avec par exemple, l’épaisseur des traits ou leur couleur, mais aussi le choix des traits à omettre. Dans cette section, nous détaillons les attributs de style que nous considérons dans notre approche. Ils peuvent être séparés en deux groupes : les attributs de bas niveau et les attributs de haut niveau. Alors que les attributs de bas niveau sont relativement classiques et ont été identifiés comme composants de style depuis longtemps déjà, les attributs de haut niveau ont, jusqu’à ce jour été peu traités, et leur identification constitue l’une de nos contributions.

3.2.1 Bas niveau

Les attributs de bas niveau du style déterminent l’apparence d’un trait donné dans un dessin. On en identifie quatre sortes, illustrées par la figure 3.7 :

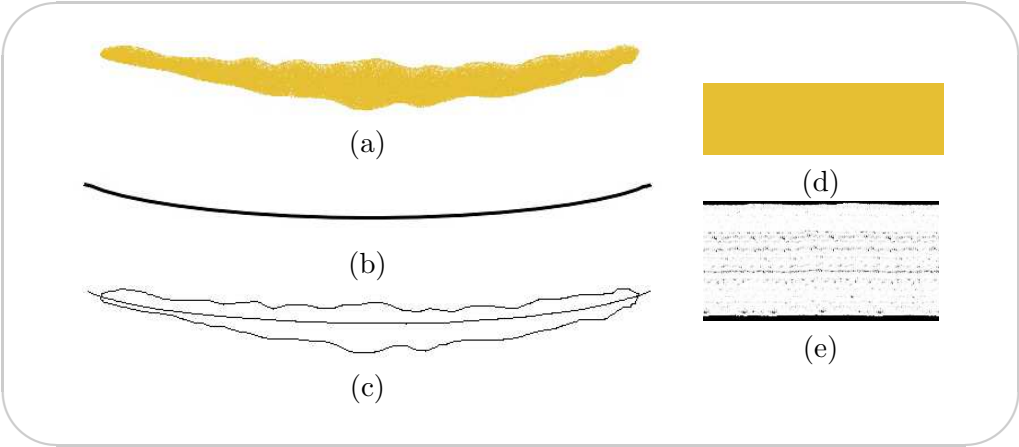


FIG. 3.7 – Les attributs de style bas niveau

(a) L’apparence d’un trait est caractérisée par, (b) sa géométrie, (c) son épaisseur, (d) sa couleur et (e) sa texture.

Géométrie La géométrie de l’épine dorsale d’un trait, illustrée figure 3.7(b).

Couleur La ou les couleurs du trait (*cf.* figure 3.7(d)).

Épaisseur L’épaisseur du trait en chacun de ses points, de part et d’autre de l’épine dorsale (*cf.* figure 3.7(c)).

Texture La texture qui simule l’interaction de l’outil et du medium sur le support (*cf.* figure 3.7(e)).

3.2.2 Haut niveau

Ces attributs de bas niveau ne suffisent pas à décrire complètement un style. Nous identifions trois composantes supplémentaires du style qui reflètent des décisions correspondant à une considération plus globale de la scène : la topologie des traits, l'omission de lignes et l'utilisation de styles multiples au sein d'un même dessin.

Topologie des traits

Nous commençons par définir quelques notions utiles à une bonne compréhension de ce qu'est la topologie des traits.

> **Définition :** On appelle *graphe de vue*, le graphe définissant l'apparence topologique de la scène sous le point de vue considéré. Il est construit comme un graphe planaire, à partir des lignes caractéristiques extraites de la scène (le graphe de vue est étudié en détail au chapitre 4)

> **Remarque :** Le graphe de vue est analogue à l'*aspect*, défini dans la théorie sur les *graphes d'aspect* [PD90b], augmenté des arêtes cachées.

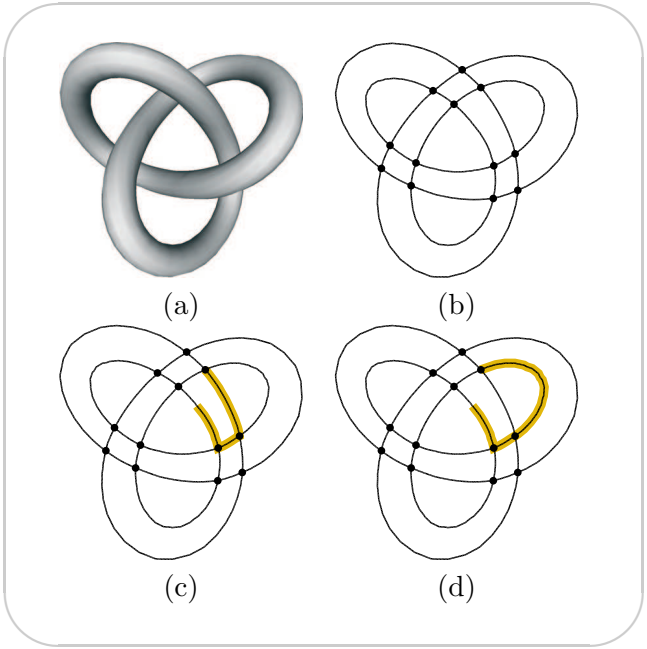


FIG. 3.8 – La topologie des traits : un chemin dans le graphe de vue

*(a) : Un modèle 3D (*torus knot*). (b) : Le graphe de vue qui définit son apparence topologique sous le point de vue considéré. (c) et (d) : Deux chemins différents de ce graphe.*

On appelle topologie d'un trait, le chemin suivi par ce dernier dans le dessin. D'un point de vue plus pratique, si l'on considère le graphe planaire formé par l'ensemble des lignes à

dessiner, c'est le chemin suivi par le trait dans ce graphe qu'on appelle la topologie du trait (*cf.* figure 3.8). Cette topologie reflète la décision de l'artiste de commencer et de terminer son trait en tel ou tel point ainsi que de le poursuivre selon tel ou tel ligne. C'est une caractéristique remarquable du style d'un artiste, en particulier lorsque l'outil ou le medium utilisé impliquent d'importantes variations d'épaisseur ou de couleur le long d'un tracé.

Lorsqu'un ensemble de lignes connectées est tracé à l'aide d'un seul trait plutôt que de plusieurs, le résultat a une apparence de continuité, comme on peut le voir sur la figure 3.9. En effet, le geste de l'artiste, les variations de pression induites, l'évolution potentielle de la quantité de pigments le long du tracé produisent une apparence caractéristique. Ainsi, deux représentations d'une même scène réalisées par deux artistes utilisant respectivement de longs et de courts traits auront des aspects très différents.

Remarque : La topologie d'un trait est indépendante de la topologie 3D de la scène. Ainsi, dès qu'ils sont connectés dans l'image, des segments de lignes appartenant à plusieurs objets différents peuvent se retrouver liés au sein d'un même trait.

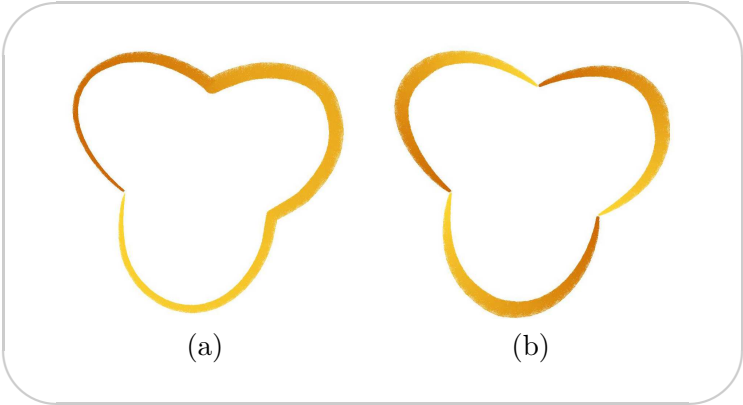


FIG. 3.9 – La topologie des traits et son influence sur les attributs bas niveau

*(a) : un seul trait dessine l'ensemble des segments du contour externe de ce modèle (*torus knot*). L'évolution de l'épaisseur et de la couleur de ce trait est continue tout au long de l'objet. (b) : Le contour externe est cette fois-ci dessiné à l'aide de trois traits.*

Omission de lignes

Comme on l'a mentionné dans la section précédente, l'omission de lignes est également caractéristique du style d'un artiste. Cette technique peut être utilisée dans un but de simplification purement esthétique (*cf.* figure 3.10 (a)), afin de concentrer l'attention sur une zone d'intérêt (en simplifiant les zones d'intérêt moindre) (*cf.* figure 3.10 (b)), ou encore pour ne pas surcharger le dessin lorsque la scène représentée est visuellement complexe (*cf.* figure 3.10 (c))

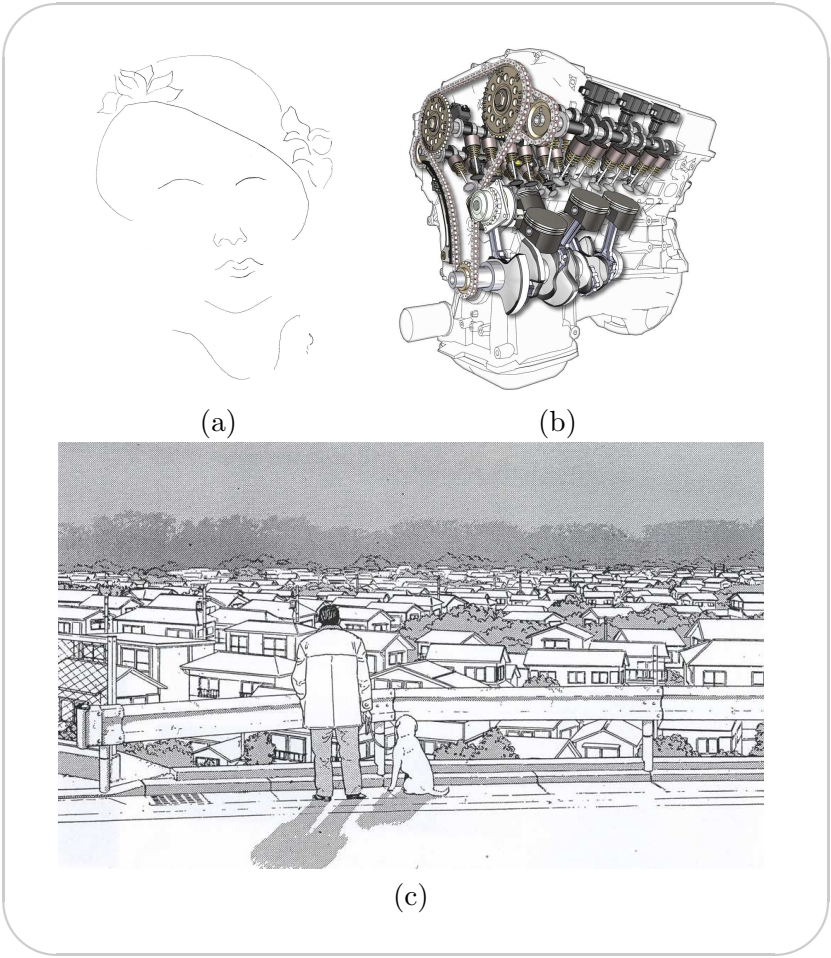


FIG. 3.10 – Omission de lignes

(a) : Matisse a ici dessiné une version épurée d'un portrait dans laquelle la plupart des lignes verticales ont été omises. © Matisse 1869-1954 - [Mat54] (b) : Sur cette illustration technique, l'auteur a voulu mettre en évidence un sous-ensemble des pièces du moteur, notamment en éliminant la plupart des lignes appartenant aux autres parties. © Kevin Hulsey Illustration Inc. - www.khulsey.com/. (c) : Les détails des toits des maisons ont été omis de manière à ne pas surcharger le dessin. © Jiro Taniguchi 2002 - Casterman [Tan02].

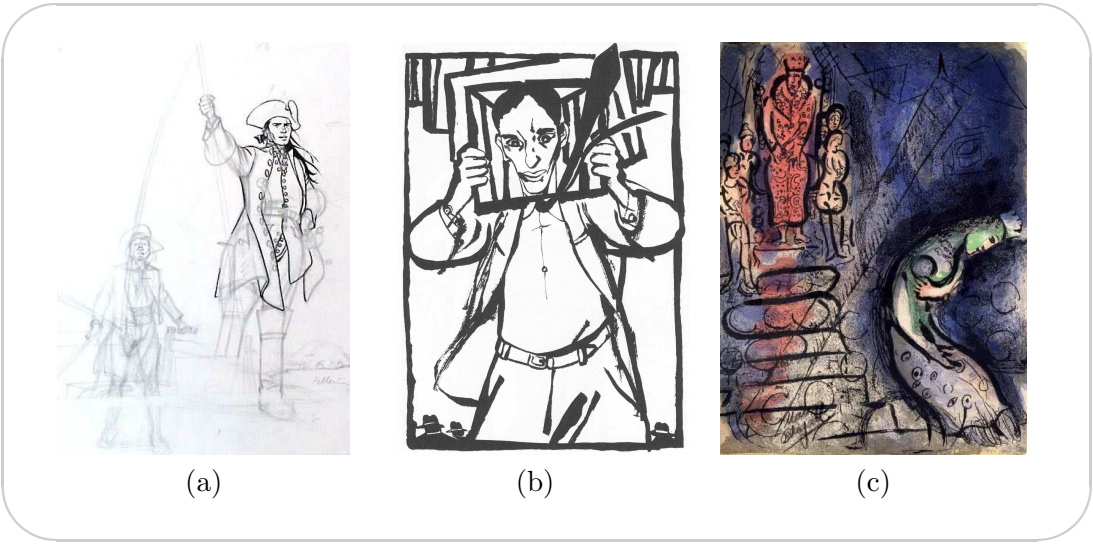
Utilisation de multiples styles de lignes au sein d'un même dessin

Finalement, le dernier attribut de style que nous considérons consiste à utiliser plusieurs styles pour tracer les traits d'un même dessin.

On distingue le cas où les sous-ensembles de lignes de même style s'intersectent, du cas où ils forment une partition de l'ensemble des lignes.

Le premier cas signifie qu'un certain nombre de lignes sont tracés dans plusieurs styles à la fois : ces lignes sont donc dessinés à l'aide de plusieurs instances (plus ou moins superposés) du même trait, chaque instance étant tracée dans un seul des styles. On observe en particulier ce cas de figure sur les dessins en cours de réalisation pour lesquels l'esquisse est encore visible, et où une partie des traits finaux ont déja été tracés, comme illustré par la figure 3.11 (a).

Dans le deuxième cas, chaque ligne est dessinée par un seul trait, mais les styles varient d'une ligne à l'autre. Cette technique peut par exemple servir à mettre en évidence un groupe de lignes par rapport à un autre. On a par exemple observé cette technique sur la figure 3.4. De même, sur les deux dessins des figures 3.11 (b) et (c), les artistes respectifs utilisent deux styles différents pour différentes lignes du même dessin.



(a) (b) (c)

FIG. 3.11 – Utilisation de multiples styles de lignes au sein d'un même dessin

(a) : Sur ce dessin inachevé, on observe deux styles : le premier, crayonné, peu précis et haché, correspond à l'esquisse du dessin. Le deuxième, à l'encre noire, plus précis et continu est utilisé pour tracer les premières lignes définitives du visage et de la veste du personnage de premier plan. Pour les lignes du visage et de la veste qui ont été tracées dans les deux styles on a donc deux traits par ligne. © Pellerin 2001 - Dupuis [Pel01]. (b) : Sur ce dessin également, deux styles de traits ont été employés : pour les contours et les grandes lignes, l'auteur trace des traits noirs très épais, peu précis, aux variations d'épaisseur importantes. En revanche, les lignes correspondant aux détails plus petits (e.g. lignes du visage) sont tracées à l'aide de traits d'une finesse qui contraste avec le premier style. C'est l'association de ces deux styles de trait dans le même dessin qui produit cet aspect visuel esthétique caractéristique de l'artiste. © Baudoin 1996 - L'Association [Bau96]. (c) : Sur cette peinture de Chagall, on observe le même type de variations de style que pour l'image précédente : des traits épais pour les contours et des traits fins pour les détails. Cette combinaison est également assez caractéristique des œuvres de Chagall. © Marc Chagall 1960 - Drawings for the bible [Cha60].

Résumé: Un style est composé d'attributs bas niveau, qui sont la géométrie, l'épaisseur, la couleur et la texture des traits, et d'attributs haut niveau, qui sont la topologie des traits, la sélection des lignes à dessiner et l'utilisation de multiples styles au sein d'un même dessin.

3.3 Informations

Comme nous avons pu le voir lors de nos observations au début de ce chapitre, nous travaillons avec l'hypothèse que les décisions stylistiques sont en partie liées à des informations génériques extraites de la scène. Dans cette section nous tâchons d'identifier et de lister l'ensemble des informations qui peuvent prendre part à ces décisions stylistiques. Cette identification se fonde en partie sur notre propre expérience et en partie sur quelques règles communément reprises par les ouvrages traitant de l'apprentissage ou de l'étude du dessin. L'objectif final étant de produire un système de rendu travaillant sur des scènes 3D, on suppose ici que l'ensemble de la géométrie de la scène représentée est accessible.

Coordonnées 3D

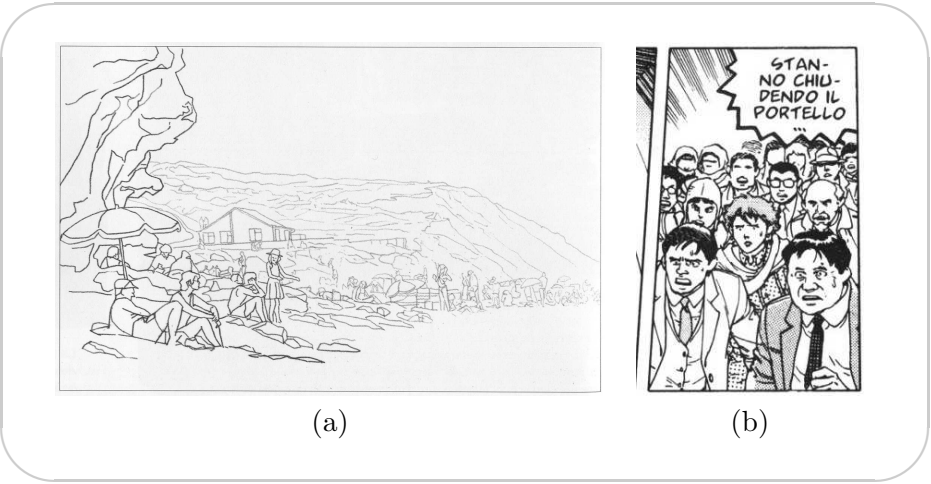
Il est courant d'utiliser les coordonnées 3D pour modifier la représentation d'une zone de l'espace dans la scène. Sur l'illustration de la figure 3.12 par exemple, les traits ont été omis dans la région 3D regroupant le bas des immeubles de manière à renforcer l'impression de brouillard.

FIG. 3.12 – Informations - Coordonnées 3D

Les traits correspondant représentant des éléments correspondant à la région 3d regroupant le bas des immeubles de deuxième plan ont été omis afin de donner l'impression d'un brouillard. © Bilal 1997 - Hazard edizioni [Bil97]

Profondeur (distance à l'observateur)

La profondeur peut par exemple servir à paramétrer un ou plusieurs attributs bas niveau de style. En particulier, il est courant de l'utiliser pour paramétrer le poids des traits, de manière à renforcer l'impression de perspective. La figure 3.12 (a) illustre cette technique. Cette même coordonnée de distance à l'observateur peut encore servir à paramétrer des attributs haut niveau de style tels que l'omission de lignes. Il est en effet fréquent d'omettre d'autant plus de lignes que les objets représentés sont éloignés de la caméra, comme illustré par la figure 3.13 (b).



(a) (b)

FIG. 3.13 – Informations - Profondeur

(a) Le poids des lignes est proportionel à la profondeur. Les lignes des premiers plans sont plus appuyées que celles du fond. © Smith et al. 1995 - Dorling Kindersley Limited [SWHW95] (b) L'omission de lignes est d'autant plus importante que les visages dessinés sont loin de la caméra. © Katsuhiro Otomo 1996 - Editions Delcourt [Oto96]

Coordonnées 2D

De même que les coordonnées 3D, les coordonnées 2D peuvent être utilisées pour paramétrer des effets de style directement associés à la géométrie de la scène après projection. On peut par exemple diviser l'image en plusieurs zones et utiliser un style différent pour les traits de chaque zone. Ce type d'effet est illustré sur la figure 3.14.

Normales 3D (Orientation 3D)

En rendu photoréaliste, les normales 3D permettent notamment de calculer l'ombrage d'une surface à partir des positions des sources lumineuses. Elles sont donc souvent utilisées de manière inconsciente par les artistes pour imprimer à leurs dessins une information d'ombrage et peuvent ainsi prendre part à la définition du style de l'illustration. La figure 3.15 illustre cette propriété.

Normales 2D

La plume et le pinceau font partie des outils les plus utilisés pour encrer les lignes d'une illustration, en particulier dans le monde de la bande dessinée. Les pleins et les déliés créés par ces outils offrent un aspect calligraphique apréciable. En supposant que l'outil garde une orientation constante le long d'un tracé, les zones de pleins et de déliés ne dépendent que de la direction de la ligne, c'est-à-dire de la normale 2D à cette ligne. La figure 3.16 montre ces effets sur une illustration faite au pinceau.

Fig. 3.14 – Informations : coordonnées 2D

Dans ce dessin l'artiste parvient à faire exister une aire triangulaire au-dessus de la tête du personnage de premier plan juste en utilisant un style plus léger pour les traits qui s'y trouvent. © Baudoin 1996 - L'Association [Bau96]

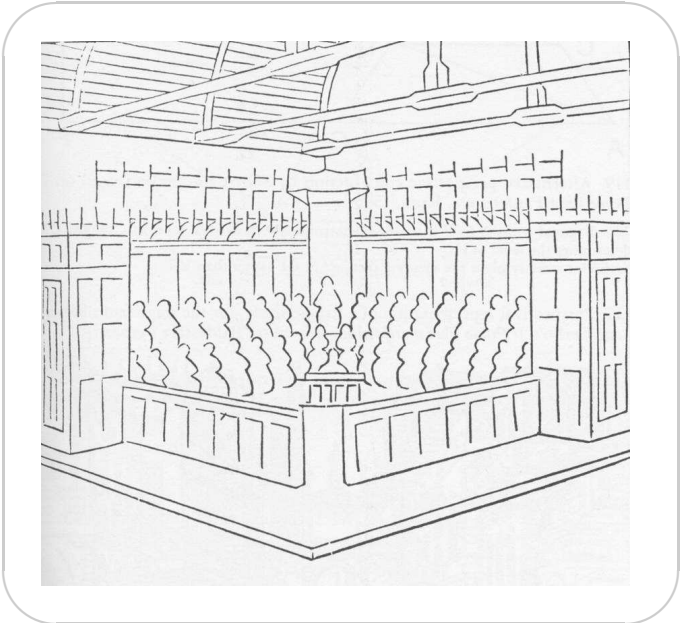


Fig. 3.15 – Informations : normales 3D

Sur cette illustration, il semble que tous les traits correspondent aux lignes de silhouette "tournées" vers un point central à la scène, semblant ainsi indiquer l'ombrage. Ce sont les normales 3D qui permettent de dire si une surface est orientée vers un point donné de l'espace.

FIG. 3.16 – Informations : normales 2D
L'aspect calligraphique observé sur cette illustration est dû à l'utilisation du pinceau pour tracer les traits. Les zones de pleins et de déliés sont en particulier liées à la direction des traits, c'est-à-dire aux normales à ces traits. © Tome et Janry 1998 - Dupuis [TJ98].

Courbure 2D

La courbure des traits peut également influer sur le style. En particulier, avec certains outils tels que le pinceau, l'épaisseur du trait varie selon l'angle d'attaque du pinceau sur le support et selon la pression exercée. Pour certains artistes, l'épaisseur d'un tel trait est d'autant plus importante que sa courbure est forte.

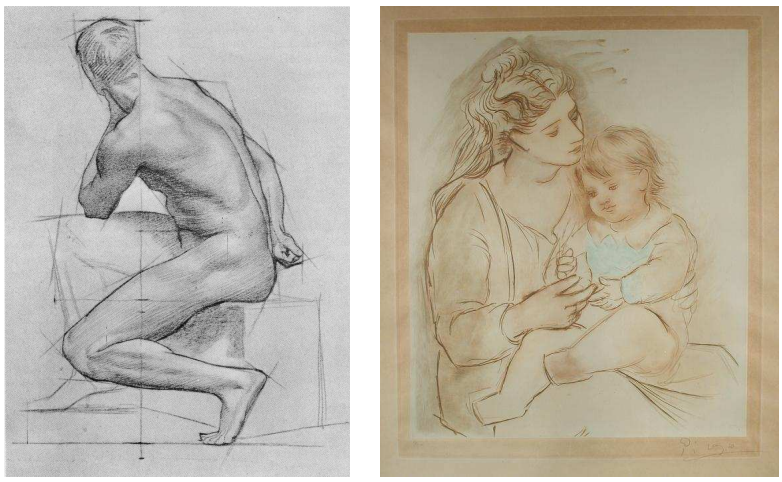
Par ailleurs, de manière assez naturelle les points de courbure importante correspondent souvent aux points de début et de fin de traits. En effet, il est plus facile de tracer avec précision un trait quasi rectiligne qu'un trait fortement courbé. Cette observation est encore plus nette sur les esquisses pour lesquelles l'artiste approche les formes courbes à l'aide de séries de traits rectilignes (*cf.* figure 3.17).

Nature des lignes

Nous avons déja observé dans la section 3.1, et en particulier sur l'illustration 3.2, que la nature des lignes (*e.g.* silhouette, contour extérieur) pouvait décider de leur épaisseur. En illustration technique cette règle de style est explicite et définit l'épaisseur de chaque ligne en fonction de sa nature [J.M89, Tay03]. Dans la figure 3.18, les lignes sont catégorisées en fonction de leur nature et un poids différent est attribué à chaque catégorie.

Invisibilité (quantitative)

Dans le domaine de l'illustration technique, l'information de visibilité est une des données les plus utilisées dans la mise en style : certaines représentations peuvent nécessiter l'affichage des lignes cachées de manière à offrir une perception plus complète d'un objet ; dans ce cas, pour une raison de lisibilité, les lignes cachées sont affichées dans un style plus discret



(a) (b)

FIG. 3.17 – Informations : courbure 2D

(a) et (b) : dans ces deux illustrations on aperçoit encore les premières esquisses de formes : sous forme de grandes lignes de construction pour la première et de séries de traits quasi-rectilignes pour la seconde. On constate que ces traits initiaux ont tendance à débuter et stopper à des points de haute courbure. (b) : © Pablo Picasso 1922 - [Pic22].

que les lignes visibles. Dans l'illustration de la figure 3.19 (a), les lignes cachées sont par exemple affichées en blanc. On remarque également que toutes les lignes cachées ne sont pas affichées. En effet, encore pour une raison de lisibilité, on préfèrera tracer les lignes cachées par un nombre d'objets inférieur à un nombre donné. L'information qui est alors utilisée est l'*invisibilité quantitative* (*cf.* 2.2.3), c'est-à-dire l'information définissant, pour chaque ligne, le nombre de surfaces qui la cachent. Sur la figure 3.19 (b), on utilise l'invisibilité quantitative pour déterminer la couleur des lignes.

Objets occultants

L'information donnant, pour une ligne, la liste des objets qui la cachent est également fréquemment utilisée en illustration technique. Par exemple, pour afficher lisiblement un objet qui est normalement caché, la norme est de tracer toutes ses lignes (y compris les lignes cachées), à l'exception de celles qui sont cachées par l'objet lui-même. C'est l'information d'occultation qui permet de connaître, pour une ligne donnée, quelle est la liste des objets qui la cachent, et donc de savoir si cette ligne est notamment cachée par l'objet auquel elle appartient. Cette technique est illustrée par la figure 3.20

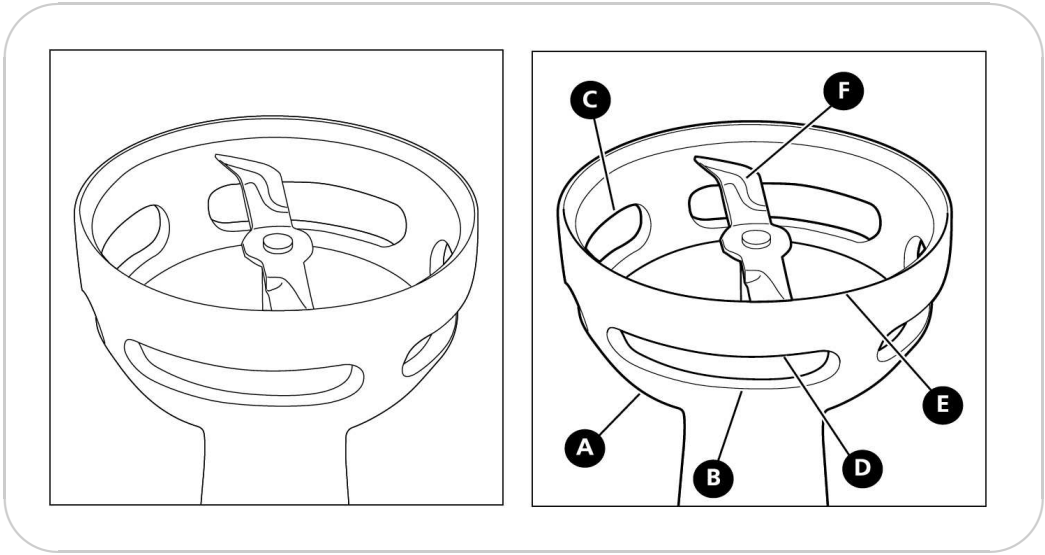


FIG. 3.18 – Informations : nature des lignes

Dans cette illustration des poids ont été attribués aux lignes en fonction de leur nature : les lignes A et C sont des contours extérieurs (elles tracent une frontière entre l'objet et le fond) et ont été tracées avec le poids le plus fort ; la ligne D est un contour (elle trace une frontière entre l'objet et un autre objet), elle est tracée avec un poids légèrement moins important ; la ligne E est une silhouette sur sa moitié inférieure et une arête vive sur sa moitié supérieure. Elle a donc été tracée avec la même épaisseur qu'un contour sur sa partie silhouette et avec un trait plus fin sur sa partie arête vive ; enfin, la ligne F est une arête vive, tracée avec le poids le plus faible. © *Conrad Taylor 2003 - [Tay03].*

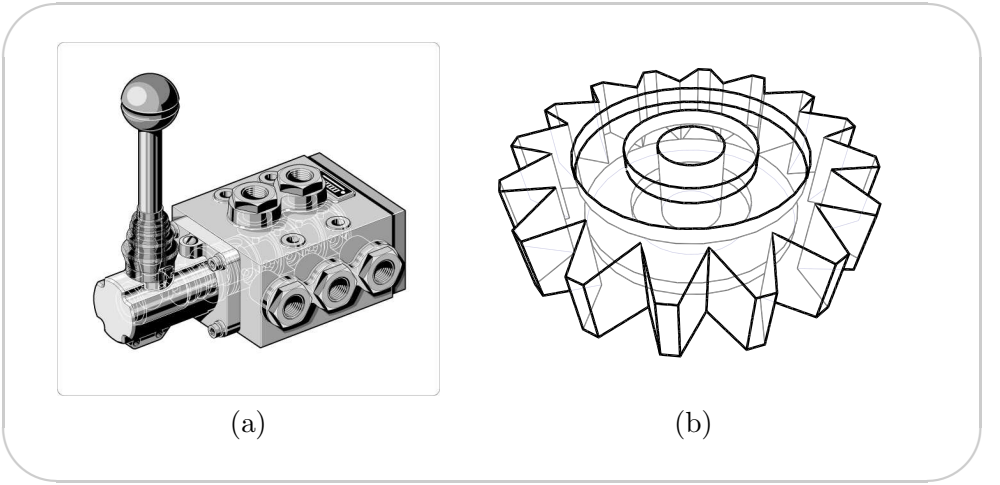


FIG. 3.19 – Informations : invisibilité (quantitative)

(a) Les lignes cachées sont affichées en blanc, dans un style plus discret que les lignes visibles. © ITEDO - *www.itedo.com (b) La couleur et le poids des lignes sont attribués en fonction de leur information d'*invisibilité quantitative *: les lignes visibles sont noires et épaisses, celles cachées par une seule surfaces sont plus claires et plus fines et celles cachées par deux surfaces sont encore plus claires et encore plus fines.*

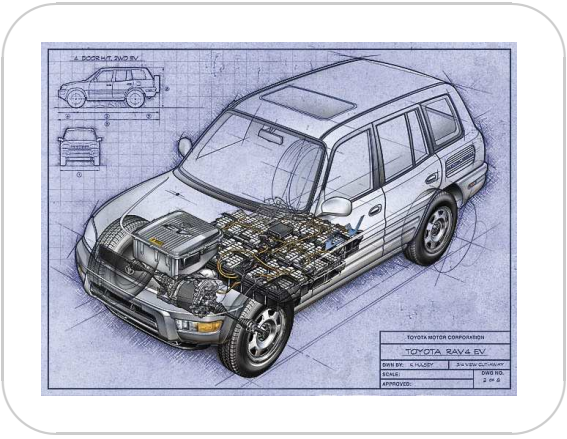


FIG. 3.20 – Informations : objets occultants

Sur cette illustration, un sous-ensemble de pièces du moteur, normalement cachées par la carrosserie, sont dessinées : les lignes tracées sont les lignes de ces objets qui ne comptent pas, parmi leurs obstacles, l'un des objets du sous-ensemble d'intérêt. © *Kevin Hulsey Illustration Inc. -* `www.khulsey.com/`.

Distance à l'objet caché (discontinuité en profondeur)

Discontinuité en profondeur : La discontinuité en profondeur en un point d'une ligne est la distance qui le sépare du plus proche objet rencontré en suivant le rayon partant du point de vue et passant par ce point (*cf.* figure 3.21). Elle n'est définie que pour les points appartenant aux lignes de silhouette. Elle est par nature nulle en tout point pour les lignes de nature différente.

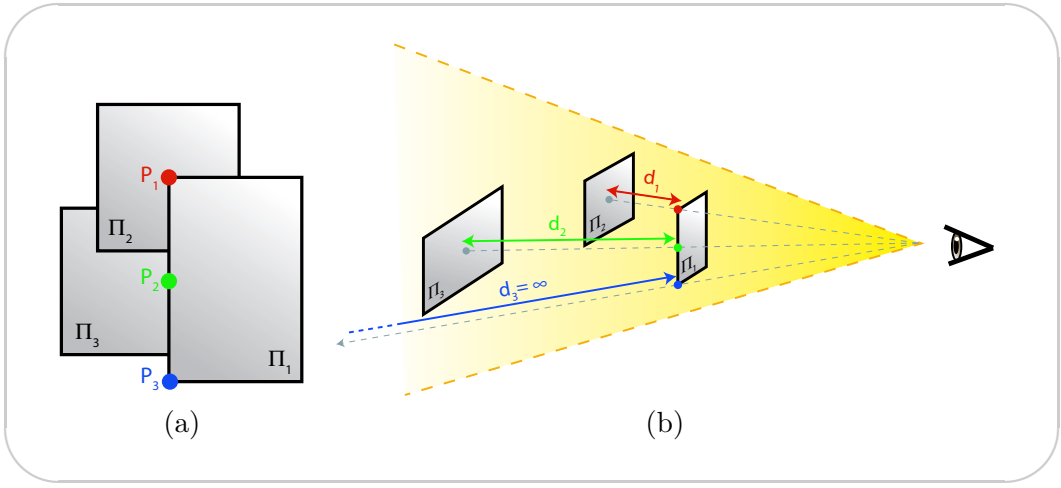


FIG. 3.21 – Informations : discontinuité en profondeur

Si l'on considère trois rectangles de l'espace observés selon le point de vue montré en (a), les discontinuités en profondeur aux points P_1, P_2 et P_3 correspondent aux distances d_1, d_2 et d_3 illustrées en (b).

L'information de discontinuité en profondeur est l'une des informations les plus utilisées dans le processus de dessin. Elle peut servir aussi bien pour des attributs de bas niveau que de haut niveau. Par exemple, dans le cadre de la simplification de dessin au trait, le problème se pose de savoir quelles lignes garder en priorité. Il est alors courant de conserver en premier les silhouettes marquant une forte discontinuité en profondeur.

Ou encore, dans la représentation de paysages, l'artiste détoure fréquemment les différents plans du fond d'une scène d'un seul trait (*cf.* figure 3.22). Inconsciemment, c'est encore l'information de discontinuité en profondeur qui est utilisée pour déterminer la topologie du trait délimitant chaque plan : ce trait relie en effet entre eux des contours se situant, premièrement à une profondeur similaire et deuxièmement présentant chacun une forte discontinuité en profondeur. Finalement, cette information est également très fréquemment utilisée pour contrôler les attributs bas niveau du style. Ainsi, les artistes choisissent souvent de dessiner de manière plus appuyée les contours ayant de fortes discontinuités en profondeur en jouant sur la couleur ou l'épaisseur. On a, par exemple, vu sur la figure 3.5 que les contours marquant une forte discontinuité en profondeur étaient dessinés avec un trait plus épais.

Fig. 3.22 – Discontinuité en profondeur : contour des arrière-plans

Sur cette illustration, on voit que les différents plans du fond sont chacun détourés par un contour unique. Tous les points de ce contour sont à une distance similaire du point de vue, et présentent une forte discontinuité en profondeur. Ainsi, la règle de construction de chacun de ces traits de contour s'appuie notamment sur la discontinuité en profondeur. © *Franquin 1952 - Dupuis [Fra52].*

Surfaces cachées

Si, comme on vient de le voir, la distance à l'objet caché représente une information souvent exploitée dans la mise en style d'un dessin, l'identité même de cet objet peut être une information intéressante et plus particulièrement en illustration technique. En effet, il arrive qu'un élément d'un assemblage soit mis en valeur en omettant l'ensemble des lignes des objets se trouvant entre cet élément et le point de vue. Lors du dessin d'une ligne, il convient, dans

ce cas, de se poser la question de savoir si cette ligne délimite une portion d'un objet cachant l'élément intéressant, de manière à décider de son élimination éventuelle. L'information dont il faut disposer est donc, pour chaque ligne, la liste des surfaces cachées. Cette technique est illustrée par la figure 3.23.

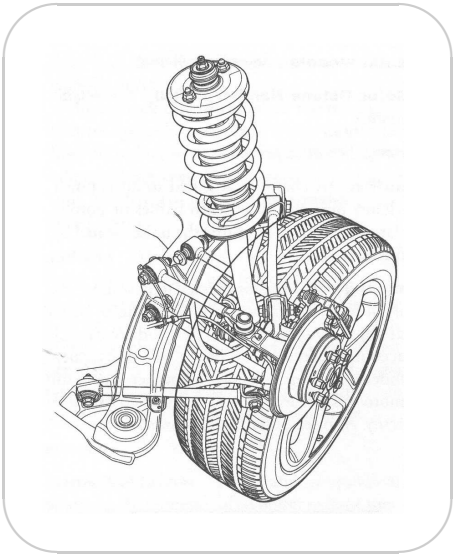


FIG. 3.23 – Informations : surfaces cachées
Les lignes qui contribuaient à cacher des parties du système de freinage à disques ont été omises de manière à afficher entièrement ce système. © ITEDO - *www.itedo.com*

Matériau

Comme on l'a vu dans la section 3.1, l'épaisseur ou la couleur des traits peuvent être liées au matériau des objets qu'ils délimitent (*cf.* figures 3.3 et 3.4).

Identité des objets

Il est également courant de tracer les lignes d'un premier objet dans un certain style et celles d'un autre objet dans un style différent, de manière à mettre en valeur un objet dans le dessin par exemple. L'information qui est alors notamment utilisée est l'identité des objets. La figure 3.24 montre une illustration sur laquelle deux objets différents ont été dessinés dans deux styles différents afin de mieux les distinguer.

Densité de la vue et du dessin

Comme on l'a vu avec la figure 3.6, la simplification d'un dessin peut être liée à une information de densité. Lorsque la complexité de la scène à représenter augmente, l'artiste pourra omettre des lignes de manière à conserver un dessin de densité raisonnable. Ainsi, les décisions d'omission de lignes dans une optique de simplification peuvent être liées à la fois à la complexité de la scène à représenter et à la densité des traits déjà tracés sur le dessin.

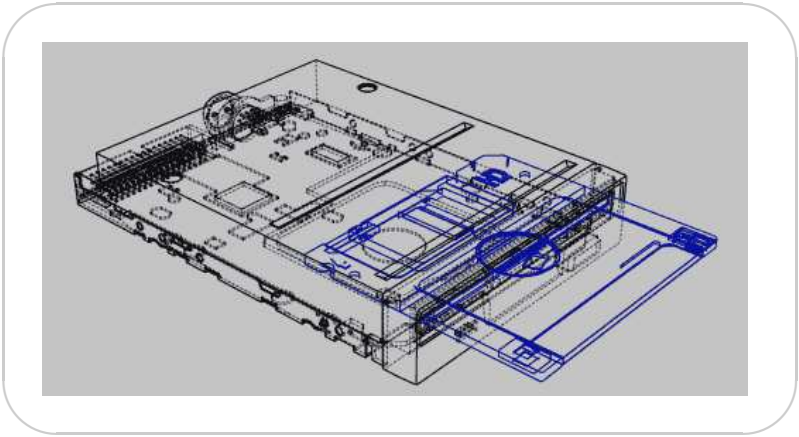


FIG. 3.24 – Informations : identité des objets

Dans cette illustration technique, deux styles différents ont été utilisés pour la disquette et le lecteur : les lignes de ce dernier ont été tracées à l'aide de traits pleins noirs pour les lignes visibles et en pointillés noirs pour les lignes cachées, tandis que pour la disquette, les lignes ont été dessinées en bleu.

La simplification de dessins au trait à partir de ces informations est étudiée en détail dans le chapitre 6.

Adjacence d'autres traits

Un style classique consiste à épaissir les lignes aux jonctions en T. L'artiste utilise alors l'information d'adjacence des lignes de deux manières. D'une part pour identifier les jonctions, et d'autre part pour que l'épaississement d'une extrémité de ligne respecte l'orientation de la ligne avec laquelle se produit la jonction. La figure 3.25 (a) montre une illustration pour laquelle ce style a été utilisé et en particulier un gros plan sur une jonction en T.

Il est également courant de marquer la séparation entre différents objets placés à différents plans (pour mettre en valeur ces objets ou pour rendre le dessin plus clair) en produisant un effet de "halo" autour des objets. Cela consiste à omettre les segments extrêmes des traits connectés à l'objet autour duquel le halo est souhaité. Cet effet est illustré par la figure 3.25 (b).

Remarque : La liste d'informations présentée précédemment est le fruit d'une longue période d'observations et d'expérimentations. Il est important d'être exhaustif dans cet inventaire dans la mesure où l'ensemble de l'approche s'appuie sur ce dernier et où tout style s'appuyant sur une information n'y figurant pas ne pourra pas être modélisé par un système implémenté suivant notre approche (à moins de modifier le système lui-même).

Résumé: Nous avons présenté un ensemble d'informations qui semblent jouer un rôle dans les décisions stylistiques prises par un artiste. Ces informations sont génériques dans la mesure où elles existent dans n'importe quelle scène 3D.

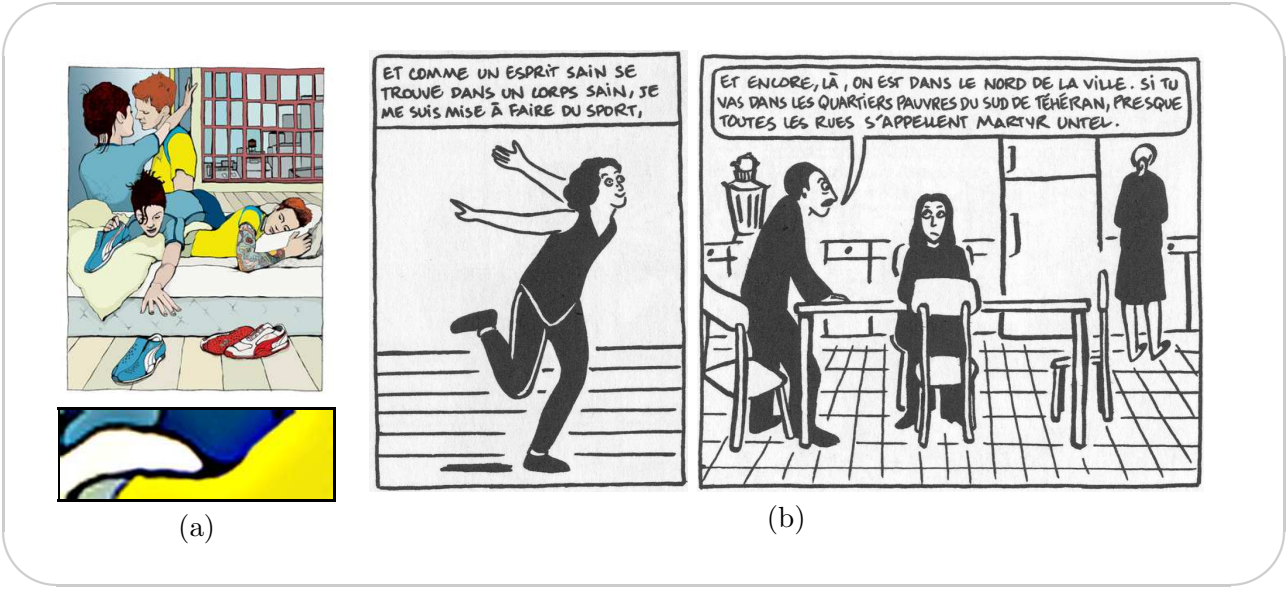


FIG. 3.25 – Informations : adjacence du graphe de vue

(a) Le style de cette illustration est principalement dû aux variations d'épaisseur des traits. Ces derniers ont la particularité d'être souvent plus épais aux extrémités qu'au centre. En particulier, on observe un effet de diffusion aux jonctions en T. L'image du bas est un grossissement centré sur un tel effet. Ce style utilise une double information du graphe de vue. La première est l'information indiquant quels sont les sommets correspondant aux jonctions en T et la deuxième permet, à une telle jonction, de connaître l'orientation de la ligne occultante de manière à produire un épaississement cohérent avec celle-ci. © Puma - 2003. (b) Dans cette illustration, lorsqu'un objet en cache un autre, l'artiste omet de façon quasi systématique les extrémités des lignes de l'objet caché qui touchent l'objet de premier plan, créant ainsi un "halo" autour de ce dernier. Ici encore, c'est l'information d'adjacence du graphe de vue qui est utilisée afin de savoir, parmi les lignes de l'objet caché, quelles sont celles qui sont en contact avec l'objet de premier plan. © Marjane Satrapi 2003 - L'Association [Sat03].

3.4 Approche programmable

Les hypothèses auxquelles nous mènent ces observations et sur lesquelles s'appuie cette recherche sont que :

- dessiner peut être vu comme un processus, c'est-à-dire comme une suite d'opérations élémentaires permettant de passer de l'ensemble des lignes à dessiner aux traits stylisés ;
- les décisions stylistiques incluses dans ce processus sont reliées au contenu de la scène et du dessin ;
- ces relations sont suffisamment génériques pour pouvoir être exploitées sur n'importe quelle scène ;
- elles peuvent être exprimées sous forme algorithmique.

Dans le contexte d'un système de rendu au trait de scènes 3D, on peut donc envisager de calculer ces informations génériques et de contrôler les attributs de style des traits en exprimant leurs relations à ces informations au sein du processus de dessin. Ce dernier consiste alors en la fabrication (et au rendu) d'un ensemble de traits stylisés à partir de l'ensemble des lignes caractéristiques (*e.g.* silhouettes) détectées sur le modèle, en suivant les règles de

stylisation. Dans la mesure où c'est à l'utilisateur de spécifier le style, il est essentiel qu'un tel système offre un contrôle flexible sur le processus de dessin. Comment faire ?

■ *L'approche programmable est le moyen naturel de spécifier explicitement un processus*

Nous avons vu section 2.4 que les approches mixtes sont une alternative intéressante aux approches automatiques, qui ont l'inconvénient de mêler les choix artistiques et techniques dans la phase de conception et de laisser un trop faible contrôle à l'utilisateur, et aux approches interactives qui ne permettent pas la réutilisation d'un style. Elles permettent ainsi de bénéficier d'une approche qui est d'une part automatique, dans la mesure où l'utilisateur crée un *patron* de style générique réutilisable sur d'autres modèles, et qui, d'autre part, offre un contrôle flexible sur la spécification de ce style. La question qui se pose alors est : quelle technique utiliser pour la phase d'édition, durant laquelle l'utilisateur doit produire une feuille de style réutilisable ? Nous avons vu section 2.4.2 que jusqu'ici la plupart des travaux de recherche s'appuyaient sur des techniques d'apprentissage (*Machine Learning* pour inférer une feuille de style à partir d'exemples fournis par l'utilisateur. L'inconvénient de ces techniques est qu'elles échouent à capturer les aspects haut niveau du style. Dès lors, l'autre alternative consiste à laisser à l'utilisateur la charge de spécifier directement la feuille de style. Nous choisissons d'adopter une approche programmable, inspirée de *Renderman*, pour la mise en œuvre de la phase d'édition. En effet, une telle approche semble s'imposer naturellement dans la mesure où elle répond aux objectifs fixés, de :

Flexibilité L'utilisateur doit pouvoir spécifier des styles variés le plus librement possible (et surtout ne pas être limité à un choix parmi une liste de styles prédéfinis). Dans une approche programmable, l'utilisateur décrit directement le style à l'aide d'un langage dédié à cet usage. En outre, seul un langage de programmation est à même de permettre la spécification des relations entre attributs de style et informations, qui peuvent être arbitrairement complexes.

Généricité En favorisant la formulation des styles sous une forme algorithmique, l'approche programmable est la plus adaptée à l'objectif fixé de produire une description de style générique, c'est-à-dire réutilisable sur n'importe quelle scène 3D.

Ainsi, une description de style aura la forme d'un programme, qui aura été écrit par l'utilisateur à l'aide du langage dédié fourni. Cette description de style est comparable aux *shaders* de *Renderman* et sera réutilisable de manière à permettre le rendu de n'importe quelle scène 3D dans le style visé. Le rendu photoréaliste a beaucoup gagné en puissance grâce à de tels systèmes [Ups89, Apo99] qui semblent d'autant mieux s'adapter au rendu non-photoréaliste que ce dernier exige encore davantage d'expressivité.

Le choix d'une telle approche pour modéliser le processus de dessin suppose, premièrement qu'on décompose ce processus en un ensemble d'opérations élémentaires et que deuxièmement on permette à l'utilisateur de programmer directement le comportement de chacune de ces opérations.

■ *Le processus de dessin doit être décomposé en une base d'opérateurs programmables*

Dans notre cas, il s'agit donc d'identifier l'ensemble des opérateurs qui constituent le processus de dessin et de mettre au point une architecture qui permette à l'utilisateur de les programmer. La décomposition du processus de dessin en cette base d'opérateurs revient par ailleurs à proposer une méthodologie de modélisation du style.

Il est important lors de la décomposition du processus de dessin en opérations de veiller à adopter à chaque fois la granularité la plus adaptée. En effet, les opérateurs de description de style ne doivent pas être de grain trop gros afin de rester flexibles et expressifs. Ils ne doivent pas non plus être de grain trop fin de manière à représenter un apport conséquent à la programmation d'un style (par rapport à une programmation ne les utilisant pas) : en pratique, seule la partie de plus haut niveau du code de description de style est laissée à la charge de l'utilisateur.

Concrètement, suivre l'approche programmable signifie qu'il faut fournir à l'utilisateur un environnement de programmation auquel sont intégrés un ensemble de structures et d'opérateurs (programmables), dédiés à la description de style. Les opérateurs doivent être suffisamment génériques pour permettre la modélisation de n'importe quel style mais également suffisamment riches pour représenter une méthodologie efficace de la modélisation de style. Un accès facile et complet aux informations de la scène (*e.g.* nature des lignes, profondeur) doit par ailleurs également être fourni à l'utilisateur. Finalement, l'environnement de programmation doit être suffisamment évolué pour permettre la formulation d'opérations complexes et supporter une "programmation interactive", c'est-à-dire que le code doit être interprété, ou compilé à la volée et exécuté, suffisamment rapidement pour pouvoir visualiser interactivement les effets de modifications apportées au programme, de manière à autoriser une expérimentation confortable.

Résumé: Dans notre approche programmable, le style est spécifié par l'utilisateur comme une suite de procédures mettant en relation les attributs de style et les informations de la scène.

3.5 Vue d'ensemble du système

Nous donnons ici une vue d'ensemble de la mise en œuvre de cette approche programmable de manière à rendre les objets manipulés plus concrets. La figure 3.26 montre le pipeline correspondant au processus de dessin.

La seule interaction avec l'utilisateur se situe dans la création de la description de style et est représentée à l'aide d'une flèche orange sur le schéma. Une fois celle-ci faite, tout est automatique.

Nous choisissons de travailler en entrée avec un modèle 3D de type maillage polygonal. Idéalement, la surface correspondant à ce maillage est une variété de dimension 2. Une structure de connectivité est générée à partir du maillage. Dans notre implémentation nous construisons une structure *winged-edge* [Gla91]. Une telle structure permet entre autres d'accéder aux faces adjacentes à une arête, ou encore à l'ensemble des arêtes partant d'un sommet, et est nécessaire au calcul des lignes caractéristiques.

La seconde étape consiste à détecter sur le modèle les lignes caractéristiques correspondant au point de vue choisi, et à les organiser sous forme d'un graphe. Les lignes caractéristiques considérées incluent les lignes de silhouette, les arêtes vives et les contours suggestifs [DFRS03]. Le graphe ainsi construit est appelé le graphe de vue (*View Map*). Sa construction et ses propriétés sont étudiés dans le prochain chapitre en section 4.2. Pa-

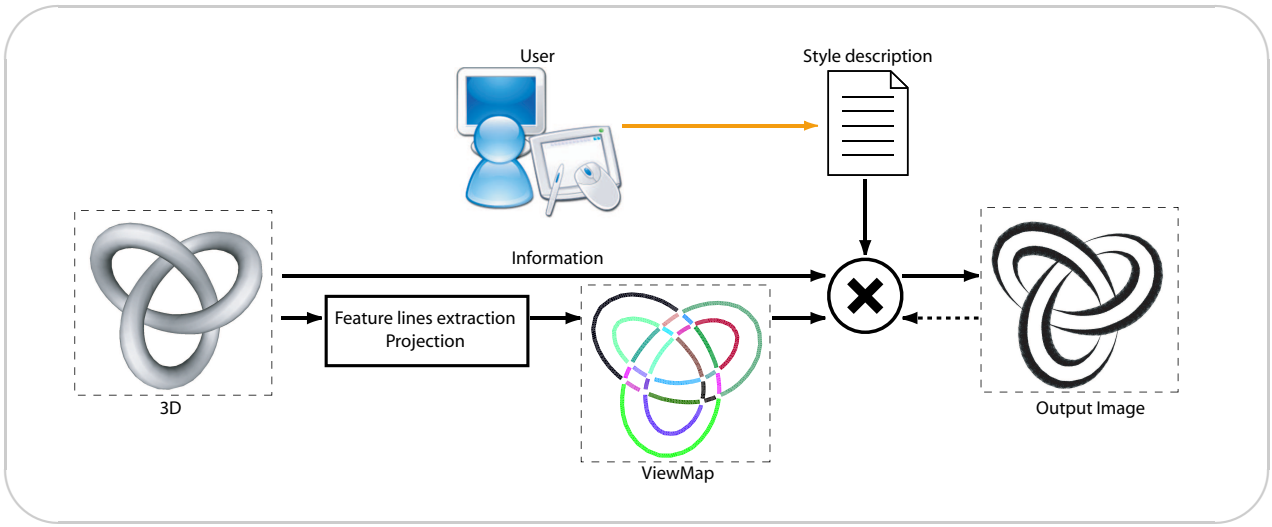


FIG. 3.26 – Le pipeline du système

rallèlement, l'ensemble des informations vues en section 3.3 est calculé et stocké au niveau du graphe de vue.

La feuille de style écrite par l'utilisateur est ensuite interprétée afin de construire un ensemble de traits (*strokes*) stylisés à partir des arêtes du graphe de vue. Cette feuille représente la spécification algorithmique du style construite à l'aide des opérateurs de description de style fournis. Dans cette description de style, l'utilisateur programme la manière dont les arêtes du graphe de vue doivent être transformées en traits stylisés. Quatre opérations peuvent être identifiées pour cette transformation. La première étape consiste à définir le sous-ensemble des arêtes destinées à être mises en style à l'aide de l'opérateur de *sélection*. Dans la deuxième étape, il s'agit de définir la topologie des futurs traits à partir, notamment, des informations d'adjacence du graphe de vue. Un opérateur de *chaînage* est fourni à cette fin. Enfin, la dernière opération consiste à assigner les attributs de style bas niveau aux traits ainsi créés à l'aide de l'opérateur de *shading*. Pour certaines des opérations, l'ordre dans lequel les éléments sont traités revêt une importance particulière. Dans le cadre de l'omission de lignes (pour simplifier le dessin) par exemple, cet ordre est essentiel dans la mesure où les arêtes les plus importantes doivent être gardées en priorité et donc dessinées en priorité. Un opérateur de *tri* permet donc de définir un ordre sur les éléments manipulés. Chacun de ces opérateurs est programmable et son action peut être paramétrée par l'ensemble des informations de la scène, ces dernières étant accessibles par le biais de fonctions d'accès. Le chapitre 5 présente la base des opérateurs proposée et l'accès à l'information est discuté en section 4.4, chapitre 4.

À partir de cette feuille de style, du graphe de vue et des informations, un ensemble de traits sont créés : ce sont ces traits qui seront finalement tracés dans le dessin, la stylisation étant contenue dans les attributs associés à chacun d'eux. La structure des traits est présentée en section 4.3, chapitre 4. Ces traits sont finalement rendus dans l'image finale.

Remarque : Le choix de construire des traits à partir des lignes caractéristiques détectées sur la scène 3D est un choix de design important. En effet, il implique que seules les lignes intialement présentes dans le graphe de vue pourront apparaître dans le dessin final. Ainsi, pour simplifier un dessin au trait, nous n'aurons que le choix d'omettre des traits ou de modifier leurs attributs visuels, et pas celui de remplacer un groupe de trait par un trait différent. Ce choix est dans notre cas incontournable dans la mesure où il garantit l'existence d'informations 3D pertinentes associées à chacune des lignes considérées. Du point de vue de la représentation picturale, cela signifie qu'on se place dans le contexte de 3D projetée en 2D plutôt que dans celui de construction 2D pour "ressembler" à la 3D. Cette dernière stratégie est moins contrainte et offre, notamment au niveau de la simplification de dessins au trait, des possibilités qu'il serait intéressant d'étudier dans l'avenir.

chapitre 4

Informations utiles aux décisions stylistiques : structuration et accès

Dans notre approche, une description de style consiste à spécifier comment construire et styliser des traits à partir d'un ensemble de lignes caractéristiques, détectées sur le modèle 3D. Dans ce chapitre, on s'interroge sur l'organisation et la structuration qui sont les plus adaptées à cet objectif. La structure de données centrale que nous adoptons, le *graphe de vue*, est le résultat d'un découpage et d'une structuration s'appuyant fortement sur les informations décrites à la section 3.3, qui semblent être utiles aux décisions stylistiques. Ce choix est motivé par le postulat que nous avons justifié au chapitre précédant, et suivant lequel ces informations jouent un rôle dans la modélisation du style. Après avoir défini ce graphe, nous expliquerons comment des accès à ces informations peuvent être aménagés depuis les différents éléments de ce dernier.

Bien qu'en pratique nous travaillions avec des maillages polyédriques, dans ce chapitre, sauf mention contraire, les concepts seront définis sur des surfaces exactes (par opposition, justement, à des surfaces "tesséllées" polyédriques) et ce, de manière à donner des définitions plus propres pour ces derniers.

> **Notation :** Si A est un ensemble, on notera $\overline{A}$ l'adhérence de A (le plus petit fermé tel que $A \subseteq \overline{A}$) et $\mathring{A}$, l'intérieur de A (le plus grand ouvert inclus dans A).

4.1 Les données d'entrée

Au chapitre précédent (section 3.5, nous avons présenté une vue d'ensemble du système dans laquelle la donnée d'entrée est un modèle 3D. Plus généralement, notre approche travaille sur un ensemble de *lignes caractéristiques* extraites du modèle 3D, ces dernières étant enrichies par des *informations* également extraites du modèle 3D. Dans cette section, nous définissons plus précisément ces lignes caractéristiques et ces informations.

4.1.1 Les lignes caractéristiques

Selon le schéma que nous avons choisi pour notre système, la première étape qui permet de passer d'un modèle 3D à un ensemble de traits (2D) est la détection de lignes caractéristiques

sur ce modèle. En effet, l'idée générale sous-jacente à cette étape est qu'il est possible de déterminer géométriquement, en 3D, les lieux sur une surface dont la projection dans l'image correspond aux traits généralement dessinés par les artistes pour représenter cette surface. Il est alors courant (et pratique) de considérer que ces lieux forment également des lignes dans l'espace, ce qui est vrai dans la plupart des cas (pour les lignes caractéristiques que nous avons choisies de considérer, à savoir les silhouettes, les crêtes et les vallées et les contours suggestifs). Ainsi, nous nous appuyons sur ces considérations et sur les techniques de détection de lignes caractéristiques (*cf.* section 2.2) pour détecter l'ensemble des points caractéristiques sur les surfaces de la scène et construire, à partir de cet ensemble de points, un ensemble de lignes 3D dont la projection constitue le support géométrique des futurs traits de l'image.

Nous détaillons à présent cette étape préliminaire de construction de lignes 3D à partir des points caractéristiques. Comme on peut le voir sur la figure 4.1, plusieurs constructions sont envisageables. Parmi ces possibilités, nous choisissons celle qui, à ce niveau, correspond à la granularité la plus fine et qui, intuitivement, consiste à construire des lignes les plus grandes possibles qui débutent et s'arrêtent sur des embranchements en 3D (figure 4.1 (d)). En l'absence d'embranchement, on obtient des boucles fermées. Cette construction est complètement menée en 3D.

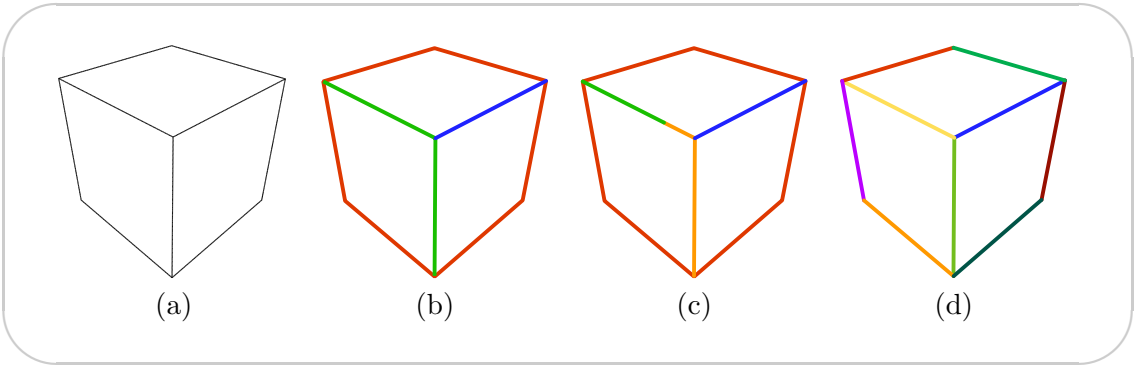


FIG. 4.1 – Lignes caractéristiques

À partir des points caractéristiques détectés sur ce cube (a), il existe une infinité de manières de construire des lignes caractéristiques. Nous choisissons la solution (d), qui consiste à construire les lignes maximales entre deux embranchements.

Cet ensemble de lignes peut être défini de manière plus mathématique de la façon suivante : si $\mathcal{P}$ désigne l'ensemble des points caractéristiques des surfaces de la scène, pour le point de vue considéré, alors, l'ensemble $\mathcal{C}$ des courbes caractéristiques C_i formées sur les surfaces de la scène à partir de ces points est défini par :

$$\mathcal{C} = \left\{ C_i \;\middle|\; \begin{array}{l} C_i \subseteq \mathcal{P} \\ C_i \textit{ ouvert et connexe} \\ \forall P \in C_i, \exists \epsilon / \mathcal{B}_\epsilon(P) \cap \mathcal{P} \text{ est une variété de dimension 1} \\ C_i \textit{ maximal} \end{array} \right\}$$

C'est à partir de cet ensemble $\mathcal{C}$ de lignes 3D que les traits seront construits. La figure 4.2 illustre ces courbes sur une scène simple.

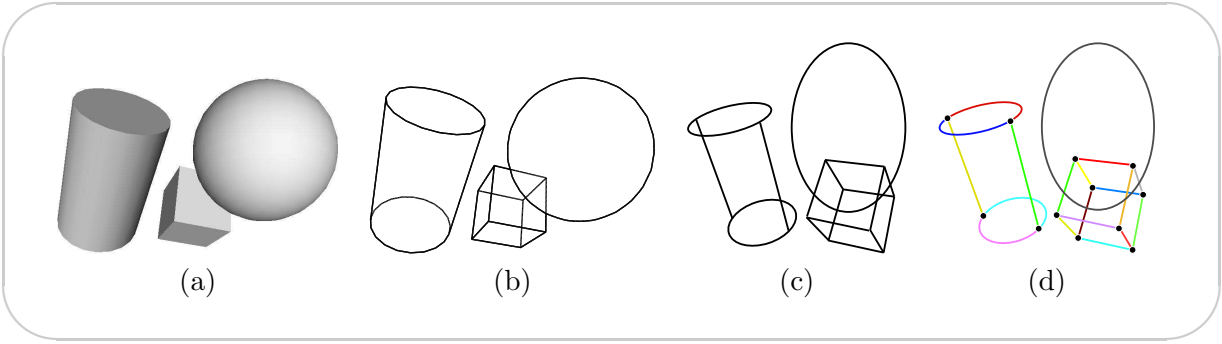


FIG. 4.2 – Graphe de vue : courbes caractéristiques

(a) : La scène 3D sous le point de vue considéré. (b) : Un ensemble $\mathcal{P}$ de points caractéristiques pour cette scène et ce point de vue. (c) : L'ensemble $\mathcal{P}$ observé sous un autre point de vue. (d) : Les sommets séparant les différentes courbes caractéristiques C_i.

Comme nous allons le voir dans la suite de ce chapitre, ces lignes peuvent (et vont) encore être subdivisées et arrangées au sein d'un graphe de manière à être le plus adapté possible à notre approche de construction de traits.

Remarque : En pratique, notre approche n'est pas dépendante des algorithmes utilisés pour détecter les points caractéristiques et permet, de manière naturelle, de facilement prendre en compte des familles supplémentaires.

4.1.2 Les informations

L'autre donnée d'entrée qui nous sert à définir les traits sont les informations : c'est sous cette désignation générale que nous regroupons l'ensemble des informations qui peuvent être extraites de la scène et de l'image finale et qui, selon nous, jouent un rôle dans les choix stylistiques d'un artiste. Elles sont définies sur les lignes caractéristiques, pour un point de vue donné. Ce sont ces informations qui ont été informellement introduites dans le chapitre précédent.

Nous reprenons ici la liste de ces informations, qui avait été donnée au chapitre 3, sous un point de vue plus formel :

Informations extraites de la scène

Coordonnées 3D Définies dans $\mathbb{R}^3$. Ce sont les coordonnées 3D des points se trouvant sur les lignes caractéristiques.

Coordonnées 2D Définies dans $\mathbb{R}^2$. Ce sont les coordonnées 2D des points dans l'image.

Profondeur Définies dans $\mathbb{R}$. À mi-chemin entre la 3D et la 2D, il s'agit de la coordonnée z d'un point, dans le repère de l'observateur, c'est-à-dire la distance de ce point au point de vue.

Normales 3D lisses Définies dans $\mathbb{R}^3$. On appelle *normale lisse*, l'unique normale 3D extérieure associée à chaque point d'une surface lisse.

Normales 3D vives Définies dans $\mathbb{R}^3$ x $\mathbb{R}^3$. On appelle *normales vives*, le couple de normales 3D associées à chaque point appartenant à une arête vive d'une surface (il y a, dans ce cas, exactement deux normales dans la mesure où les surfaces considérées sont des variétés de dimension 2).

> **Remarque :** Il est important de remarquer que cette information est destinée à être exploitée dans son ensemble. En particulier, il est impossible de déterminer à laquelle des faces adjacentes correspond chacune des normales dans la mesure où l'orientation des arêtes vives ne répond à aucune règle.

Normales 2D Définies dans $\mathbb{R}^2$. Il s'agit des normales 2D définies en chaque point des projections des lignes caractéristiques, dans l'image.

Courbures 2D Définies dans $\mathbb{R}$. Il s'agit de la courbure 2D définie en chaque point des projections des lignes caractéristiques, dans l'image.

Nature des lignes Définie dans $\{0,1\}^n$. L'ensemble des lignes caractéristiques est construit comme l'union de plusieurs familles de lignes vérifiant des propriétés différentes (*e.g.* silhouettes, contours suggestifs). La nature d'une ligne correspond à l'ensemble des familles auxquelles elle appartient : si on a n familles différentes, il s'agit d'un vecteur de dimension n dont chaque composante vaut 0 ou 1 selon qu'elle s'applique à une famille auquel la ligne appartient ou non (une même ligne peut appartenir à plusieurs familles différentes). En pratique, les natures considérées dans notre système sont les silhouettes, les contours suggestifs, les arêtes vives et les bords. Cette information est définie en chaque point des lignes caractéristiques.

Invisibilité quantitative Définie dans $\mathbb{N}$. Introduite par Appel *et al.* [App67a] (*cf.* section 2.2.3), l'invisibilité quantitative est définie en un point, comme le nombre de surfaces se trouvant entre ce point et l'observateur. Un point visible a une invisibilité quantitative de 0.

Objets occultant Définie dans $\{0,1\}^n$. Il s'agit d'un vecteur dont la composante k indique si la surface k appartient à la liste des surfaces occultantes (c'est-à-dire si la surface k se trouve entre l'observateur et le point auquel l'information est évaluée en suivant la direction de vue).

Discontinuité en profondeur Définie dans $\mathbb{R}$. La discontinuité en profondeur en un point d'une des lignes caractéristiques est définie comme la distance entre ce point et le plus proche objet en suivant la direction de vue (*cf.* figure 3.21 page 74). Cette information n'a de sens que pour les points sur les lignes de silhouette. En pratique, on la définit comme nulle sur l'ensemble des points des lignes qui ne sont pas des silhouettes.

Objets cachés Définie dans $\{0,1\}^n$. Il s'agit d'un vecteur dont la composante k indique si la surface k appartient à la liste des surfaces cachées (c'est-à-dire si le point auquel est évaluée l'information se trouve entre l'observateur et la surface k en suivant la direction de vue).

Couleur du matériau Dans le cadre de cette thèse, nous considérons que les matériaux sont définis par objet, ce qui signifie, que pour un point donné d'une surface, on n'a qu'un seul matériau possible. Cette information est alors définie dans $\mathbb{R}^3$ (couleur diffuse) $\times$ $\mathbb{R}^3$ (couleur spéculaire)$\times$ $\mathbb{R}^3$ (couleur ambiante) $\times$ $\mathbb{R}^3$ (couleur émissive) $\times$ $\mathbb{R}$ (coefficient de brillance).

> **Remarque :** En réalité, dans le cas général, le type de maillages avec lequel on travaille, permet de définir un matériau par face. Il aurait donc fallu prévoir de stocker, non pas un, mais deux matériaux par point, dans certains cas. En pratique, dans la mesure où nous ne détectons pas, sur les surfaces, les lignes correspondant à des transitions de matériaux, il nous a semblé plus cohérent de ne travailler qu'avec des matériaux définis par objet. L'inclusion de telles lignes et la gestion de plusieurs matériaux par objet font partie des travaux futurs.

Identité des objets Définie dans $\mathbb{N}$. Il s'agit de l'identifiant de l'objet auquel le point considéré appartient. Nous ne gérons pas les intersections entre objets.

Arrangement des lignes dans le plan Cette information est différente des autres dans la mesure où elle met en relation les lignes entre elles. Elle indique, pour une ligne donnée, quelles lignes lui sont "connectées", c'est-à-dire quelles lignes l'intersectent, en 3D (sur la surface), ou en 2D, dans le plan image (ce sont alors les projections des lignes qui s'intersectent). En pratique, comme nous allons le voir, les lignes vont être arrangées dans un graphe planaire, qui contiendra intrinsèquement cette information de connectivité.

Informations extraites du dessin

Densité Définie dans $\mathbb{R}$. Cette information est étudiée plus en détails dans le chapitre 6.

Longueur totale des traits Définie dans $\mathbb{R}$. Cette information étant mesurée dans l'image finale (et non plus sur l'ensemble des lignes caractéristiques initiales), il ne s'agit plus de lignes, mais de traits (les traits seront présentés plus en détail section 4.3). Cette information correspond donc à la longueur d'un trait.

Abscisse curviligne le long d'un trait Définie dans $\mathbb{R}$.

Pour définir un style (topologie et attributs visuels des traits), l'utilisateur pourra accéder à l'ensemble de ces informations depuis les points sur les lignes caractéristiques. Par exemple, pour définir un style dans lequel l'épaisseur d'un trait en chacun de ses points dépend de la distance à l'observateur, l'utilisateur pourra faire des requêtes sur l'information de profondeur. En pratique, comme nous le verrons après, nous allons considérer la *fonction d'information* qui, à un point d'une courbe C_i, associe l'ensemble des informations qui lui sont associées, et nous appuyer sur son domaine de continuité pour subdiviser les lignes caractéristiques. Les informations que nous avons vues sont, d'une manière générale, définies comme des scalaires de $\mathbb{R}$ ou $\mathbb{N}$, ou des vecteurs de $\mathbb{R}^n$ ou $\mathbb{N}^n$. Dans la mesure où, d'une part, l'étude de continuité d'une quantité vectorielle se fait composante par composante, et, d'autre part, $\mathbb{N} \subset \mathbb{R}$, nous pouvons donc mener les études de continuité dans $\mathbb{R}$.

Dans les définitions qui suivent, les formulations sont faites dans le cas général (c'est-à-dire pour un ensemble d'informations quelconques), et on notera $I = I_0 \times \ldots \times I_m$ l'espace défini par les m différents types d'informations considérés, la $k^{ième}$ information étant définie dans l'espace I_k.

Avec cette notation, la fonction d'information définie plus haut s'écrit :

$$\begin{array}{rccc} \mathcal{I} : & \bigcup C_i & \longrightarrow & I \\ & P & \longmapsto & (\lambda_0, \ldots, \lambda_m) \end{array}$$

Résumé: À partir du modèle 3D considéré en entrée, on détecte un ensemble de points caractéristiques (qui correspondent aux silhouettes, *etc.*...) et on construit un ensemble de courbes 3D C_i à partir de ces points. En outre, l'ensemble des informations présentées au chapitre 3 sont calculées et on définit la fonction d'information $\mathcal{I}$ pour y accéder depuis les C_i.

4.2 Structuration des lignes et des informations : le graphe de vue

Dans cette section, nous motivons et présentons le raffinement des lignes caractéristiques et leur organisation au sein d'un graphe (le *graphe de vue*).

4.2.1 Pourquoi et comment structurer les lignes caractéristiques ?

■ ***Nous choisissons un raffinement et une organisation des lignes caractéristiques qui s'appuient sur les informations considérées pour la description de style.***

Comme nous l'avons vu plus haut, ce sont les lignes caractéristiques 3D qui constituent le point de départ du processus de création des traits de l'image finale. La structuration de l'ensemble des points caractéristiques en cet ensemble de lignes s'appuie uniquement sur des considérations géométriques 3D et on peut se demander s'il n'existe pas une organisation qui soit plus adapté à notre objectif.

Dans notre approche, la définition des traits, en termes de topologie et d'attributs visuels, s'appuie d'une part sur la manière dont les projections des lignes caractéristiques sont arrangées dans l'image, et, d'autre part, sur un sous-ensemble des informations définies à la section 4.1.2. Il semble donc pertinent de s'appuyer davantage sur ces informations pour définir une subdivision des lignes caractéristiques. Par exemple, si on considère l'information d'invisibilité quantitative et que l'on suppose que le style désiré par l'utilisateur consiste à créer des traits noirs pour les segments de lignes visibles, et gris pour les segments de lignes cachés : avec le découpage actuel, une même ligne caractéristique peut à la fois comporter des parties visibles et des parties invisibles ; intuitivement, il semble plus approprié de disposer d'un découpage plus fin des lignes, prenant en compte ce type d'informations, de manière à pouvoir facilement sélectionner les lignes correspondant à un ensemble donné de critères.

■ ***On adopte le découpage induit par la continuité par morceaux de la fonction d'information et par l'arrangement planaire des lignes***

En pratique (*cf.* section 4.2.3), la fonction d'information $\mathcal{I}$, qui à tout point d'une des lignes C_i associe le vecteur d'informations associé, est continue par morceaux sur l'ensemble des C_i et nous avons choisi de respecter le découpage induit par son domaine de définition pour partitionner l'ensemble constitué par les points caractéristiques : on divise le support créé par l'ensemble des points caractéristiques en courbes sur lesquelles la fonction d'information $\mathcal{I}$ est continue. Nous avons ainsi le découpage qui est à la fois la plus compacte et la plus flexible vis-à-vis de la continuité $\mathcal{C}^0$ des informations.

En outre, comme nous le verrons dans le prochain chapitre, les traits (qui sont les éléments dessinés dans l'image finale) sont définis comme un chemin dans le graphe planaire des lignes, il faut donc également subdiviser les lignes aux endroits où elles s'intersectent entre elles. Avec la combinaison de ces deux découpages, on obtient un ensemble de lignes sur lesquelles la fonction d'information est continue et qui reflète l'arrangement 2D des projections de ces lignes.

Finalement, une fois un partitionnement efficace en arêtes déterminé, il reste encore à décider de quelle structure utiliser pour parcourir ces arêtes.

On structure ces courbes sous forme d'un graphe

Nous avons mentionné dans la section 3.2.2 que la topologie des traits était une composante essentielle du trait. En particulier, il n'y a pas de bijection entre l'ensemble des courbes précédemment défini et l'ensemble des traits : un trait peut être supporté par plusieurs ligne et une ligne peut être dessinée à l'aide de plusieurs traits. Afin de définir la topologie des traits, il est donc nécessaire de disposer d'une structure de connectivité permettant le parcours des courbes connectées entre elles (*e.g.* une structure de graphe).

Nous avons par ailleurs vu dans cette même section 3.2.2, que la définition de la topologie des traits s'appuyait non seulement sur celle des objets représentés, mais plus généralement sur l'*aspect* [PD90b] de la scène sous le point de vue considéré. Notre structure de graphe devra donc également refléter la projection planaire de la scène, en particulier en connectant à un même sommet les courbes disjointes dans l'espace mais s'intersectant dans le plan image.

C'est au travers des sommets et des arêtes de ce graphe (le *graphe de vue*) que l'on accèdera à l'ensemble des informations défini à la section 4.1.2.

Remarque : La définition du graphe de vue s'appuie sur les informations extraites de la scène mais pas sur celles extraites du dessin (*cf.* section 4.1.2). En effet, c'est à partir des arêtes du graphe de vue que le dessin est formé, on ne peut donc pas les faire elles-mêmes dépendre de ce dernier.

Remarque : En pratique, certaines des informations sont coûteuses à calculer et pour cette raison ne sont pas utilisées dans la définition du découpage du graphe de vue (la densité par exemple). Toutefois, comme nous le verrons dans le chapitre suivant, il existe parmi les opérateurs utilisables au sein de la description de style un opérateur de *découpage* qui permet de raffiner davantage la structure (en fonction de combinaisons d'informations par exemple) au moment de la construction des traits. Les informations les plus complexes sont ainsi calculées à la demande dans la spécification de style.

Nous définissons maintenant les arêtes et les sommets du graphe de vue dans le cas général avant d'en faire une caractérisation plus concrète s'appuyant sur les informations considérées. Cette caractérisation constitue également un guide pratique de construction du graphe.

4.2.2 Définition du graphe de vue dans le cas général

Bien qu'une description intuitive de la structure du graphe de vue puisse sembler suffisante, il est en réalité indispensable d'en donner une définition mathématique permettant de définir précisément les objets manipulés et d'éviter ainsi d'éventuelles ambiguïtés.

Raffinement des lignes caractéristiques

L'étude du découpage des lignes caractéristiques s'articule autour de deux contraintes bien distinctes. La première consiste à subdiviser les lignes de manière à assurer la continuité de la fonction d'information, tandis que la deuxième intègre les intersections entre les projections de ces lignes dans le plan image, de façon à pouvoir prendre en compte chaque embranchement en 2D.

Découpage induit par la fonction d'information Comme nous l'avons expliqué précédemment, pour cette contrainte, nous nous appuyons sur les discontinuités de la fonction d'information, c'est-à-dire de $\mathcal{I}$, pour assurer un découpage pratique (dans le contexte de la construction et de la stylisation de traits) des lignes caractéristiques.

On définit donc un ensemble de courbes $\mathcal{E}_{cont}$, tel que chaque élément provient du découpage des C_i en morceaux sur lesquels $\mathcal{I}$ est continue :

$$\mathcal{E}_{cont} = \left\{ E_i \;\middle|\; \begin{array}{l} E_i \text{ ouvert et connexe} \\ \exists C_i \;/\; E_i \subseteq C_i \; et \; \mathcal{I}_{|E_i} \; continue \\ E_i \; maximal \end{array} \right\}$$

On appelle $\mathcal{V}_{cont}$ l'ensemble des points de $\mathcal{P}$ qui sont des extrémités des C_i et points de discontinuité pour $\mathcal{I}$:

$$\mathcal{V}_{cont} = \mathcal{P} \backslash (\bigcup_{E_i \in \mathcal{E}_{cont}} E_i)$$

Si chaque fonction d'information $\mathcal{I}_k$, définie par :

$$\begin{array}{rccc} \mathcal{I}_k : & \bigcup C_i & \longrightarrow & I_k \\ & P & \longmapsto & \lambda_k \end{array}$$

est continue par morceaux, alors $\mathcal{I}$ l'est aussi et (par définition) $\mathcal{V}_{cont}$ est un ensemble discret.

Découpage induit par l'arrangement planaire des lignes Pour respecter la deuxième contrainte, les lignes C_i dont les projections s'intersectent dans le plan doivent être subdivisées : si C_i et C_j sont telles que leurs projections L_i et L_j s'intersectent en P, alors C_i est découpée au point Q_i, qui correspond à la projection inverse de P sur C_i, et, de même, C_j est découpée au point Q_j, qui correspond à la projection inverse de P sur C_j.

On appelle $\mathcal{E}_{inter}$ l'ensemble des courbes issues de cette subdivision. Si $\mathbb{P}$ est la projection de l'espace 3D vers l'image, alors, on peut écrire $\mathcal{E}_{inter}$ ainsi :

$$\mathcal{E}_{inter} = \left\{ E_i \;\middle|\; \begin{array}{l} E_i \text{ ouvert et connexe} \\ \exists C_i \;/\; E_i \subseteq C_i \; et \; \forall j \neq i, \mathbb{P}(E_i) \cap \mathbb{P}(C_j) \\ E_i \; maximal \end{array} \right\}$$

On notera $\mathcal{V}_{inter}$, l'ensemble des points de $\mathcal{P}$ qui se projettent sur des intersections :

$$\mathcal{V}_{inter} = \mathcal{P} \backslash (\bigcup_{E_i \in \mathcal{E}_{inter}} E_i)$$

Découpage issu de la combinaison des deux contraintes Finalement, les lignes caractéristiques sont découpées de manière à intégrer l'ensemble des points définis par chacun des deux découpages présentés juste au-dessus. Ainsi, l'ensemble $\mathcal{E}$ des lignes issues de ce découpage s'écrit ainsi :

$$\mathcal{E} = \{E_i \ \in \ \mathcal{E}_{cont} \cup \mathcal{E}_{inter} \mid \nexists \ E_j \neq E_i \ / \ E_j \subset E_i\}$$

Et l'ensemble $\mathcal{V}$ des points qui correspondent, soit à des extrémités des C_i, soit à des points de discontinuité pour $\mathcal{I}$, soit à des points d'intersection des projections des C_i dans le plan s'écrit :

$$\mathcal{V} = \mathcal{V}_{cont} \cup \mathcal{V}_{inter} = \mathcal{P} \backslash (\bigcup_{E_i \in \mathcal{E}} E_i)$$

En pratique, comme on le verra, il y a un important recouvrement entre $\mathcal{V}_{cont}$ et $\mathcal{V}_{inter}$, c'est-à-dire que la plupart des points de discontinuité de la fonction d'information correspondent à des intersections des projections des lignes dans le plan.

Organisation en graphe

Il s'agit à présent d'organiser les nouvelles lignes 3D définies par le découpage précédent en graphe. Comme nous l'avons écrit plus haut, ce graphe doit, en plus des informations de connexité existant entre les lignes de $\mathcal{C}$ en 3D, refléter l'arrangement 2D des projections de ces lignes. Pour ce faire, on construit le graphe planaire de $\mathcal{C}$.

On commence donc par organiser ces courbes en un graphe $\widetilde{\mathcal{G}}$ dont $\mathcal{E}$ et $\mathcal{V}$ définissent respectivement les arêtes et les sommets. L'ensemble des arêtes de $\widetilde{\mathcal{G}}$ est défini comme :

$$E(\widetilde{\mathcal{G}}) = \left\{(V_i, V_j) \in \mathcal{V}^2 \mid \exists E_k \in \mathcal{E} / \{V_i, V_j\} \subseteq \overline{E_k}\right\}$$

Ainsi construit, ce graphe respecte la topologie des objets, c'est-à-dire que les arêtes connectées à un même sommet se rencontrent géométriquement sur la surface. Or on a également vu, qu'en plus de la topologie des surfaces, la construction des traits pouvait s'appuyer sur l'aspect de la vue : en particulier, il est possible que les lignes caractéristiques de deux objets différents dont les projections dans le plan se rencontrent soient réunies au sein d'un même trait (*cf.* section 3.2.2). Donc, si V_a et V_b sont deux points de $\mathcal{V}$ dont les coordonnées 2D sont les mêmes, on veut pouvoir réunir deux arêtes E_i et E_j de $\mathcal{E}$ respectivement connectées à V_a et V_b.

Le graphe de vue $\mathcal{G}$ est obtenu à partir du graphe $\widetilde{\mathcal{G}}$ en fusionnant les sommets dont les projections sont confondues dans le plan. Si $\mathbb{P}$ est la projection de l'espace 3D dans l'image, alors les sommets de $\mathcal{G}$ sont les sommets de $\widetilde{\mathcal{G}}$ modulo $\mathbb{P}$:

$$V(\mathcal{G}) = V(\widetilde{\mathcal{G}})_{/\mathbb{P}} = \left\{ \left\{ V' \in V(\widetilde{\mathcal{G}}) \mid \mathbb{P}(V') = \mathbb{P}(V) \right\} \mid V \in V(\widetilde{\mathcal{G}}) \right\}$$

Remarque : Dans ce travail, on considèrera toujours, qu'à l'exception des sommets issus d'intersections en T, les projections des sommets et des lignes dans le plan ne sont jamais exactement colinéaires. Ainsi, seuls les sommets correspondant à des intersections en T feront l'objet de fusions. La gestion des configurations singulières constitue un projet de recherche intéressant mais qui sort du cadre de cette thèse. En pratique, nous n'avons pas constaté de problèmes liés à de telles configurations.

Remarque : En informatique graphique, le terme "arête" désigne souvent un segment rectiligne 3D (*e.g.* arêtes d'un maillage). En ce qui nous concerne, nous employons ce terme pour désigner une arête du graphe de vue, c'est-à-dire une courbe (a priori non rectiligne) de $\mathcal{E}$.

4.2.3 Caractérisation des sommets du graphe

Outre les sommets correspondant à des intersections en T (qui sont faciles à identifier), il est important d'être capable de caractériser les sommets correspondant à des discontinuités de la fonction d'information. Après avoir présenté ces derniers dans le cas général, nous les étudions ici plus concrètement, au regard des différentes informations considérées.

Étude de continuité des fonctions d'information

Nous proposons ici une étude informelle des propriétés de continuité des fonctions d'informations $\mathcal{I}_k$ (*cf.* section 4.2.2). Chaque discontinuité d'une de ces fonctions d'information implique la présence d'un sommet du graphe de vue.

Nature des arêtes du graphe de vue ($\mathbb{N}$) Nous avions vu à la section 4.1.2 que l'information de nature des arêtes du graphe de vue était un vecteur de dimension 4 (pour les quatre natures possibles : silhouette, arête vive, contour suggestif, bord) dont chaque composante vaut 0 ou 1, selon que l'arête appartient à la famille concernée ou non. Un changement de ce vecteur de nature sur l'une de ses composantes suffit donc à créer une discontinuité et donc un sommet du graphe de vue. La figure 4.3 montre par exemple une courbe qui passe de l'état (arête vive) à l'état (silhouette, arête vive).

Par définition, seules les silhouettes et les arêtes vives peuvent cohabiter sur une même ligne caractéristique C_i. Dans ce cas, la seule configuration possible, pouvant engendrer une discontinuité de l'information de nature, est qu'une ligne soit uniquement arête vive sur une partie, et à la fois arête vive et silhouette sur l'autre partie. Cette configuration correspond à une ligne de crête qui, à partir d'un point donné, intègre la silhouette. En pratique, une arête vive peut devenir une silhouette si et seulement si elle intersecte, sur la surface, une autre ligne de silhouette. Dans la mesure où ce type de croisements a déjà donné lieu à des sommets (parmi les extrémités des C_i), aucun sommet n'est créé par cette information.

Invisibilité quantitative ($\mathbb{N}$) C'est l'information qui est à l'origine du plus grand nombre de discontinuités de $\mathcal{I}$. On considère une courbe C_i de $\mathcal{C}$ et sa projection L_i dans le plan. Trois types d'événements peuvent provoquer un changement d'invisibilité quantitative sur L_i et donc l'introduction d'un sommet dans le graphe de vue :

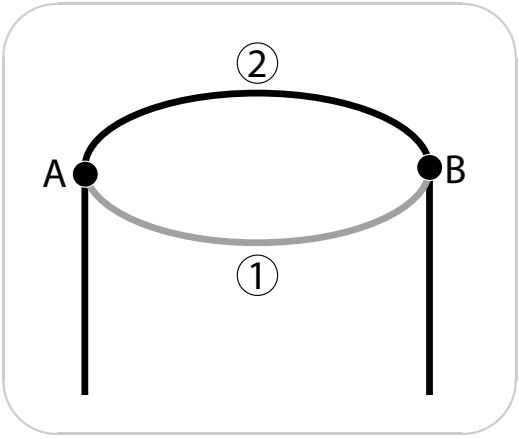


FIG. 4.3 – Sommets du graphe de vue : discontinuité de nature

Sur cette illustration l'arête marquant le couvercle du cylindre est uniquement arête vive sur la partie (1) (tracée en gris clair) et à la fois arête vive et silhouette sur la partie (2). On a donc inséré 2 sommets pour le graphe de vue aux points A et B marquant la discontinuité.

- L'intersection (dans le plan) de L_i avec la projection d'une ligne de silhouette (ou de bord), celle-ci se trouvant plus proche du point de vue que C_i. On crée alors un sommet sur C_i en re-projetant le point d'intersection trouvé dans le plan image sur C_i. Le sommet créé sur C_i correspond alors à une jonction en T (*T-Vertex*). Les jonctions en T sont illustrées par la figure 2.18 page 37 (a). Par définition, ces points correspondent à des intersections dans le plan image et sont inclus dans l'ensemble des sommets issus de l'information d'arrangement planaire des lignes.
- L'intersection (en 3D, sur la surface) de C_i avec une ligne de silhouette. Le sommet créé correspond alors à une jonction en Y. Les points correspondant à des jonctions en Y peuvent également (mais pas nécessairement) correspondre à un changement de nature de C_i. Les sommets A et B de la figure 4.3 en sont une illustration. Les points correspondant à des jonctions en Y sont déjà des extrémités des courbes de $\mathcal{C}$ et ont déjà, à ce titre, donné lieu à la création de sommets.
- La présence d'un point de rebroussement (*cusp* sur C_i si C_i est une silhouette. Les points de rebroussement doivent donc être détectés (plusieurs techniques existent pour ce faire, *cf.* section 2.2.3) et donner lieu à la création de sommets du graphe de vue. La figure 2.18 illustre ces points.

Objets occultants ($\mathbb{N}^n$) L'information donnant la liste des objets occultants, change sur une courbe C_i lorsque sa projection L_i intersecte la projection d'une ligne de silhouette (ou de bord) qui est également un contour et qui est plus proche du point de vue que C_i. Ces sommets font également partie des sommets issus des intersections de lignes dans le plan.

Surfaces cachées ($\mathbb{N}^n$) On observe un changement dans la liste des surfaces cachées par une courbe C_i si et seulement si sa projection L_i dans le plan image intersecte la projection d'une ligne de silhouette (ou de bord) qui est également un contour et qui se trouve plus loin du point de vue que C_i. Le sommet correspondant est également une intersection en T, et

est donc inclus dans l'ensemble des sommets issus des intersections de lignes dans le plan. La figure 4.4 illustre ces points de discontinuités.

Discontinuité en profondeur ($\mathbb{R}$) Cette information présente une discontinuité le long d'une ligne de silhouette (ou de bord) C_i si et seulement si sa projection L_i intersecte la projection d'une autre ligne de silhouette (ou de bord), cette dernière se trouvant directement derrière C_i en suivant le rayon de vue. La figure 4.4 (c) montre des sommets créés en des points de discontinuité de l'information de discontinuité en profondeur.

Ces sommets sont des jonctions en T et, à ce titre, sont inclus dans la liste des sommets issus du calcul des intersections de lignes dans le plan.

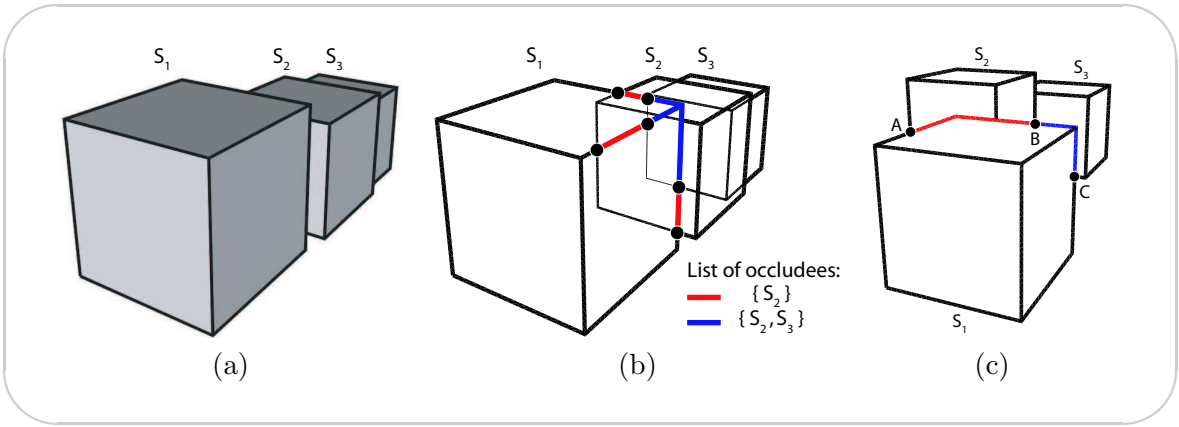


FIG. 4.4 – Sommets du graphe de vue : surfaces cachées et discontinuité en profondeur

(a) scène 3D composée de trois objets S_1, S_2 et S_3. (b) On trace les lignes caractéristiques visibles ou cachées par deux ou quatre surfaces (les objets étant fermés, seules les lignes dont l'invisibilité quantitative est paire ont une influence sur la liste des surfaces cachées). Pour S_1, la couleur des lignes dépend de la liste des surfaces cachées : noir pour les lignes ne cachant aucune Les lignes visibles sont dessinées avec le poids le plus important, les lignes cachées par deux surfaces sont dessinées avec un poids moyen et les lignes cachées par quatre surfaces sont dessinées avec un poids faible. (c) : On a une discontinuité en profondeur le long d'une lignes de silhouette (ou de bord) C_i lorsque la projection de cette dernière intersecte la projection d'une ligne de silhouette appartenant à une surface située directement derrière C_i. En B par exemple, l'objet caché passe de S_2 à S_3 provoquant ainsi un changement abrupte de la valeur de discontinuité en profondeur.

Normales 3D lisses ($\mathbb{R}^3$) ou vives ($\mathbb{R}^3$ x $\mathbb{R}^3$) On a une discontinuité de l'information de normales 3D le long d'une ligne C_i si et seulement si celle-ci intersecte, sur la surface 3D, une arête vive. Les sommets correspondant à de telles intersections font partie des sommets initiaux (extrémités des C_i) et n'ont donc pas besoin d'être ajoutés.

Courbures 2D ($\mathbb{R}$) Pour un objet lisse, les seules discontinuités de la courbure 2D le long de la projection d'une ligne C_i sont observables aux points de rebroussement (*cusps*). Ces sommets sont donc inclus dans ceux créés par les discontinuités de l'information d'invisibilité quantitative.

Par ailleurs, pour un objet comportant des arêtes vives, on aura potentiellement des discontinuités au points présentant des discontinuités des normales 3D (mais pas nécessairement).

Normales 2D ($\mathbb{R}^2$) Les points de discontinuités des normales 2D sont les mêmes que ceux de la courbure 2D.

Coordonnées 3D ($\mathbb{R}^3$) Les lignes de $\mathcal{C}$ formant des courbes (continues) sur les surfaces des objets de la scène, il n'y a aucune discontinuité du vecteur de coordonnées 3D le long des courbes de $\mathcal{C}$.

Profondeur ($\mathbb{R}$) Pour la même raison qu'au-dessus, il n'y a pas de discontinuité de l'information de profondeur le long des lignes de $\mathcal{C}$.

Coordonnées 2D ($\mathbb{R}^2$) Les projections des courbes de $\mathcal{C}$ étant elles-mêmes des courbes dans le plan image, il n'y a aucune discontinuité du vecteur de coordonnées 2D le long des lignes de $\mathcal{C}$.

Identité des objets ($\mathbb{N}$) Ici encore, puisque les lignes de $\mathcal{C}$ restent sur les surfaces (ne passent pas d'une surface à l'autre), il ne peut pas y avoir de changement d'objet (et donc d'identité d'objet) le long d'une ligne.

Matériau ($\mathbb{R}^k$) En pratique, bien que les composantes des informations de matériau soient définies dans $\mathbb{R}$, ces dernières prennent généralement des valeurs constantes le long d'une ligne (le cas le plus courant est de rencontrer un nombre fini de matériaux dans toute la scène).

Nous avons supposé que les matériaux étaient définis par objet. Dans ce cas, de même que pour l'information d'identité des objets, on ne peut pas avoir de discontinuité de l'information de matériau le long d'une ligne.

Construction pratique du graphe de vue

Nous résumons ici les informations concernant la création de sommets du graphe que nous venons de voir, de manière à offrir une méthode pratique de construction du graphe de vue.

La première étape consiste à détecter un ensemble de points caractéristiques sur les surfaces de la scène. Nous utilisons la technique de Hertzman et Zorin [HZ00] pour détecter les lignes de silhouette, les arêtes vives correspondent aux points auxquels sont associés plus qu'une normale, les contours suggestifs sont calculés selon la méthode de DeCarlo *et al.* [DFRS03] et finalement, les bords sont détectés comme les arêtes du maillage qui sont bordées par une unique face. En pratique, les surfaces avec lesquelles nous travaillons sont des maillages polygonaux, les techniques de détection de points caractéristiques nous permettent de construire un ensemble de segments caractéristiques sur ces surfaces (des arêtes du maillages pour les arêtes vives et les bords, des segments traversant les faces pour les silhouettes).

Il est nécessaire de disposer, pour le maillage initial, d'une structure contenant les informations d'adjacences (*e.g.winged-edge* [Gla91]).

Il s'agit ensuite de construire les courbes de $\mathcal{C}$ à partir de cet ensemble, c'est-à-dire les plus grandes parties connexes topologiquement équivalente à une ligne, qui sont la base de départ

du graphe de vue. La structure d'adjacence du maillage nous permet également de facilement mener à bien cette étape. La première opération consiste à marquer, d'une manière ou d'une autre, les faces par lesquelles passent une ou plusieurs lignes caractéristiques. Pour chacune de ces faces, on a donc un ou plusieurs segments (ces derniers pouvant être des arêtes du maillage), faisant partie de l'une ou l'autre des lignes caractéristiques. Dans l'étape suivante, on construit des chaînes à partir de ces segments isolées : on parcourt les faces pour repérer ceux qui contiennent plus d'un segment caractéristique (et calculons éventuellement le sommet correspondant à l'intersection, si elle existe, de ces segments caractéristiques). Les lignes C_i débuteront sur ces faces. Grâce à la structure d'adjacence, on peut ensuite examiner les faces voisines afin de détecter celles qui contiennent des lignes caractéristiques (de même nature que notre segment initiale) et parcourir les faces de proche en proche jusqu'à arriver à une autre extrémité.

Une fois les C_i disponibles, la construction du graphe de vue peut commencer. Cette construction se fait en partant des lignes caractéristiques : les C_i sont les lignes initiales du graphe de vue et leurs extrémités, les sommets initiaux. Nous remarquons, au travers de la caractérisation faite précédemment, que, dans leur grande majorité, les points de discontinuité de la fonction d'information correspondent à des intersections de lignes dans le plan. La seule exception à cette règle sont les points de rebroussement, qui correspondent à des discontinuités des informations d'invisibilité quantitative, de courbure 2D et de normale 2D.

Pour construire le graphe de vue, à partir des lignes, il suffit donc de détecter les points de rebroussement (des techniques de détection de ces points en 3D sont citées dans le chapitre 2, section 2.2.3) et de créer des sommets aux points obtenus, puis de calculer les intersections, dans le plan, entre les projections des lignes résultantes, et de créer des sommets en "projetant inverse" ces intersections sur les lignes 3D. La fonction d'information sera ainsi continue sur l'ensemble des lignes construites. On calcule alors l'ensemble des informations, en traitant les lignes les unes après les autres (ces calculs sont détaillés dans la section 4.5). À chaque création de sommet, les informations de connectivité entre lignes sont stockées, de manière à créer la structure de graphe.

Remarque : Le nombre de sommets du graphe de vue ainsi créé peut être relativement élevé dans le cas d'objets complexes. Or, parmi les sommets issus des calculs d'intersections des lignes dans le plan, certains ne correspondent à aucune discontinuité de la fonction d'information, et sont associés à une intersection rarement exploitable. Par exemple, une intersection dans l'image entre les projections de deux arêtes vives (qui ne s'intersectent par ailleurs pas en 3D) constitue un embranchement auquel il est rare qu'un trait bifurque. En pratique, nous avons expérimenté notre approche avec une version du graphe de vue dans laquelle les intersections dans l'image, entre les projections de deux lignes, dont aucune n'est une silhouette ou un bord, ne créent pas sommets.

Résumé: Les lignes caractéristiques détectées en 3D sont découpées et arrangées dans un graphe planaire. Ce découpage et cet arrangement ont été définis en prévision des opérations au sein desquelles les lignes seront manipulées dans le cadre de la description de style. En particulier, dans la mesure où cette dernière s'appuie fortement sur les informations, celles-ci jouent un rôle prépondérant dans la définition de ce découpage.

4.3 Les traits

Les traits sont définis comme des chemins dans le graphe de vue

Outre les arêtes du graphe de vue, l'autre type d'élément 1D que l'utilisateur est amené à manipuler, dans le cadre d'une description de style, sont les *traits*. Les traits sont les objets qui seront finalement dessinés ; la spécification de leur topologie et de leurs attributs visuels, à partir du graphe de vue et des informations, sont l'objet d'une description de style. Comme nous le verrons dans le prochain chapitre, grâce aux opérateurs de description de style, l'utilisateur peut spécifier la topologie qu'il désire pour ces traits : celle-ci consiste en un chemin, parcourant, dans le graphe de vue, un ensemble connexe d'arêtes. Dans ce chapitre, on suppose que les traits ont été construits selon les désirs de l'utilisateur, et nous nous focalisons sur leur utilisation en tant de portail d'accès aux informations.

Les sommets des traits peuvent inclure des sommets du graphe de vue

Un trait est donc défini comme un ensemble 1D connexe (dans le plan) de points appartenant au graphe de vue. À la différence des arêtes du graphe de vue, un trait peut contenir un ou plusieurs sommet du graphe de vue. Une autre différence entre les arêtes du graphe de vue et les traits, est que ces derniers sont enrichis d'attributs : à chaque point est associé un vecteur contenant des informations visuelles à propos de ce point. En pratique, les attributs standard sont l'épaisseur (à droite et à gauche), la couleur, la transparence et la texture. L'utilisateur a la possibilité de rajouter n'importe quel attribut de type $\mathbb{R}$, $\mathbb{R}^2$, $\mathbb{R}^3$ ou $\mathbb{N}$. Ainsi, il est possible de tenir compte d'attributs de style spécialisés supplémentaires en branchant le moteur de rendu adéquat en bout de chaîne.

Si A désigne l'espace des attributs, un trait peut donc être écrit comme une application :

$$\begin{array}{rccc} T: & [0,1] & \longrightarrow & (\mathcal{E} \cup \mathcal{V}) \times A \\ & t & \longmapsto & (P, a) \end{array}$$

T étant telle que la projection $\mathbb{P}(Img(T))$ de ses points dans le plan image soit connexe.

La figure 4.5 illustre un trait construit à partir d'un graphe de vue. Nous verrons dans le prochain chapitre quels opérateurs permettent de passer des arêtes du graphe de vue aux traits.

> **Remarque :** Un trait ne correspond pas forcément à une ligne 3D, c'est-à-dire à un ensemble connexe de points 3D. En effet, un trait peut, par exemple, être issu d'un chaînage impliquant deux arêtes appartenant à des objets différents, mais dont les projections dans le plan image sont connexes. La figure 4.6 illustre ce cas de figure.

Il est important de remarquer que la construction d'un trait constitue un passage d'une structure de graphe à une structure 1-dimensionnelle, donc plus contrainte spatialement. Comme nous le verrons à la prochaine section, cet aspect nous aidera à lever un certain nombre d'ambiguïtés présentes dans la requête d'informations.

Dans notre implémentation, la géométrie d'un trait est stockée sous la forme d'une liste de sommets, chaque sommet stockant lui-même la liste d'attributs visuels (épaisseur, couleur, texture). Cette géométrie constitue l'*épine dorsale* du trait. La géométrie complète du trait, quant à elle, prendra en compte les valeurs d'épaisseur spécifiées pour les points de cette épine dorsale.

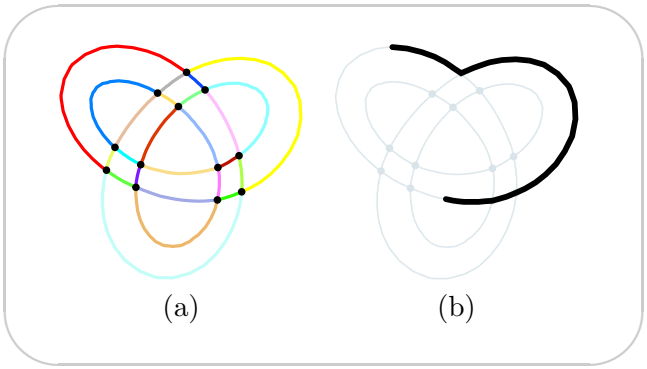


FIG. 4.5 – Traits

(a) : Graphe de vue. (b) : Un trait construit à partir de ce graphe de vue. Les points d'un trait peuvent inclure des sommets du graphe de vue.

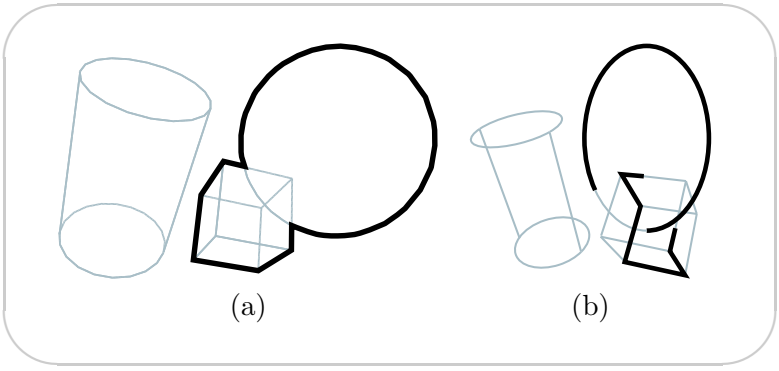


FIG. 4.6 – Les traits ne correspondent pas nécessairement à des lignes en 3D

(a) : Un trait (en noir) construit comme le chaînage de plusieurs arêtes du graphe de vue. Les projections de ces arêtes sont connexes dans le plan image. (b) : Sous un autre point de vue, on constate que ces arêtes ne sont pas connectées en 3D et donc que le trait construit correspond, en 3D, à un ensemble disjoint d'arêtes.

■ *Les traits peuvent être rééchantillonnés à la demande*

Nous proposons également deux mécanismes de rééchantillonnage des points d'un trait. Le premier mécanisme est lié à la requête d'information : comme on le verra un peu plus loin, les traits font partie des éléments manipulés dans une description de style, et doivent, à ce titre, permettre un accès aux informations en n'importe lequel de leur point. En particulier, nous verrons au prochain chapitre qu'il existe un opérateur de découpage, qui nécessite de déterminer le point du trait réalisant le minimum d'une fonction donnée. Dans ce cas, il est essentiel d'être capable de raffiner la géométrie du trait, si besoin est, pour disposer d'une évaluation suffisante de la fonction. Un trait peut donc être rééchantillonné à la demande par l'utilisateur (qui spécifiera la valeur de rééchantillonnage désirée). Dans notre implémentation, ce rééchantillonnage est virtuel dans la mesure où un sommet temporaire est créé à la demande, à la position désirée, mais n'est pas stocké de manière permanente.

Le deuxième mécanisme de rééchantillonnage concerne l'affectation d'attributs visuels : certaines mises en style imposent un échantillonnage relativement fin, qui, s'il n'est pas naturellement disponible, doit être forcé. Par exemple, si l'on veut que la couleur d'un trait varie graduellement d'une couleur à une autre entre ses deux extrémités, il faut disposer d'un nombre suffisant de points sur l'ensemble du trait. Dans les deux cas, le rééchantillonnage est régulier, et est calculé dans l'espace image.

Par constraste, il peut également être nécessaire de supprimer des sommets d'un trait. Par exemple, pour raccourcir les traits (suppression de portions extrêmes) ou pour leur donner une apparence plus polygonale (en supprimant des sommets plus ou moins régulièrement le long du trait). La figure 4.7 illustre ces deux derniers effets.

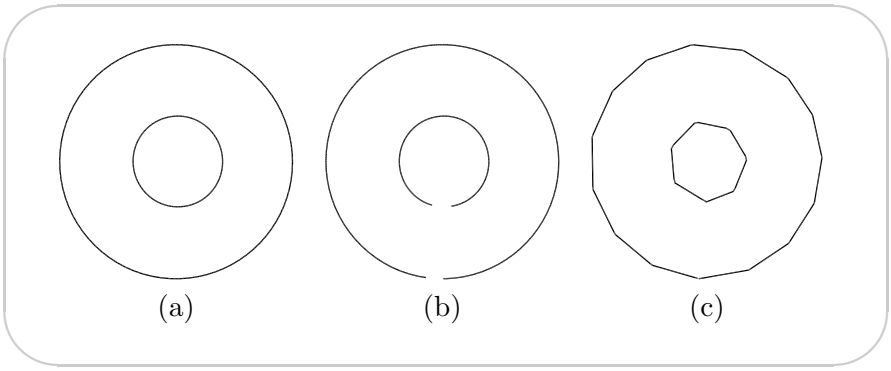


FIG. 4.7 – Interface des traits : suppression de sommets

(a) : Lignes de silhouette d'un tore. (b) : Sur cette illustration, les traits ont été raccourcis en supprimant des sommets aux extrémités. (c) : Ici, l'apparence plus polygonale est obtenue en supprimant des sommets régulièrement le long du trait.

Résumé: Les traits sont, avec les arêtes du graphe de vue, l'autre grand type d'éléments 1D manipulés. À la différence des arêtes du graphe vue, les traits peuvent inclure, parmi leurs sommets, des sommets du graphe de vue. On notera par ailleurs qu'ils ne correspondent pas forcément à des lignes en 3D dans la mesure où ils peuvent résulter du chaînage de deux arêtes connectées dans le plan mais pas dans l'espace.

4.4 Accès aux informations

Selon notre approche, une modélisation de style s'appuie fortement sur l'ensemble des informations définies à la section 4.1.2. Il est donc essentiel que ces dernières soient facilement accessibles depuis l'ensemble des éléments manipulés au sein d'une description de style. On distingue quatre types d'éléments au travers desquels les informations doivent pouvoir être accédées :

– les arêtes du graphe de vue

– les points appartenant à ces arêtes
– les traits
– les points sur les traits

On distingue deux catégories : d'un côté, les éléments 0-dimensionnels, qui regroupent les points appartenant aux arêtes du graphe de vue et les points sur les traits, et, de l'autre côté, les élémeents 1-dimensionnels, qui regroupent les arêtes du graphe de vue et les traits. Comme nous le verrons dans le prochain chapitre, la plupart des opérateurs mis à la disposition de l'utilisateur pour programmer un style peuvent travailler indifféremment avec l'un ou l'autre type de chaque catégorie. Il est donc en particulier souhaitable de fournir un accès aux informations similaire depuis chacun de ces types.

Il existe toutefois quelques différences entre chaque paire d'éléments, dont il est important de tenir compte pour aménager un accès efficace aux informations. Dans la suite de cette section, nous étudions séparément les requêtes 0- et 1-dimensionnelles.

4.4.1 Requêtes 0-dimensionnelles

4.4.2 Problèmes posés

La plupart des difficultés associées à ce type de requêtes proviennent, d'une part, du fait qu'il existe des points pour lesquels l'information n'est pas définie et, d'autre part, des ambiguïtés qui peuvent naître d'une requête à une information 3D depuis un environnement 2D (ambiguïtés liées à la projection inverse).

Comme nous l'avons écrit, les requêtes 0-dimensionnelles peuvent être faites sur des points appartenant à des arêtes du graphe de vue ou à des traits. Dans le premier cas, l'accès à l'information ne pose pas de problème. En revanche, dans le deuxième, un point peut correspondre à un point d'une arête du graphe de vue ou à un sommet du graphe de vue, et pour cette dernière configuration, une requête peut être problématique. En effet, la fonction d'information $\mathcal{I}$ étant, par définition, continue sur l'ensemble $\mathcal{E}$ des arêtes du graphe de vue, toute requête d'informations en un point d'une de ces arêtes ne présente pas de difficulté particulière. Au contraire, une requête d'informations en un sommet du graphe n'est pas forcément définie. En effet, si on a n arêtes connectées à ce sommet, on a potentiellement n vecteurs d'informations différents possibles pour la valeur de la fonction d'information en ce sommet.

Requêtes Contextuelles

On aimerait donc définir une fonction de requête d'information $\mathcal{I}^{\mathcal{G}} : \mathcal{G} \rightarrow I$ qui à un point P du graphe associe un unique vecteur d'information.

Comme nous l'avons écrit, pour tous les points du graphe qui ne sont pas des sommets, $\mathcal{I}^{\mathcal{G}}$ est bien définie :

$$\forall P \in \left(\bigcup_i E_i \right), \mathcal{I}^{\mathcal{G}} = \mathcal{I}$$

En revanche, pour tous les points correspondant aux sommets du graphe, $\mathcal{I}^{\mathcal{G}}$ n'est pas nécessairement définie. Pour lever cette indéfinition, il faut décider d'une valeur pour $\mathcal{I}^{\mathcal{G}}$ aux points de discontinuité. Mais quel vecteur d'information choisir parmi les n possibilités ? En réalité, dans notre approche, le nombre de possibilités est moins important. En effet, une requête 0-dimensionnelle sur un sommet se fait toujours dans le contexte d'un élément 1D,

c'est-à-dire qu'un point est toujours considéré comme faisant partie d'un trait. Grâce à ce contexte, le nombre de possibilités, pour l'information en un sommet du graphe de vue, passe de n à 2 (ou 1 lorsque le sommet correspond à une extrémité du trait) et il est possible de renvoyer une valeur pertinente pour le vecteur d'information. Cette valeur peut par exemple être la limite à gauche ou à droite, ou bien même une combinaison quelconque de ces valeurs.

Le contexte d'un point P du graphe est constitué soit d'une arête de $\mathcal{E}$, si $P \in \bigcup_i E_i$, soit de deux arêtes si $P \in V(\mathcal{G})$ (ces deux arêtes sont alors des arêtes adjacentes au sommet P). La figure 4.8 illustre l'apport d'un contexte pour une requête en un sommet du graphe.

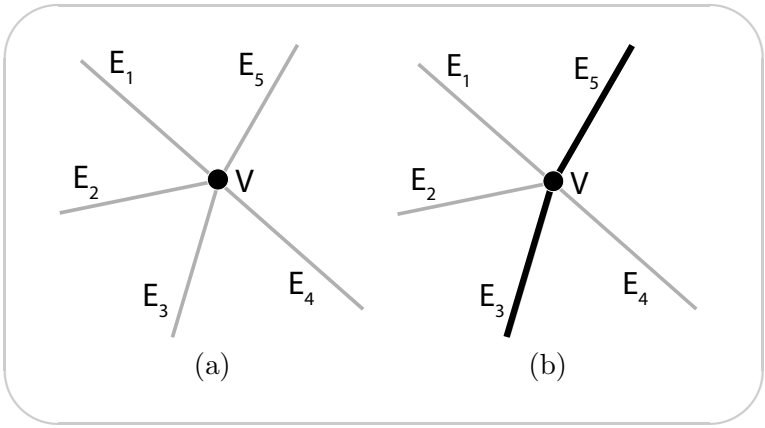


FIG. 4.8 – Requêtes 0-dimensionelles et contexte

(a) : Sur cet exemple, le vecteur d'information peut potentiellement prendre cinq valeurs différentes au sommet V. (b) : Dans le contexte de l'élément 1D (E_3, E_5), le nombre de valeurs possibles est réduit à 2. Il devient notamment raisonnable de choisir une valeur à gauche sur E_3 ou à droite sur E_5 pour V.

On définit donc la fonction $\mathcal{I}^{\mathcal{G}}$ de requête d'information, non pas sur $\mathcal{G}$ mais sur $\mathcal{G} \times \mathcal{E} \times \mathcal{E}$.

Soient E_i et E_j deux arêtes de $\mathcal{G}$ connectées à un même sommet V. Il existe alors $(P_n^i)_{n\in\mathbb{N}}$, une suite de points de l'arête E_i et $(P_n^j)_{n\in\mathbb{N}}$, une suite de points de l'arête E_j, telles que :

$$\begin{cases} \mathbb{P}(\lim\limits_{n\to\infty} P_n^i) = V \\ \mathbb{P}(\lim\limits_{n\to\infty} P_n^j) = V \end{cases}$$

$\mathbb{P}$ étant la projection qui transforme le repère monde en le repère caméra.

On définit $\mathcal{I}^{\mathcal{G}}$ comme :

$$\begin{array}{rccc} \mathcal{I}^{\mathcal{G}} : & \mathcal{G} \times \mathcal{E} \times \mathcal{E} & \longrightarrow & I \\ & (V, E_i, E_j) & \longmapsto & f(\lim\limits_{n\to\infty} \mathcal{I}(P_n^i), \lim\limits_{n\to\infty} \mathcal{I}(P_n^j)) \end{array}$$

où f est une fonction de $I \times I$ dans I qui décide notamment de quelle valeur renvoyer en cas de discontinuité. On peut par exemple définir f par défaut comme :

$$\begin{array}{rccc} f : & I \times I & \longrightarrow & I \\ & (\Lambda_1, \Lambda_2) & \longmapsto & \Lambda_1 \end{array}$$

Si V n'est pas un sommet, $E_i = E_j$ et $\lim\limits_{n\to\infty} \mathcal{I}(P_n^i) = \mathcal{I}(\lim\limits_{n\to\infty} P_n^i) = \mathcal{I}(V)$

Si V est un sommet, alors E_i et E_j sont deux arêtes distinctes connectées à V et $\mathcal{I}^{\mathcal{G}}(V, E_i, E_j)$ est égale à la limite à gauche de $\mathcal{I}$ en V sur l'élément 1-dimensionel constitué par E_i et E_j.

En pratique, la fonction f est plus spécialement dédiée au comportement à adopter aux sommets du graphe de vue (en cas de discontinuité), les requêtes 0-dimensionelles étant parfaitement définies pour tout autre point.

Le choix de la limite à gauche comme valeur de l'information en un point de discontinuité n'est pas nécessairement le plus adapté à l'utilisation que l'on veut en faire. L'utilisateur peut spécifier quelle fonction f utiliser, c'est-à-dire quel comportement adopter en cas de discontinuité.

Comme nous le verrons à la section 4.5, l'implémentation des requêtes contextuelles est faite par le biais d'itérateurs.

Remarque : L'accès au contexte 1D lors d'une requête 0D permet en outre d'introduire un paramètre d'échelles dans la requête à certaines informations. Par exemple, une requête à l'information de courbure en un point pourra se faire à une échelle plus ou moins grande selon le but poursuivi par l'utilisateur.

Continuité partielle

Bien que l'apport d'un contexte aux requêtes 0-dimensionelles permette de définir des valeurs pour l'information aux sommets du graphe, celle-ci reste discontinue pour les sommets problématiques.

Toutefois, si on considère un sommet V, bien que $\mathcal{I}^{\mathcal{G}}$ y soit discontinue, il est possible qu'un sous-ensemble des informations y soit, lui, continu. La figure 4.9 illustre cette propriété sur l'information d'invisibilité quantitative. Il est alors important d'être capable de tester la continuité d'une partie des informations en un point afin, notamment, de savoir s'il est nécessaire de traiter ce point comme un cas particulier (un point de discontinuité pour lequel une fonction spéciale a été définie et doit être appelée) ou non.

On définit une fonction de requête d'information partielle $\mathcal{I}_k^{\mathcal{G}}$ qui à un point du graphe associe la composante k du vecteur d'information.

Soit Π_k la fonction qui à un vecteur d'informations défini dans I associe la composante k définie dans I_k :

$$\begin{array}{rccc} \mathcal{R}_k : & I & \longrightarrow & I_k \\ & (\lambda_0, \ldots, \lambda_m) & \longmapsto & \lambda_k \end{array}$$

On définit alors $\mathcal{I}_k^{\mathcal{G}}$ comme :

$$\begin{array}{rccc} \mathcal{I}_k^{\mathcal{G}} : & \mathcal{G} \times \mathcal{E} \times \mathcal{E} & \longrightarrow & I_k \\ & (V, E_i, E_j) & \longmapsto & \Pi_k(\mathcal{I}^{\mathcal{G}}(V, E_i, E_j)) \end{array}$$

En pratique on ne fait appel à la fonction f, qui décide de quelle valeur retourner en cas de discontinuité, que lorsqu'on rencontre une discontinuité. En particulier, si la fonction $\mathcal{I}_k^{\mathcal{G}}$ est continue au sommet V dans le contexte considéré, il n'est pas nécessaire d'y avoir recours.

On dit que $\mathcal{I}_k^{\mathcal{G}}$ est continue en un sommet V dans le contexte E_i, E_j ssi :

$$\lim_{n \to \infty} \mathcal{I}_k^{\mathcal{G}}(P_n^i) = \lim_{n \to \infty} \mathcal{I}_k^{\mathcal{G}}(P_n^j)$$

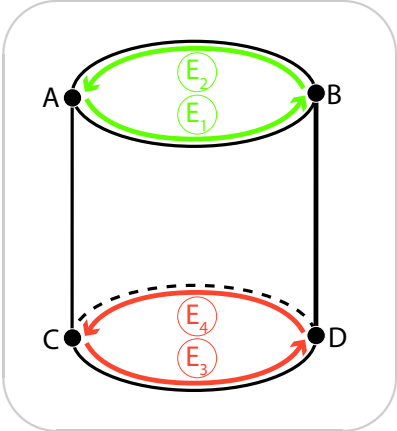


FIG. 4.9 – Continuité partielle de l'information

Sur cet exemple, l'information est généralement discontinue sur $\overline{E_1 \cup E_2}$ *et* $\overline{E_3 \cup E_4}$ *(e.g. l'information de nature des lignes). En revanche, si on considère uniquement l'information d'invisibilité quantitative, elle est continue sur* $\overline{E_1 \cup E_2}$ *et est discontinue sur* $\overline{E_3 \cup E_4}$*.*

Remarque : Les problèmes posés par la requête d'informations 3D en un point 2D, et que nous avons décrits précédemment, sont incontournables en NPR. La solution consistant à enrichir toute requête en un point d'un contexte 1D impose une certaine technicité au travail demandé à l'utilisateur dans la mesure où elle lui laisse la charge de définir les valeurs d'une fonction d'information en ses points de discontinuité. Il semble toutefois difficile d'éviter ce travail à l'utilisateur sans lui enlever un contrôle fondamental.

4.4.3 Requêtes 1-dimensionnelles

Les requêtes 1-dimensionnelles servent à récupérer l'information associée à un élément de dimension 1. En pratique, ces requêtes concernent les arêtes du graphe de vue ou les traits.

Puisque l'information est définie par point, ces requêtes se font, dans le cas général, en "intégrant" le long de l'élément 1-dimensionnel l'information obtenue en chaque point à l'aide de requêtes 0-dimensionnelles. En pratique, l'"intégration" peut être n'importe quelle opération d'agrégation permettant de combiner un ensemble de valeurs en un seul résultat pour l'ensemble de la ligne.

Il y a plusieurs questions associées à ces requêtes. Premièrement, toutes les informations ont-elles un sens pour un élément 1-dimensionnel ? Ensuite, si l'information est par définition continue le long d'une arête du graphe de vue, ce n'est pas nécessairement le cas pour les traits qui peuvent regrouper plusieurs arêtes. Que faire en cas de discontinuité de l'information demandée le long de l'élément 1-dimensionnel ? Finalement, de nombreuses opérations permettent d'agréger un ensemble de valeurs en une seule mais toutes ne sont pas pertinentes dans tous les cas. Quelles fonctions d'agrégation proposer ?

Il y a deux cas de figure à considérer pour chaque information :

1. l'information est continue le long de l'élément 1-dimensionnel (il s'agit alors d'une arête du graphe de vue ou d'un trait joignant deux arêtes sur lesquels l'information est continue)

2. l'information présente une ou plusieurs discontinuités le long de l'élément 1-dimensionnel

On identifie quatre opérations usuelles qui ont prouvé, au cours de nos expérimentations, être utiles pour combiner les informations obtenues pour l'ensemble des points de l'élément 1D :

La moyenne Renvoie la moyenne des valeurs correspondant aux requêtes 0-dimensionnelles. Dans le cas des informations constantes par morceaux, le résultat ne fera généralement pas partie des valeurs initiales.

Le maximum Renvoie le maximum des valeurs correspondant aux requêtes 0-dimensionnelles.

Le minimum Renvoie le minimum des valeurs correspondant aux requêtes 0-dimensionnelles.

L'union Renvoie l'union des valeurs correspondant aux requêtes 0-dimensionnelles. L'union n'est pas exactement une fonction d'intégration dans la mesure où elle ne renvoie pas une unique valeur mais une liste de valeurs. Elle est cependant très utile dans de nombreux cas et doit pour cette raison figurer parmi les fonctions proposées. Cette opération n'est valable que pour les informations discrètes.

On pourrait bien entendu imaginer d'autres types d'agrégation et l'utilisateur peut, s'il le désire, définir ses mécanismes personnalisés.

Nous avons choisi d'étudier le comportement de chaque information par rapport à ces quelques opérations afin de rendre plus concrète la signification d'une requête 1D et des problèmes qui peuvent se poser. Pour chaque information, on considère les deux cas possibles de continuité le long de l'élément 1D et on examine si l'intégration des requêtes 0D, en utilisant chacune des quatre opérations exemples, a du sens. Cette étude est synthétisée sous la forme d'un tableau pour chaque information. Des exemples d'utilisation sont donnés pour certaines requêtes, à la suite du tableau, et un certain nombre de remarques générales, liées au type de l'information par exemple, sont formulées à la fin de cette section.

Remarque : L'évaluation, le long d'un élément 1D, d'une information 3D définie dans $\mathbb{R}$ est problématique. En effet, la régularité de l'échantillonnage d'un tel élément est seulement garantie en 2D. Par conséquent, l'utilisation de la moyenne ou de toute autre opération d'agrégation sur une information 3D risque de donner des résultats incohérents. L'accès à une paramétrisation 3D régulière (au moins pour les morceaux 3D connexes) fait partie des développements futurs. Nous indiquons dans la suite, des exemples de traitements pour les informations 3D qui supposent qu'une telle paramétrisation est accessible.

Coordonnées 3D

	Moyenne	Maximum/Minimum	Union
Continu	√	√	*X*
Discontinu	√	√	*X*

D'une manière générale, cette information permet de prendre des décisions impliquant la position d'un élément 1D par rapport à une boite d'encombrement, cette dernière étant définie en 3D. Avec la moyenne, on peut, par exemple, imaginer sélectionner les traits (ou arêtes) dont le barycentre appartient à une région 3D. Avec le minimum et le maximum, on est capable de sélectionner les traits complètement inclus dans une région 3D ou l'intersectant.

Remarque : Nous illustrerons souvent l'exploitation d'une information donnée par une opération de sélection. Cet opérateur (que nous présentons dans le prochain chapitre) nous semble en effet être celui qui met le plus en évidence l'information utilisée.

Remarque : Lorsque le sens d'une requête est similaire dans le cas continu et le cas discontinu, on parlera de traits (cas le plus général) pour désigner les éléments 1D (arêtes ou traits).

Profondeur

	Moyenne	Maximum/Minimum	Union
Continu	√	√	X
Discontinu	√	√	X

Les traitements envisageables pour cette information sont similaires à ceux utilisés pour illustrer l'information de coordonnées 3D.

Coordonnées 2D

	Moyenne	Maximum/Minimum	Union
Continu	√	√	X
Discontinu	√	√	X

Les traitements envisageables pour cette information sont similaires à ceux utilisés pour illustrer l'information de coordonnées 3D.

Normales 3D

	Moyenne	Maximum/Minimum	Union
Continu	√	√	X
Discontinu	√	√	X

D'une manière générale, cette information permet de prendre des décisions impliquant l'orientation des surfaces aux points par lesquels un trait passe. En utilisant la moyenne, il est, par exemple, possible de sélectionner les traits qui correspondent à des lignes tracées sur des surfaces dont l'orientation moyenne (en ces points) est dans un certain angle solide. De la même manière, en analysant l'information fournie par les opérations min/max composante par composante, il est possible de sélectionner les traits correspondant à des surfaces dont l'orientation est garantie appartenir à un certain angle solide.

Normales 2D

	Moyenne	Maximum/Minimum	Union
Continu	√	√	X
Discontinu	√	√	X

L'utilisation de cette information est similaire à celle des normales 3D.

Courbures 2D

	Moyenne	Maximum/Minimum	Union
Continu	√	√	X
Discontinu	√	√	X

Pour cette information, la moyenne, le minimum et le maximum permettent de sélectionner les traits dont la courbure moyenne, respectivement, minimum et maximum est dans un intervalle donné.

Nature des arêtes

	Moyenne	Maximum/Minimum	Union
Continu	X	X	$\surd$(1)
Discontinu	X	X	$\surd$

Dans le cas discontinu, l'union des natures des différentes arêtes présentes dans un trait peut permettre de déterminer si ce trait appartient, au moins par l'un de ses segments, à une famille de traits. On peut, par exemple, ainsi sélectionner l'ensemble des traits dont au moins une partie appartient au contour extérieur.

Invisibilité quantitative

	Moyenne	Maximum/Minimum	Union
Continu	$\surd$	$\surd$	$\surd$(1)
Discontinu	X	$\surd$	$\surd$

À priori, la moyenne d'un ensemble d'invisibilités quantitatives ne fournit pas d'information réellement exploitable. En outre, elle est, a priori du type réel alors que le type de base pour cette information est $\mathbb{N}$. Pour ces raisons, nous avons considéré que l'opération de moyennage de cette information, dans le cas discontinu, n'avait pas de sens. On peut, en revanche, imaginer utiliser les opérations min/max pour sélectionner les traits qui sont cachés au plus ou au moins par un certain nombre de surfaces. Finalement, avec l'opération d'union, on sait, en plus, dans quelle proportion telle ou telle valeur d'invisibilité quantitative est présente dans le trait.

Objets occultants

	Moyenne	Maximum/Minimum	Union
Continu	X	X	$\surd$(1)
Discontinu	X	X	$\surd$

La principale utilisation de cette information consiste à vérifier si un trait est masqué par tel ou tel objet.

Objets cachés

	Moyenne	Maximum/Minimum	Union
Continu	X	X	$\surd$(1)
Discontinu	X	X	$\surd$

De manière similaire à l'information précédente, on peut ainsi savoir si un trait cache tel ou tel objet.

Matériau

	Moyenne	Maximum/Minimum	Union
Continu	$\surd$	X	$\surd$(2)
Discontinu	$\surd$(3)	X	$\surd$(2)

Bien qu'ayant mathématiquement un sens, les opérations de min/max sur des couleurs de matériau ne semblent pas présenter une grande validité dans l'espace de couleurs RGB et ont, pour cette raison, été marquées en rouge. Concernant la moyenne, on peut imaginer attribuer à un trait la couleur correspondant à la moyenne des couleurs des arêtes du graphe de vue la composant. Enfin, l'union permet par exemple de tester si un trait possède tel ou

tel matériau (et en quelle quantité) parmi les matériaux associés aux arêtes du graphe de vue composant ce trait.

Identité des objets

	Moyenne	Maximum/Minimum	Union
Continu	X	X	√(1)
Discontinu	X	X	√

Permet de savoir si un trait dessine, au moins partiellement, un objet donné.

Densité

	Moyenne	Maximum/Minimum	Union
Continu	√	√	X
Discontinu	√	√	X

Le chapitre 6 propose une présentation complète de l'utilisation de l'information de densité.

Adjacence du graphe de vue

Il n'existe pas, pour cet information, de mécanismes standard. En effet, on pourrait imaginer une infinité de manières d'exploiter les différents types de renseignements fournis par chaque requête 0D à cette information. Par exemple, une requête 1D pourrait consister à vérifier si un trait contient un certain nombre d'intersections en T, ou encore, s'il est adjacent à tel autre trait, *etc.*...

Abscisse curviligne

L'abscisse curviligne correspond à la paramétrisation du trait. On connaît donc exactement ses valeurs et une requête 1D à cette information n'a pas de signification.

Longueur totale du trait

On parle ici de longueur 2D dans la mesure où la longueur 3D d'un trait n'est pas nécessairement définie (dans le cas où l'information de coordonnées 3D est discontinue sur ce trait). Cette information est définie pour un élément 1D (et non 0D) et sa requête est donc naturelle (et ne fait pas l'objet d'une agrégation).

(1) L'union de plusieurs valeurs identiques est réduite à un seul élément. Ainsi, dans le cas d'une information constante par morceaux prenant la même valeur sur deux arêtes, l'union renverra, comme on est en droit de l'attendre, un unique élément.
(2) L'union n'est pas définie pour les éléments de $\mathbb{R}$ qui varient continûment le long des arêtes. En revanche, pour les informations définies dans $\mathbb{R}$ mais qui sont constantes par morceaux sur les points du graphe, l'union est bien définie. (Lorsqu'on rencontre un nombre fini n de matériaux le long d'un élément 1D, l'union renverra la liste des n matériaux.)
(3) Pour les informations constantes par morceaux et définies dans $\mathbb{R}$ (et pas uniquement dans $\mathbb{N}$), la moyenne a un sens. Par exemple, si un élément 1D est d'une certaine couleur sur une première partie et d'une autre couleur sur l'autre, il semble raisonnable de calculer une couleur moyenne pour l'ensemble de cet élément.

Remarque : La grande diversité des informations calculées limite la factorisation d'opérateurs d'aggrégation : comme nous l'avons vu précédemment aucun des opérateurs proposés ne son

Résumé: Les mécanismes d'accès aux informations permettent, dans le cas de requêtes 0-dimensionnelles, de gérer les cas problématiques (requête en un sommet du graphe de vue), en intégrant un contexte 1D à toute requête 0D, et, dans le cas de requêtes 1D, de bénéficier de solutions d'agrégation standard.

4.5 Implémentation

4.5.1 Calcul des informations

La plupart des informations sont directement accessibles depuis le modèle 3D. C'est le cas des coordonnées 3D, des coordonnées 2D (en utilisant la matrice de projection), des normales 3D, des identités d'objets, des matériaux.

La nature des arêtes est également directement déterminée par construction des arêtes caractéristiques.

On s'appuie sur un algorithme de balayage de ligne (*sweep line*) [BKOS00] pour calculer les intersections (qui donneront lieu à une partie des sommets du graphe de vue) entre les projections des arêtes dans le plan. Les informations d'adjacence sont en même temps calculées et stockées.

Après découpage des lignes caractéristiques en arêtes du graphe de vue, on sait que les informations seront continues sur chaque arête. En particulier l'information d'invisibilité quantitative sera constante sur chacune de ces arêtes. Elle est calculée pour chacun des segments par lancer de rayons : on compte le nombre de surfaces intersectées par le rayon et qui se trouvent entre le point de vue et le segment. En raison des nombreuses erreurs numériques constatées sur ce type de calcul, lorsqu'une arête du graphe de vue est constituée de plusieurs sous arêtes issues du maillage, l'invisibilité quantitative est calculée pour chaque sous arête et le résultat est obtenu par vote. La liste des surfaces intersectées par le rayon lors de ce calcul est stockée afin de construire l'information des objets occultants. Le rayon utilisé pour calculer l'invisibilité quantitative est également prolongé au-delà du segment considéré de manière à calculer et stocker la liste des surfaces cachées. L'information de discontinuité en profondeur découle également de ce dernier calcul. Le calcul de ces informations par lancer de rayons est l'étape la plus coûteuse du système. Sa complexité varie, bien évidemment, significativement avec la manière dont les objets de la scène sont disposés (*cf.* figure 4.10). Pour disposer d'un ordre d'idée, pour le modèle du compresseur, que l'on peut voir sur la figure 7.10 page 197, et qui compte à peu près 45 000 triangles, il faut environ 15 secondes sur un Pentium 4 à 1,3 GHz pour mener ces calculs.

En sachant que l'on travaille toujours muni d'un contexte 1-dimensionnel, les informations différentielles telles que la courbure 2D ou la normale 2D peuvent être facilement calculées en tout point. De même les informations d'abscisse curviligne ou de longueur sont naturellement disponibles dans un tel contexte.

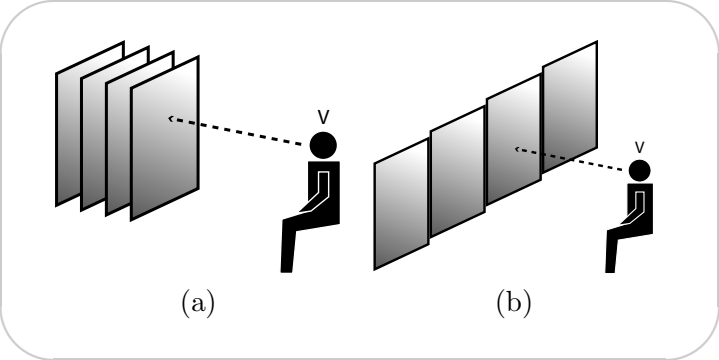


FIG. 4.10 – Complexité des calculs de visibilité

La complexité des calculs de visibilité est plus importante dans le cas (b) que le cas (a).

Pour l'information de densité, il existe plusieurs approches possibles que nous discutons dans le chapitre 6.

4.5.2 Stockage des informations

Dans notre implémentation, une arête du graphe de vue est stockée sous forme d'une polyligne, dont les segments proviennent de la tessélation du maillage 3D, elle-même ayant été éventuellement subdivisée à l'occasion du calcul d'intersections entre arêtes dans le plan.

On liste, pour chaque type d'objet (arête, sommet du graphe, segment de polyligne, sommet de polyligne) les informations qui y sont stockées :

sommet de polyligne
- Coordonnées 3D
- Coordonnées 2D
- Profondeur
- Normales 3D
- Normales 2D
- Courbure 2D
- Abscisse curviligne

segment de polyligne
- Longueur
- Disontinuité en profondeur

sommet du graphe
- Nature du sommet (*e.g.* T-Vertex)
- Adjacence du graphe

arête du graphe
- Nature des arêtes

- Invisibilité quantitative
- Surfaces occultants
- Surfaces cachées
- Longueur
- Identité des objets
- Matériau

4.5.3 Accès à l'information

Le système que nous fournissons doit implémenter des accès à l'ensemble des informations discutées précédemment depuis l'ensemble des objets manipulés (points, sommets, arêtes, traits). Dans notre mise en œuvre, nous avons adopté une approche objet dans laquelle les fonctions (d'accès à l'information) sont proposées sous forme de *functors* (objets fonctions, [Ale01]). Ces objets présentent la même interface d'appel qu'une fonction (grâce au mécanisme de surcharge de l'opérateur "()" tout en offrant les avantages fournis par l'approche objet (initialisation, stockage de données membres,*etc.*...). Comme nous le verrons dans le prochain chapitre, les langages de programmation que nous utilisons, à savoir *C++* et *Python*, offrent un tel mécanisme.

Les fonctions d'accès à l'ensemble des informations sont donc disponibles sous forme de *functors* travaillant soit sur des éléments 0D soit sur des éléments 1D.

Contexte

Nous avions remarqué que pour les requêtes 0-dimensionnelles, il fallait disposer du contexte 1D de manière à potentiellement lever d'éventuelles ambiguïtés.

Dans notre implémentation, le contexte 1D d'un élément 0D est encapsulé dans un itérateur [GHJV95] pointant sur cet élément. Cette approche permet au programmeur de parcourir la chaîne 1D de part et d'autre du point courant et en particulier de récupérer les informations stockées à la droite comme à la gauche de ce point.

Nous avons expliqué que des mécanismes de rééchantillonnage étaient disponibles au sein des traits. Ces mécanismes sont accessibles au travers de tels itérateurs, qui permettent à l'utilisateur de spécifier la valeur d'échantillonnage avec laquelle il faut travailler.

Fonctions 0-dimensionnelles

Les fonctions 0-dimensionnelles sont des *functors* travaillant sur des itérateurs 0D, les éléments pointés par ces derniers pouvant être des points sur les arêtes ou des sommets du graphe. Le type de retour varie en fonction de l'information accédée par la fonction.

Des fonctions d'accès à l'ensemble des informations définies sont fournies de manière standard dans le système.

Fonctions 1-dimensionnelles

Comme on a pu le voir précédemment, dans la grande majorité des cas une requête d'information sur un élément 1D est calculée comme une "intégration" de plusieurs valeurs 0D (obtenues chacune par requête 0D). On fournit ce mécanisme au travers d'une fonction *integrate* qui à partir d'un *functor* 0D, d'un itérateur 0D pointant sur le premier point de

l'élément 1D considéré et du type d'intégration (*e.g.* moyenne, max) à utiliser, renvoie une valeur globale à tout l'élément 1D.

Ainsi, la plupart du temps, un *functor* 1D se limite à un appel à ce mécanisme, le type d'intégration faisant partie des données membres communes à tous les *functors* 1D.

Remarque : Nous verrons dans le prochain chapitre que l'utilisateur a la possibilité de définir ses propres fonctions dans le langage de description de style. Dans ce contexte, les mécanismes d'intégration décrits dans cette section lui sont bien évidemment accessibles.

Remarque : Grâce au mécanisme des itérateurs, il est possible d'utiliser la fonction integrate sur une portion seulement d'un élément 1D. En particulier, on peut ainsi facilement programmer des requêtes 0-dimensionnelles, pour lesquelles la valeur retournée est calculée (comme une moyenne par exemple) à l'aide de l'ensemble des valeurs se trouvant (sur l'élément 1D) dans une certaine fenêtre autour du point considéré. Dans le futur, nous aimerions développer davantage cette stratégie de manière à inclure, pour toute requête 0D, une valeur d'échelle, indiquant la taille de la fenêtre à utiliser autour du point de requête, c'est-à-dire le nombre de valeurs à moyenner, *etc.*..., pour calculer la valeur en ce point.

Résumé: En particulier, les requêtes 0D contextuelles sont implémentées grâce à des itérateurs, qui permettent d'itérer sur l'ensemble des sommets 0D d'un élément 1D.

4.6 Dans quelle mesure les informations sont-elles génériques ?

4.6.1 Discussion sur la généralité des informations

Certaines informations peuvent être utilisées de manière spécifique à la scène ou au dessin

Nous avons, jusqu'ici, écrit que l'ensemble des informations utilisées étaient non spécifiques à la scène de manière à définir un style transposable d'une scène à l'autre. Il faut en réalité nuancer cette affirmation. Si toutes les informations sont génériques dans la mesure où elles existent toutes dans n'importe quelle scène 3D, l'utilisation de certaines d'entre elles peut, quant à elle, être spécifique à une scène donnée et ne pas être transposable directement à une scène différente. Par exemple, l'information d'identité des objets existe dans toute scène mais est la plupart du temps utilisée pour sélectionner l'ensemble des lignes d'un objet particulier de la scène considérée afin de leur appliquer un style particulier. Un style s'appuyant sur cette information permet de produire une image ou une séquence d'images de la même scène, mais requiert quelques modifications pour être appliqué à une autre scène (*i.e.* mettre à jour l'identité de l'objet auquel appliquer le style).

Ainsi, certaines informations sont spécifiques à la scène ou même à l'image. Par ailleurs, la spécificité ou la généralité de chacune de ces informations dépend également de son utilisation. Ainsi, par exemple, l'information de coordonnées 2D peut-elle être utilisée pour désigner une

zone dans le support de vue lui-même (par exemple, dessiner la moitié de l'image dans tel style et l'autre moitié différemment), ou pour désigner les positions des projections d'objets de la scène. Dans le premier cas, l'information est parfaitement générique est peut être transposée à une autre scène directement. Dans le deuxième cas (illustré par la figure 3.14 page 70), bien que l'information soit générique, l'utilisation qui en faite est spécifique à la scène (et même à la vue) et il est alors nécessaire de modifier des paramètres pour l'appliquer à une autre scène.

On peut donc distinguer, pour l'ensemble des informations, celles qui sont complètement génériques, celles qui peuvent être génériques ou spécifiques selon l'utilisation qui en est faite, et celles qui sont complètement spécifiques. Le tableau suivant résume cette catégorisation :

	Générique	Spécifique
Coordonnées 2D	Quand s'applique au support de vue	(à la vue) quand s'applique aux projections d'objets de la scène
Coordonnées 3D	Quand s'applique au volume de scène	(à la scène) quand s'applique aux positions des objets dans la scène
Profondeur	Quand s'applique au volume de scène	(à la scène) quand s'applique aux positions des objets dans la scène
Normales 3D	x	
Normales 2D	x	
Courbures 2D	x	
Nature	x	
Invisibilité quantitative	x	
Objets occultants		à la scène
Discontinuité en profondeur	x	
Objets cachés		à la scène
Matériau	x	
Identité des objets		à la scène
Densité	x	
Adjacence du graphe de vue	x	
Abscisse curviligne	x	
Longueur des lignes	x	

4.6.2 Discussion sur la normalisation des informations

■ ***Une information réellement générique doit être normalisée pour être cohérente d'une scène à l'autre***

Même au sein des informations réellement génériques, il peut être nécessaire de modifier des seuils lors de la transposition d'un style d'une scène à l'autre. Par exemple, supposons qu'on ait défini un style qui s'applique uniquement à une zone donnée de l'espace sur une scène et qu'on l'applique à une autre scène. Si le rapport entre les tailles des boites englobantes des

deux scènes est important, la zone d'application du style sera très différente entre les deux scènes. Il pourrait dans ce cas être souhaitable de disposer d'une version normalisée de cette information.

Par exemple, pour l'ensemble des informations de coordonnées géométriques (coordonnées 3D, 2D, profondeur), on pourrait imaginer une stratégie de normalisation s'appuyant sur le volume de vue. Une normalisation de l'information de densité, qui est étudiée en détail au chapitre 6, pourrait consister à calculer la moyenne et l'écart type des valeurs de densités présentes pour une scène donnée et à modifier les valeurs de manière à se ramener à une moyenne et un écart-type standard.

Résumé: L'ensemble des informations est générique dans la mesure où elles existent toutes dans n'importe quelle scène 3D. En revanche, l'utilisation qui en est faite peut être spécifique à une scène donnée et un style peut par conséquent nécessiter quelques modifications pour être transposé à une autre scène. En outre, la réutilisation de styles d'une scène à l'autre peut être normalisée via une normalisation des informations.

---

chapitre 5

Programmation de style

Nous arrivons maintenant à l'étape clé de notre système : la construction et la stylisation des traits à partir des arêtes du graphe de vue (*cf.* chapitre 4). Dans notre approche, ces opérations sont programmées par l'utilisateur au sein d'une *feuille de style*. Nous proposons ici un ensemble d'*opérateurs* programmables, assimilables à des procédures, permettant à l'utilisateur de formuler aisément sa description de style. En outre, nous présentons une méthodologie pour l'utilisation de ces opérateurs, sous la forme d'un pipeline.

C'est dans cette programmation de style que les structures, les informations et les outils d'accès à ces informations, que nous avons présentés au chapitre précédent sont utilisés.

5.1 Une approche modulaire : les modules de style

Nous choisissons de diviser la formulation d'un style en plusieurs modules de style

L'ensemble des procédures qui implémentent un style pictural donné est appelé une *feuille de style*. Nous choisissons de décomposer celle-ci en une série de *modules de style*, chacun étant responsable d'une sous-partie du dessin. Les modules de style sont un moyen naturel de varier le style au sein d'un même dessin. Par exemple, l'objet principal peut être dessiné à l'aide d'un module de style différent du reste de la scène, ou encore, les lignes cachées peuvent être dessinées à l'aide d'un module de style différent des lignes visibles. La modélisation d'un style complexe peut ainsi être naturellement divisée en tâches plus simples, chacune étant spécifiée dans un module de style différent.

Lors de la composition d'un style complexe à partir de plusieurs modules de style, ces derniers sont appliqués de manière séquentielle à l'ensemble des arêtes du graphe de vue. Cette approche est similaire à l'approche par calques couramment utilisée par les logiciels de dessin commerciaux tels que Photoshop et Illustrator. Différents modules de style peuvent s'appliquer à différents sous-ensembles d'arêtes, mais également à un même sous-ensemble d'arêtes. Par exemple, pour obtenir un effet de croquis, un premier module de style pourra tracer les lignes avec des traits crayonnés et un deuxième pourra tracer ces mêmes lignes avec des traits encrés. C'est finalement la superposition des lignes tracées dans ces deux styles qui constitue le dessin final. La figure 5.1 reprend la partie droite du pipeline représenté sur la figure 3.26 page 81 pour lequel la décomposition en modules de style est mise en évidence.

Le travail d'un module de style consiste principalement à spécifier les caractéristiques topologiques des traits ainsi que leurs attributs visuels.

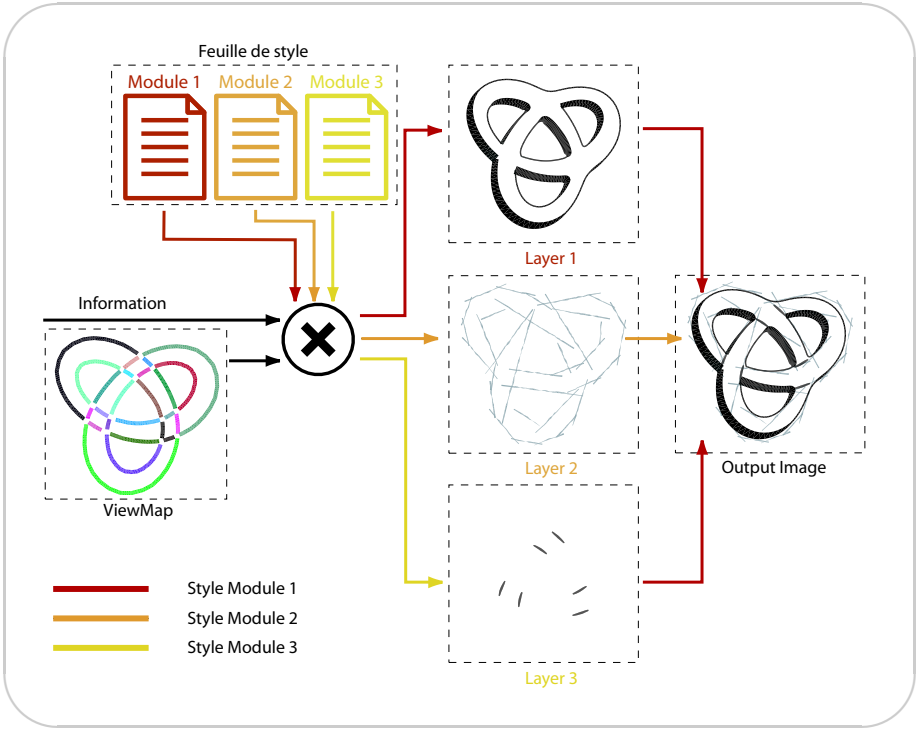


FIG. 5.1 – Décomposition en modules de style

Cette illustration reprend en partie la figure 3.26 page 81 en y ajoutant la décomposition en modules de style. Ainsi le style complexe visible sur l'image de droite est le résultat de la superposition de trois modules de style : le premier est responsable du tracé du contour extérieur dans un style calligraphique, le deuxième trace des lignes de construction tandis que le dernier dessine les lignes visibles qui n'appartiennent pas au contour extérieur.

On se pose maintenant la question de savoir quelles opérations peuvent être utilisées dans un module de style et comment elles peuvent être organisées.

5.2 Opérateurs de description de style

Identifier les opérateurs élémentaires qui constituent le processus de dessin revient à analyser les opérations nécessaires à la construction de traits à partir des arêtes du graphe de vue. Quels sont, de manière informelle, les besoins pour une mener à bien une telle construction ?

■ *La liste des opérateurs découle des besoins inhérents à la modélisation de style*

L'objectif consiste simplement à fournir une base d'opérateurs qui permettent de contrôler tous les attributs de style énoncés section 3.2, aussi bien de bas que de haut niveau. On veut, premièrement, pouvoir restreindre à un sous-ensemble d'éléments (arêtes ou traits), les actions liées à la mise en style. Cette fonctionnalité est à la base de la séparation d'un style complexe en modules de style, et donc de l'attribut de style consistant à utiliser plusieurs styles au sein d'un même dessin, ainsi que de l'omission de lignes. Deuxièmement, il faut disposer d'opérateurs permettant de spécifier la topologie d'un trait à partir d'un ensemble d'arêtes. Cela signifie que l'on doit être capable de décrire, dans le graphe de vue, un chemin (comme

une suite d'arêtes) ainsi qu'un point de départ et point d'arrêt. La figure 5.2(a) illustre un tel chemin. Nous choisissons de mettre en œuvre ce contrôle de la topologie à l'aide de deux opérations distinctes. La première est un chaînage qui permet de spécifier un chemin à suivre dans le graphe. Le résultat de cette opération est un chaîne d'arêtes (entières) du graphe, connexes deux à deux. La deuxième est un découpage qui permet de raffiner la topologie des traits en spécifiant des points de départ et d'arrêt qui ne sont pas forcément des sommets de graphe de vue. La figure 5.2(b) illustre ces opérations sur l'exemple précédent.

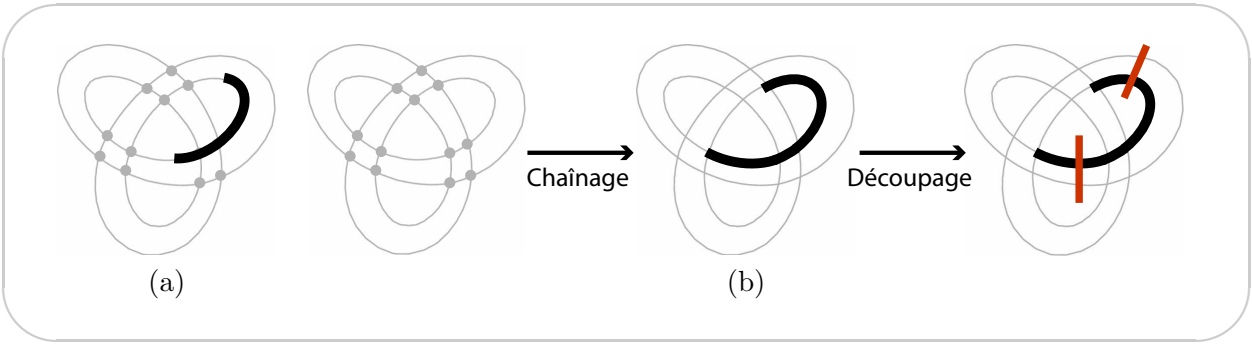


FIG. 5.2 – Contrôle de la topologie des traits

(a) : Un trait. (b) : Pour construite le trait, on commence par chaîner un nombre entier d'arêtes, puis dans un deuxième temps, on raffine la topologie en découpant ce trait aux points de départ et d'arrêt souhaités.

Ensuite, une fois la topologie spécifiée il reste encore à assigner ses attributs visuels à chaque trait. On doit donc disposer d'opérateurs dédiés à cette opération (nous les appellerons *shaders*). L'ensemble des traits ainsi construits doit finalement être tracé dans l'image finale. Un opérateur de dessin doit donc être proposé, de manière à ce que l'utilisateur puisse finalement initier le rendu des traits dans l'image finale. On identifie également un autre besoin : celui de pouvoir trier les éléments d'une séquence. En effet, comme nous le verrons plus loin, dans le processus de dessin, l'ordre dans lequel les éléments (arêtes du graphe de vue ou traits) sont traités peut avoir de l'importance.

Nous proposons un ensemble d'opérateurs pour la modélisation de style qui permettent de respecter les objectifs fixés. Pour chaque fonctionnalité recherchée (*e.g.* contrôle de la topologie), on propose un ou plusieurs opérateurs dédiés. Ces derniers peuvent être vus comme des procédures dont la programmation revient, en partie, à l'utilisateur. La (difficile) question de la validation de cette base d'opérateurs est discutée dans le chapitre 8.

Un module de style sera donc principalement constitué d'une séquence d'appels à ces opérateurs décrivant comment fabriquer, à partir de l'ensemble des arêtes du graphe de vue, les traits stylisés qui forment le dessin (*cf.* figure 5.3(a)).

En quoi cette approche est-elle programmable ? S'il ne s'agissait que d'appeler un ensemble d'opérateurs prédéfinis au sein d'un programme, le contrôle sur le style serait bien faible. En réalité, en plus de programmer la séquence d'appels aux opérateurs, l'utilisateur programme également la définition de ces derniers. En pratique, il ne s'agit pas de demander à l'utilisateur de réimplémenter complètement chaque opérateur pour chaque nouveau style, mais une petite partie seulement de cet opérateur : dans l'algorithme définissant un opérateur, on identifie la partie variable (celle qui change entre deux utilisations différentes de cet opérateur) et

la partie fixe (partie algorithmique structurelle). Cette partie variable, quand elle existe, est groupée en un ensemble de *règles*. Ce sont ces règles que devra programmer l'utilisateur et qui fixeront le comportement de l'opérateur. La figure 5.3 illustre la définition des règles de style et leur utilisation.

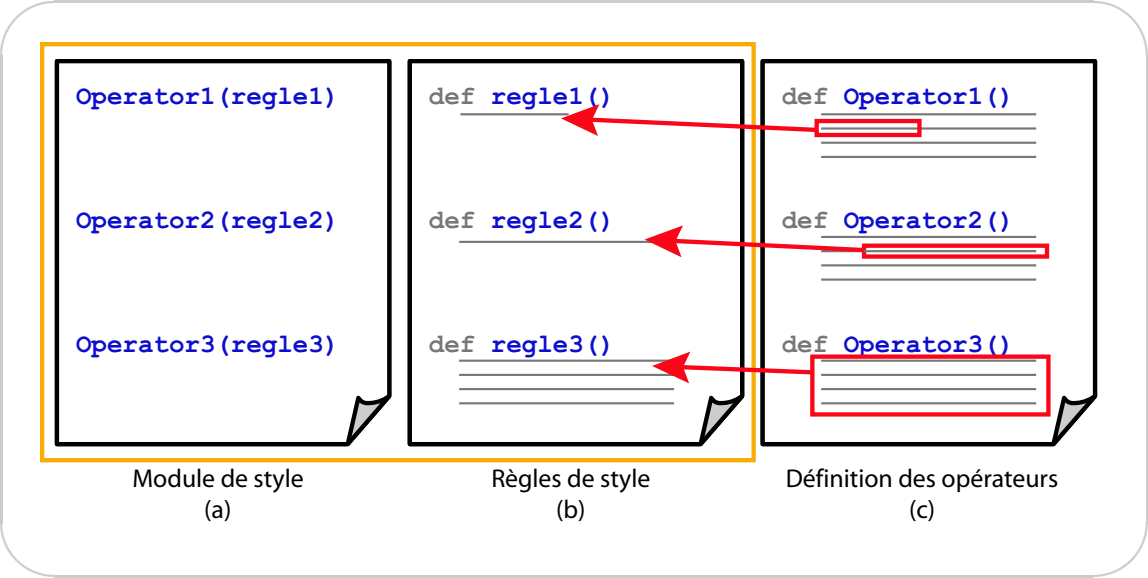


FIG. 5.3 – Programmabilité des opérateurs : les règles de style

Un module de style (a) est constitué d'appels aux opérateurs de description de style. Chaque opérateur est programmable : une partie de sa définition (cadres rouges sur la figure (c)) est externalisée sous forme de règles (b) dont l'implémentation est laissée à l'utilisateur. Les parties écrites par l'utilisateur sont entourées en orange.

Remarque : En plus d'épargner à l'utilisateur un travail de code important, le choix de programmer les opérateurs au travers de règles de style a l'avantage de permettre une meilleure factorisation et donc une meilleure réutilisation des composants développés. Toutefois, dans le cas d'opérateurs complexes, l'inconvénient de cette approche est que le nombre de règles à programmer peut être relativement important rendant le travail quelque peu fastidieux.

Dans la suite, on se pose la question de savoir quelles spécifications adopter pour ces différents opérateurs.

Pour chacun de ces opérateurs, on expose :
- le principe de fonctionnement
- les types des entrées et des sorties
- les règles
- l'algorithme

Bien que les opérateurs de topologie ou de sélection soient logiquement les premiers à être appelés, nous commençons par présenter l'opérateur lié à l'assignement d'attributs d'apparence à un trait déjà construit. En effet, la plupart du temps, les opérateurs œuvrant à la définition topologique d'un trait agissent dans la perspective d'une telle affectation d'attri-

buts. Commencer par présenter cette opération permet donc de mieux appréhender les buts à atteindre pour les opérateurs précédents.

5.2.1 Contrôle des attributs d'apparence des traits : opérateur de *shading*

Principe

Une fois les traits construits (leur topologie spécifiée), il faut leur assigner les valeurs d'attributs tels que la couleur, l'épaisseur, et la transparence. Cette étape est la plus similaire aux systèmes traditionnels de *shading*, la principale différence étant que les éléments manipulés sont ici des traits 1-dimensionnels plutôt que des fragments 0-dimensionnels. Pour cette raison, nous empruntons au rendu réaliste le terme *shading* pour désigner cette opération.

Outre les attributs énumérés ci-dessus, on considère la position spatiale de chaque point du trait comme un attribut particulier. En effet, nous remarquons que les traits d'un dessin ne suivent souvent pas exactement la géométrie sous-jacente (exception faite des illustrations techniques). Ainsi, l'opérateur de *shading* peut modifier la position spatiale des points de l'épine dorsale d'un trait. Cette technique est, d'une certaine manière, similaire à la technique de *displacement mapping*, bien connue en modélisation 3D procédurale [EMP+94].

Plusieurs opérateurs de *shading* peuvent être appliqués séquentiellement à un trait de manière à faciliter le contrôle de différents attributs au sein d'un même module de style. Par exemple, un premier *shader* peut affecter la couleur tandis que le deuxième affecte l'épaisseur. En outre, les attributs peuvent être affectés à un trait de manière absolue (la valeur précédente est remplacée) ou relative (la valeur précédente est modifiée). La modification relative d'attributs est particulièrement utile pour ajouter une petite quantité de bruit après que d'autres *shaders* aient affecté des valeurs pour un attribut donné.

La figure 5.4 montre des exemples d'opérateurs de *shading*. Lorsque la fréquence des variations d'un attribut donné est trop importante pour l'échantillonnage courant d'un trait (un attribut étant associé à un sommet du trait, cette situation se produit lorsque le nombre de sommets est insuffisant), il est possible de spécifier un rééchantillonnage adapté du trait (ce rééchantillonnage fait partie des opérations standard que la structure de trait doit proposer, *cf.* section 4.3).

Un ensemble de *shaders* simples, tels que l'affectation d'attributs constants, est fourni. Un *shader* spécial permet de spécifier le style de marques utilisé pour le rendu du trait (*cf.* section 5.4.5). L'ensemble des *shaders* standard fournis par notre système est présenté à la section 5.4.2.

Entrées/Sorties

Les opérateurs de *shading* sont implémentés comme des procédures travaillant sur les traits. On peut donc considérer que l'entrée de l'algorithme est un trait, muni d'un ensemble d'attributs, et que la sortie est le même trait avec des attributs modifiés.

Algorithme et règles

La programmation de ce type d'opérations (*e.g.* affectation d'attributs, déformation géométrique) étant relativement simple et nécessitant le maximum de flexibilité, aucune structure algo-

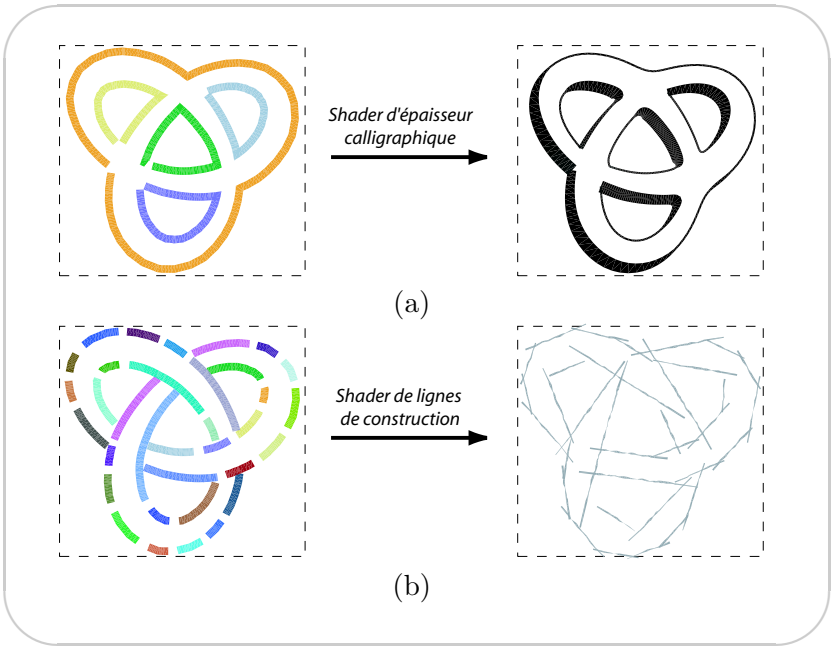


FIG. 5.4 – Opérateur de *shading*

Sur ces deux illustrations, on peut voir à gauche la topologie des traits à qui les attributs doivent être assignés. (a) : les traits dessinent les contours extérieur de l'objet. Le shader *le plus remarquable utilisé pour réaliser cette image assigne au trait une épaisseur d'autant plus importante que celui-ci est orienté selon une direction donnée, de manière à produire un effet calligraphique. Les traits ont par ailleurs été lissés à l'aide d'un* shader *de lissage géométrique. (b) : les traits correspondent ici aux silhouettes visibles, coupées aux points de haute courbure (cf. section 5.2.4). Pour imiter l'aspect de lignes de construction, ces traits ont en particulier été déformés de manière à correspondre à la tangente en leur milieu. Par ailleurs, un* shader *permettant d'étirer leur géométrie et un autre permettant de perturber l'épaisseur le long du trait ont également été utilisés sur ces traits.*

rithme n'est imposée à l'utilisateur (et donc, aucune règle n'est spécialement exposée). Programmer un *shader* revient donc à spécifier une procédure complète.

5.2.2 Stylisation sélective et omission de lignes : opérateur de sélection

Principe

L'opérateur de sélection répond au besoin de pouvoir appliquer le style de manière sélective, à un sous-ensemble des éléments 1D par exemple. En effet, comme on l'a expliqué précédemment, il est plus intéressant d'organiser un style complexe en modules de style plus simples, chacun s'appliquant à un sous-ensemble des lignes. Cet opérateur est également à la base de la fonctionnalité d'omission de lignes, pour laquelle seul un sous-ensemble de lignes respectant un critère donné est sélectionné pour être dessiné.

L'opérateur de sélection peut intervenir à plusieurs niveaux. Au début du processus, pour sélectionner le sous-ensemble des arêtes du graphe de vue auquel appliquer le style, ce sous-ensemble pouvant par exemple être l'ensemble des arêtes qui appartiennent à la silhouette. On peut également avoir besoin de faire une sélection sur les traits une fois que ceux-ci sont

construits. Par exemple, si on ne veut garder que les traits dont la longueur est supérieure à un seuil donné, on utilisera l'opérateur de sélection directement après que les traits aient été construits. On peut également vouloir garder les traits dont les attributs correspondent à une spécification donnée. On appliquera alors la sélection sur les traits après que les attributs leur aient été assignés. La figure 5.5 montre des exemples de sélections réalisées à l'aide de cet opérateur.

Un appel à l'opérateur de sélection, dans un module de style, n'est pas obligatoire. En son absence, l'ensemble des arêtes du graphe de vue est considéré.

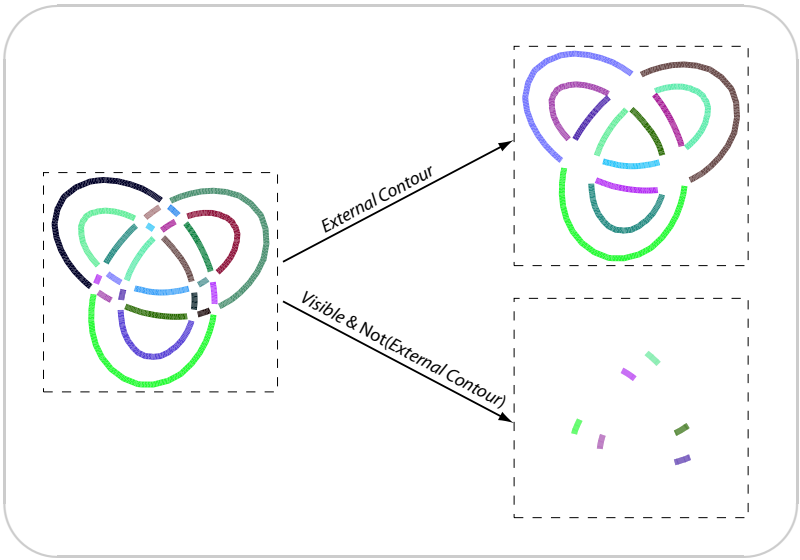


FIG. 5.5 – Opérateur de sélection

Sur cette figure, on peut voir deux sélections différentes, réalisées à partir du même ensemble d'arêtes du graphe (l'ensemble des arêtes visibles). **haut :** *La règle de sélection teste ici si une arête appartient au contour extérieur ou non.* **bas :** *On a ici sélectionné le sous-ensemble des arêtes visibles mais n'appartenant pas au contour extérieur.*

Entrées/Sorties

L'opérateur de sélection travaille sur des éléments 1-dimensionnels. En fonction de la place dans le processus de dessin à laquelle cet opérateur est utilisé, les éléments manipulés peuvent être :

- des arêtes du graphe de vue
- des traits

La sortie de cet opérateur est un sous-ensemble des éléments d'entrée. Les entrées et les sorties sont donc des éléments de même type.

Règle

La règle de sélection est une condition portant sur un élément 1D. Si cette condition est vérifiée, l'élément est sélectionné, sinon il est rejeté. Cette règle est passée à l'opérateur de sélection sous forme d'un prédicat.

> **Rappel :** Un prédicat est un opérateur logique qui retourne $vrai$ ou $faux$. On distingue les prédicats unaires, qui évaluent une condition sur un élément des prédicats binaires qui évaluent une condition sur deux éléments. Les prédicats binaires englobent notamment les opérateurs de comparaison entre éléments.

Algorithme

L'algorithme pour l'opérateur de sélection (*cf.* algorithme 5.1) prend en argument un prédicat unaire travaillant sur des éléments 1D, pouvant être des arêtes du graphe de vue ou des traits. De même les éléments 1D manipulés en entrée et en sortie peuvent être des arêtes ou des traits.

Algorithme 5.1 Opérateur de sélection

Fonction *Select*
Entrée: Π : un prédicat unaire opérant sur des éléments 1-dimensionnels, $\mathcal{E}$: un ensemble d'éléments 1D
Sortie: $\mathcal{E}'$: un ensemble d'éléments 1D
1. $\mathcal{E} \leftarrow$ ensemble actif d'éléments 1-dimensionnels
2. $\mathcal{E}' \leftarrow \{\emptyset\}$
3. **pour** $E \in \mathcal{E}$
4. **si** $\Pi(E) = vrai$
5. **alors** $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{E\}$
6.

5.2.3 Contrôle de la topologie des traits : opérateurs de chaînage

Principe

La topologie des traits est une composante importante du style qui doit pouvoir être contrôlée de manière flexible à l'aide d'un ou plusieurs opérateurs. Spécifier la toplogie d'un trait revient en réalité à décrire un chemin dans le graphe de vue (*cf.* section 3.2.2). En effet, ce dernier fournit des informations de connectivité entre les arêtes caractéristiques, alors qu'un dessin de lignes est fait de chemins 1-dimensionnels.

On doit donc disposer d'un opérateur permettant de suivre un chemin dans le graphe de vue et de construire le trait constitué par la suite 1-dimensionnelle des arêtes formant ce chemin. C'est le rôle de l'opérateur de chaînage. L'algorithme de chaînage proposé à cet effet parcourt le graphe en partant d'une arête donnée. Une règle spécifiée par l'utilisateur permet de choisir pour une arête quelconque, quelle arête prendre, parmi les arêtes candidates, pour poursuivre le parcours. Une autre règle permet de définir à quelle arête stopper. L'ensemble des arêtes parcourues forment alors une chaîne qui correspond au futur trait.

En pratique, si un appel à l'opérateur de sélection précède un appel à l'opérateur de chaînage, seules les arêtes ayant été sélectionnées pourront initier des traits. En outre, par défaut, une arête parcourue une première fois pendant le chaînage ne sera pas chaînée à nouveau. Ce comportement peut être modifié par l'utilisateur, qui peut ainsi décidé de construire des chaînes qui s'étendent hors de la sélection (*cf.* figure 5.7 en bas). Un chaînage pour lequel

une même arête est chaînée plusieurs fois au sein d'un même trait, permet en particulier d'obtenir un aspect croquis réaliste (cet effet sera illustré plus loin sur la figure 5.14 (c)).

On distingue deux manières de parcourir le graphe à partir d'une arête de départ :

le chaînage unidirectionnel un seul des deux sommets de l'arête de départ est considéré pour parcourir le graphe. Cette dernière est alors une extrémité de la chaîne ainsi construite.

le chaînage bidirectionnel les deux sommets de l'arête de départ sont considérés. Ainsi l'exploration du graphe s'effectue de chaque côté de cette arête.

La figure 5.6 montre deux traits obtenus à partir d'une même arête du graphe de vue, en chaînant respectivement selon le schéma undirectionnel et le schéma bidirectionnel.

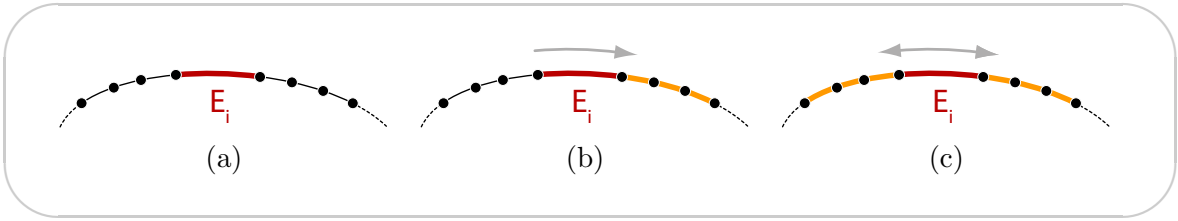


FIG. 5.6 – Opérateurs de chaînage uni- et bi-directionnels

(a) : L'arête E_i du graphe de vue initie le processus de chaînage. La règle d'arrêt qu'on utilisera dans les deux chaînages consiste à s'arrêter aux arêtes cachées (en pointillé). (b) : Le trait (orange) est ici construit à l'aide d'un chaînage unidirectionnel. (c) : Le trait (orange) est ici construit à l'aide d'un chaînage bidirectionnel.

L'opérateur de chaînage bidirectionnel permet de construire les chaînes les plus longues respectant les contraintes fixées (c'est le schéma de chaînage le plus couramment employé). L'opérateur de chaînage unidirectionnel offre un contrôle plus fin sur les arêtes de début de chaînes, ce qui est parfois souhaitable, ces arêtes ayant, en règle générale, été spécifiquement sélectionnées dans cette perspective par un appel antérieur à l'opérateur de sélection.

Grâce à ces deux opérateurs, on peut construire des traits constitués de plusieurs arêtes connexes du graphe de vue.

L'opérateur de chaînage est la clé de voûte d'un module de style. C'est suite à son utilisation que les objets manipulés dans ce module de style ne sont plus des arêtes du graphe mais les traits qui seront à terme tracés dans l'image finale. Le chaînage peut par exemple être utilisé pour construire un unique trait décrivant le contour extérieur d'une scène. Des exemples de chaînage illustrant cet opérateur sont proposés sur la figure 5.7.

Entrées/Sorties

L'entrée des opérateurs de chaînage est un ensemble d'arêtes du graphe de vue. La sortie est un ensemble de traits.

Règles

L'utilisateur doit pouvoir programmer la règle permettant de choisir, à un sommet donné, quelle arête choisir parmi les arêtes candidates. Cette règle est en réalité une fonction (*traverse*) qui prend en argument l'ensemble des arêtes candidates et qui retourne l'arête élue.

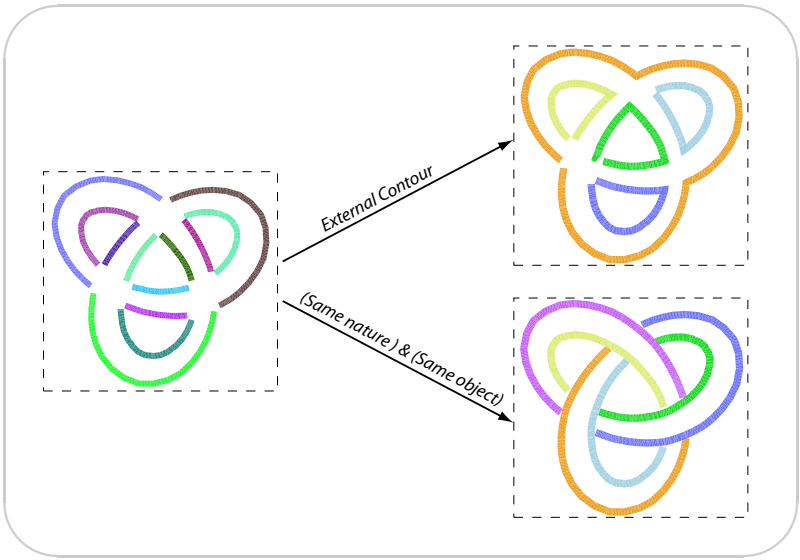


FIG. 5.7 – Opérateur de chaînage

Sur cette illustration, deux ensembles de traits différents on été créés en partant du même ensemble d'arêtes (l'ensemble des arêtes du contour extérieur). **haut :** *La règle de chaînage consiste ici à suivre le contour extérieur. Ainsi, on peut voir que suite à un tel chaînage une chaîne unique relie l'ensemble des arêtes pour chaque partie connexe du contour extérieur.* **bas :** *Pour cette image, le chaînage a suivi les arêtes en respectant la topologie des objets et en reliant les arêtes de même nature. On remarque que, bien que l'ensemble des arêtes de départ soit le même, ce chaînage n'a pas été restreint à la sélection, et à par la même occasion, introduit, dans les chaînes, des arêtes supplémentaires. La règle d'arrêt de chaînage porte ici sur la visibilité : une arête cachée n'est pas valide.*

Remarque : En fonction des contraintes spécifiées, l'ensemble des arêtes candidates fourni par le système à la fonction *traverse* comprendra ou non les arêtes qui sont hors de la sélection et les arêtes déjà chaînées.

On encapsule cette règle sous forme d'un itérateur [GHJV95]. La fonction *traverse* est alors la fonction appelée par les méthodes d'incrémentation et de décrémentation de cet itérateur (pour le chaînage unidirectionnel, seule l'incrémentation est utile). L'utilisation d'un objet itérateur pour effectuer le parcours du graphe présente notamment l'intérêt de pouvoir stocker un historique des arêtes parcourues. On pourra, par exemple, disposer d'un compteur du nombre d'arêtes déjà chaînées.

L'autre règle que l'utilisateur doit spécifier pour le chaînage est la règle d'arrêt de ce chaînage. En effet, l'algorithme ne termine pas forcément sans cette dernière. Cette règle est un prédicat travaillant sur les arêtes du maillage. Par exemple, si l'on désire spécifier un chaînage suivant les arêtes de silhouette visibles, une telle règle peut par exemple consister à s'arrêter à une arête si elle n'est pas visible.

Algorithmes

L'algorithme 5.2 illustre le chaînage unidirectionnel.
L'algorithme 5.3 illustre le chaînage bidirectionnel.

Algorithme 5.2 Opérateur de chaînage unidirectionnel

Fonction *UnidirectionalChain*

Entrée: *suivant* : une fonction d'itération sur les arêtes du graphe de vue, Π : un prédicat unaire d'arrêt opérant sur des éléments 1-dimensionnels, $\mathcal{E}$: un ensemble d'arêtes du graphe de vue

Sortie: $\mathcal{C}$: un ensemble de chaînes, chaque chaîne étant une suite d'arêtes

1. $\mathcal{E} \leftarrow$ l'ensemble actif des arêtes du graphe de vue
2. $\mathcal{C} \leftarrow \{\emptyset\}$
3. **pour** $E_S \in \mathcal{E}$
4. $\quad E \leftarrow E_S$
5. $\quad$ **si** $\Pi(E)$ est vrai ou E déjà dans une chaîne
6. $\quad\quad$ **continuer**
7. $\quad c$: une chaîne
8. $\quad c \leftarrow \emptyset$
9. $\quad$ **répéter**
10. $\quad\quad c \leftarrow c.E$
11. $\quad\quad E \leftarrow suivant(E)$
12. $\quad$ **jusqu'à ce que** $\Pi(E)$ soit vrai
13. $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
14.

Algorithme 5.3 Opérateur de chaînage bidirectionnel

Fonction *BidirectionalChain*

Entrée: *suivant*, *precedent* : deux fonctions d'itération sur les arêtes du graphe de vue, Π : un prédicat unaire d'arrêt opérant sur des éléments 1-dimensionnels, $\mathcal{E}$: un ensemble d'arêtes du graphe de vue

Sortie: $\mathcal{C}$: un ensemble de chaînes, chaque chaîne étant une suite d'arêtes

1. $\mathcal{E} \leftarrow$ l'ensemble actif des arêtes du graphe de vue
2. $\mathcal{C} \leftarrow \{\emptyset\}$
3. **pour** $E_S \in \mathcal{E}$
4. $\quad E \leftarrow E_S$
5. $\quad$ **si** $\Pi(E)$ est vrai ou E déjà dans une chaîne
6. $\quad\quad$ **continuer**
7. $\quad c$: une chaîne
8. $\quad c \leftarrow \emptyset$
9. $\quad$ **répéter**
10. $\quad\quad c \leftarrow c.E$
11. $\quad\quad E \leftarrow suivant(E)$
12. $\quad$ **jusqu'à ce que** $\Pi(E)$ soit vrai
13. $\quad E \leftarrow precedent(E_S)$
14. $\quad$ **tant que** $\Pi(E)$ est faux
15. $\quad\quad c \leftarrow E.c$
16. $\quad\quad E \leftarrow precedent(E)$
17. $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
18.

5.2.4 Raffinement de la topologie des traits : opérateurs de découpage

Principe

L'opérateur de chaînage seul n'offre pas un contrôle suffisant sur la topologie des traits. En effet, l'unité de construction pour les traits est l'arête, c'est-à-dire qu'un trait ne peut être constitué que par un nombre entier d'arêtes du graphe de vue. Or, on veut également pouvoir construire des traits de taille inférieure à une arête. Il faut donc un opérateur de raffinement de la topologie, qui puisse, en particulier, subdiviser un trait en morceaux plus petits. On aura donc également besoin d'un opérateur de découpage permettant ce type de manipulations. On identifie deux stratégies de découpage possibles : un découpage séquentiel et un découpage récursif.

Avec le découpage séquentiel, on parcourt le trait de manière séquentielle et on décide de couper ou non en un point, en y évaluant une condition. Cette condition peut par exemple évaluer si la chaîne est suffisamment longue ou si le point testé est de telle ou telle nature. Ce mécanisme est facile à spécifier mais il prend ses décisions de manière gloutonne et uniquement en se fondant sur des informations locales.

Le découpage récursif, en revanche, prend une décision globale sur la chaîne entière et coupe de manière récursive aux points qui réalisent le minimum d'une fonction donnée.

Avant de discuter davantage ces deux stratégies, il est important de noter qu'on peut vouloir couper les chaînes à des points qui ne correspondent pas nécessairement à des sommets du graphe ou même à des sommets des arêtes caractéristiques initiales. Pour cette raison, le système doit travailler sur une version échantillonée de la courbe, l'échantillonnage étant spécifié par l'utilisateur (dans notre implémentation, des sommets temporaires correspondant à cet échantillonnage sont créés au fur et à mesure que la chaîne est traversée, mais ils ne sont pas stockés de manière permanente, *cf.* section 4.3).

Découpage séquentiel Dans sa version de base, il parcourt le trait et évalue une condition de coupure en chaque point pour décider où couper. Un nouveau trait commence à chaque point de coupure. Afin d'introduire plus de flexibilité dans le choix des extrémités des traits, on raffine la stratégie de découpage en découplant les conditions de coupure pour le début et la fin du trait et en évaluant chacune d'elles dans un parcours différent, les deux parcours s'effectuant en parallèle. Ce processus peut produire un ensemble de traits qui forment une partition du trait initial (lorsque les conditions de début et de fin de trait sont les mêmes), un ensemble de traits se recouvrant partiellement, ou un ensemble de traits isolés. Ainsi défini, cet opérateur permet notamment de générer des dessins dans un style "esquissé" (*sketchy*) en choisissant une configuration pour laquelle les traits se recouvrent. Notons qu'en utilisant l'abscisse curviligne sur le trait dans le prédicat de découpage, il est très facile d'assurer une longueur maximale pour les traits.

Découpage récursif Il a un comportement plus global. Il évalue une fonction pour chaque point d'un trait (la fréquence d'évaluation dépend de l'échantillonnage courant du trait), et coupe au point réalisant le minimum de cette fonction. L'opérateur s'applique alors récursivement aux deux sous traits ainsi créées jusqu'à ce qu'une condition de récursion ne soit plus vérifiée. Par exemple, le découpage récursif est idéal pour couper une chaîne aux points de plus haute courbure.

L'expérience montre qu'il est souvent possible d'obtenir le même résultat en commençant par créer des longues chaînes pour les couper par la suite, ou en utilisant un critère d'arrêt plus agressif pour le chaînage. Différents utilisateurs appréhendent le proces-

sus de dessin de différentes manières et adopterons préférentiellement une stratégie ou l'autre.

La figure 5.8 montre un exemple de découpage récursif s'appuyant sur l'information de courbure.

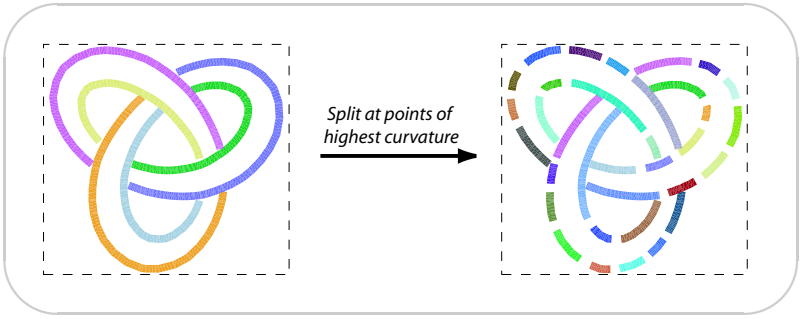


FIG. 5.8 – Opérateur de découpage récursif

Sur cette figure, les chaînes de gauche ont été coupées aux points de haute courbure pour produire les traits de droite. La fonction évaluée est donc la fonction 0-dimensionnelle d'évaluation de la courbure 2D en un point. La récursion s'arrête dès que la taille des chaînes est inférieure à un seuil donné. Ce type de découpage est notamment utilisé pour produire des lignes de construction, telles que celles illustrées par la figure 5.4 (b). En effet, le shader *de lignes de construction se contente de déformer les traits de manière à les projeter sur la tangente en leur milieu. Il est donc fortement dépendant de la topologie (et de la longueur) des traits qui lui sont donnés. Le découpage des traits aux points de haute courbure permet de disposer des traits idéaux pour ce type de déformation spatiale.*

Entrées/Sorties

Pour les deux opérateurs de découpage, les entrées et les sorties sont des traits. Si $\{T_i\}_i$ et $\{T_j^i\}_{i,j}$ désignent respectivement l'ensemble des traits considérés en entrée et l'ensemble des traits produits en sortie de l'opérateur de découpage, ce dernier peut être vu comme une relation s qui à un trait en entrée associe une série de traits en sortie :

$$\begin{array}{rccc} s: & \{T_i\}_i & \longrightarrow & \left\{\{T_j^i\}_j\right\}_i \\ & T_i & \longmapsto & \{T_j^i\}_j \end{array}$$

Pour le découpage séquentiel travaillant avec deux conditions de coupure identiques et le découpage récursif, les $\{T_j^i\}_j$ forment une partition de chaque T_i. En revanche, pour le découpage séquentiel travaillant avec deux conditions de coupure différentes, les $\{T_j^i\}_j$, soit se recouvrent, soit sont disjoints.

Règles

Pour les deux opérateurs de découpage, la période d'échantillonnage utilisée pour parcourir les points d'un trait est un paramètre spécifié par l'utilisateur. Les autres règles varient entre les deux types d'opérateurs et sont étudiées indépendamment.

Découpage séquentiel Le contrôle sur cet opérateur se fait à l'aide de deux prédicats unaires travaillant sur des points. Le premier positionne les points de début de chaîne tandis que le deuxième positionne les points de fin de chaîne.

Découpage récursif On identifie trois règles pour cet opérateur. La première est la fonction évaluée par l'algorithme de découpage récursif dont le minimum identifie le point de coupure. La deuxième est le prédicat de récursion. C'est un prédicat unaire travaillant sur des traits. Si ce prédicat n'est pas vérifié sur un trait candidat à être coupé en deux, la récursion s'arrête. La troisième règle est un prédicat unaire, travaillant sur des points, qui permet de faire une présélection des points candidats à être des lieux de coupure. Il est par exemple fréquent de vouloir prévenir une coupure si le point vérifiant le minimum de la fonction se trouve proche d'une extrémité du trait. Cette condition peut être spécifiée à l'aide d'un prédicat ne sélectionnant comme candidats à la coupure, que les points dont l'abscisse curviligne est comprise entre une valeur minium et une valeur maximum.

Algorithmes

L'algorithme pour le découpage séquentiel est illustré par l'algorithme 5.4.

Algorithme 5.4 Opérateur de découpage séquentiel

Fonction *SequentialSplit*
Entrée: Π_{start}, Π_{stop} : les prédicats unaires 1D de début et de fin de chaîne, $\mathcal{T}$: un ensemble de traits
Sortie: $\mathcal{T}'$: un ensemble de traits

$\mathcal{T} \leftarrow$ l'ensemble actif des traits
$\mathcal{T}' \leftarrow \{\emptyset\}$
pour $T \in \mathcal{T}$
    P_{start} : itérateur sur les points de T
    P_{stop} : itérateur sur les points de T
    $P_{start} \leftarrow$ premier point de T
    répéter
        $P_{stop} \leftarrow suivant(P_{start})$
        T' : un trait
        $T' \leftarrow \{P_{start}\}$
        répéter
            $T' \leftarrow T'.P_{stop}$
            $P_{stop} \leftarrow suivant(P_{stop})$
        jusqu'à ce que $\Pi_{stop}(P_{stop})$ soit vrai
        $T' \leftarrow T'.P_{stop}$
        $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{T'\}$
    (∗ on cherche le prochain point de début de chaîne ∗)
        répéter
            $P_{start} \leftarrow suivant(P_{start})$
        jusqu'à ce que $\Pi_{start}(P_{start})$ soit vrai
    jusqu'à ce que P_{start} = dernier point de T

Pour le découpage récursif, on définit la fonction récursive *RecursiveSplitChain*, qui coupe une seule chaîne, passée en argument, de manière récursive (*cf.* algorithme 5.5).

Algorithme 5.5 fonction récursive de découpage d'un trait

Fonction *RecursiveSplitChain*
Entrée: T : un trait, f : fonction de l'ensemble des points de $\mathcal{G}$ dans un ensemble $\mathbb{E}$, Π^{1D}_{stop} : prédicat unaire 1D d'arrêt de récursion, Π^{0D}_{select} : prédicat unaire 0D de préselection des points de coupure candidats
Sortie: $\mathcal{T}'$: un ensemble de traits

```
1.  si Π_stop(T) est vrai
2.          T' ← T' ∪ {T}
3.          retourner
4.
(∗ Recherche du point réalisant le minimum de f ∗)
5.  P_split : point de T
6.  P_split ← nil
7.  m ← max(E)
8.  pour P_i ∈ {P_0, ..., P_n}, points échantillonés sur T
9.          si Π^0D_select(P_i) est faux
10.                 continuer
11.         si f(P_i) < m
12.                 m ← f(P_i)
13.                 P_split ← P_i
14. si P_split = nil
15.         T' ← T' ∪ {T}
16.         retourner
17.
(∗ Un point de coupure existe, on construit les deux sous chaînes ∗)
18. T'_1, T'_2 : deux traits
19. T'_1 ← {P_0, ..., P_split}
20. T'_2 ← {P_split, ..., P_n}
21. T' ← T' ∪ {T'_1} ∪ {T'_2}
22. RecursiveSplit(T'_1, f, Π^1D_stop, Π^0D_select)
23. RecursiveSplit(T'_2, f, Π^1D_stop, Π^0D_select)
```

Cette fonction est appelée successivement sur tous les éléments 1D actifs au sein de l'opérateur de découpage récursif (*cf.* algorithme 5.6).

5.2.5 Contrôle de l'ordre de traitement des éléments 1D : opérateur de tri

Principe

Dans notre approche l'ordre dans lequel les arêtes ou les traits sont traités peut influencer le dessin. L'opérateur de chaînage par exemple, qui inclut un mécanisme permettant d'empêcher la réutilisation d'une arête, produira des ensembles de chaînes différents si l'ordre dans lequel les arêtes lui arrivent, change. De même, lorsque l'information de densité est utilisée pour éviter l'encombrement de traits, il est essentiel de tracer en premier les traits qui

Algorithme 5.6 Opérateur de découpage récursif

Fonction *RecursiveSplit*

Entrée: f : fonction de l'ensemble des points de $\mathcal{G}$ dans un ensemble $\mathbb{E}$, Π^{1D}_{stop} : prédicat unaire 1D d'arrêt de récursion, Π^{0D}_{select} : prédicat unaire 0D de présélection des points de coupure candidats, $\mathcal{T}$: un ensemble de traits

Sortie: $\mathcal{T}'$: un ensemble de traits

1. $\mathcal{T} \leftarrow$ l'ensemble actif des traits
2. $\mathcal{T}' \leftarrow \{\emptyset\}$
3. **pour** $T \in \mathcal{T}$
4. $\quad RecursiveSplitChain(T)$

sont visuellement plus importants, de manière à ce qu'ils aient le moins de chance d'être omis (le chapitre 6 expose plus en détails l'omission de lignes basée sur la densité et l'utilisation de l'opérateur de tri dans ce contexte).

En réalité, pour tous les opérateurs agissant selon un algorithme "glouton", l'ordre dans lequel les éléments sont traités peut être important. Ainsi, nous fournissons un opérateur de tri utilisable sur n'importe quelle séquence d'éléments 1D.

Entrées/Sorties

L'entrée de cet opérateur est une séquence d'éléments 1D et la sortie et une séquence triée contenant les mêmes éléments 1D.

Règle

La règle à spécifier est ici un prédicat binaire de comparaison permettant de déterminer le plus prioritaire de deux éléments 1D. Ce prédicat peut s'appuyer sur l'ensemble des informations défini au chapitre précédent.

La définition d'un ordre pertinent pour les arêtes du graphe de vue ou les traits peut être très difficile à trouver et requiert l'évaluation et l'intégration de nombreuses informations de types différents, qui peuvent être spécifiés uniquement à l'aide d'une approche programmable.

Algorithme

N'importe quel algorithme de tri peut être utilisé pour implémenter l'opérateur de tri. Cet algorithme doit utiliser le prédicat binaire de comparaison spécifiée comme règle, pour comparer deux éléments de la séquence.

Dans notre implémentation, nous utilisons le tri introspectif [Mus97] fourni par la *Standard Template Library* [SLM00].

5.2.6 Opérateur de dessin

Finalement, une fois les traits construits et leurs attributs visuels spécifiés, un opérateur de dessin doit permettre de tracer ces traits.

Cet opérateur doit être mis à disposition de manière explicite (par opposition à un système où le dessin serait automatiquement tracé à la fin d'un module de style, sans appel explicite) puisqu'il peut être couplé avec des appels à d'autres opérateurs. Par exemple, si l'on souhaite

omettre des traits de manière à garder une faible densité de traits dans l'image finale, il faut coupler un appel à l'opérateur de sélection avec l'appel à l'opérateur de dessin. Le prédicat, pour cet opérateur de sélection, doit alors s'appuyer sur l'information de densité qui est mis à jour au fur et à mesure que les traits sont dessinés. (Les mécanismes mis en jeu pour mettre en œuvre un tel couplage sont discutés à la section suivante.)

L'opérateur de dessin est donc l'initiateur du processus de rendu de chaque trait. Les techniques utilisées pour ce rendu sont présentées à la section 5.4.5.

Entrées/sorties

Les entrées de cet opérateur sont des traits et la sortie est l'affichage de ces traits. Cet opérateur se branche directement sur le composant de rendu de marques en charge de l'affichage. Ce composant de rendu peut ensuite générer n'importe quel type de sortie, allant d'une écriture dans le *framebuffer* à une sortie textuelle.

C'est donc cette base d'opérateurs qui est mise à la disposition de l'utilisateur pour lui permettre de faire de la description de style. Bien qu'il soit difficile d'évaluer la qualité d'une telle base, le chapitre 8 propose une discussion autour du choix de ces opérateurs et s'attache, tout du moins, à poser les questions nécessaires.

Résumé: Nous avons identifié un ensemble d'opérateurs permettant de formuler un style : la sélection, le chaînage, le découpage, le *shading*, le dessin et le tri. Chacun de ces opérateurs est contrôlé par des règles, dont la programmation revient à l'utilisateur.

5.3 Pipeline d'opérateurs

Les opérateurs que nous venons de définir seront appelés de manière séquentielle par l'utilisateur, dans un module de style. Certains de ces opérateurs ne peuvent être appelés à la suite d'autres (par exemple, on ne peut pas faire deux appels à l'opérateur de chaînage, pour des raisons d'incompatibilité d'entréées/sorties) tandis que, pour d'autres, l'appel peut avoir lieu à tout moment du module de style. En imposant certaines contraintes et en offrant certaines libertés dans l'organisation de leurs appels, ces opérateurs suggèrent une méthode d'utilisation, et même une méthodologie pour la modélisation de style.

Dans cette section, nous présentons une organisation de ces opérateurs sous forme de pipeline. En outre, nous accompagnons cette organisation d'un schéma de synchronisation des données. C'est-à-dire que nous proposons une manière d'ordonnancer l'entrée des arêtes du graphe de vue dans le pipeline et leur progression (éventuellement après transformation en traits) d'opérateur à opérateur. Comme nous allons le voir, les différents choix de synchronisation ont une influence sur les informations disponibles à chaque étage du pipeline.

L'entrée d'un module de style, et donc de ce pipeline, est toujours l'ensemble des arêtes du graphe de vue. La sortie du pipeline est toujours un calque de l'image finale.

Remarque : Nous employons ici, et dans la suite, le terme “pipeline”, uniquement pour désigner la mise bout à bout d'opérateurs. En particulier, nous supposons ici un mode de fonctionnement purement séquentiel, c'est-à-dire qu'à un instant t, un seul opérateur traite un seul élément.

5.3.1 Propriétés des opérateurs

Nous commençons par énoncer un certain nombre de propriétés générales sur les opérateurs qui permettent d'aider à déterminer les positions relatives des différents opérateurs dans le pipeline du module de style et ainsi à en dessiner les grandes lignes.

Entrées/sorties Les types des entrées et des sorties des différents opérateurs fixent en partie la disposition de certains éléments au sein d'un pipeline. En effet, pour pouvoir mettre bout à bout deux opérateurs, le type des sorties du premier doit correspondre au type des entrées du deuxième.

Fixité Nous remarquons que certains opérateurs peuvent intervenir à plusieurs niveaux du processus de dessin, en particulier parce qu'ils acceptent plusieurs types d'entrées différents (l'opérateur de sélection, par exemple, peut être utilisé en début de pipeline sur les arêtes du graphe de vue ou, plus en aval, sur les traits) tandis que d'autres y occupent une place fixe (l'opérateur de dessin, par exemple, doit nécessairement se trouver en bout de pipeline). On distingue donc les opérateurs *flottants*, qui comprennent la sélection, le tri et le *shading*, des opérateurs *fixes*, incluant le chaînage, le découpage, le dessin.

> **Remarque :** La mobilité de l'opérateur de *shading* est en réalité relativement limitée. En effet, les traits étant les seuls éléments à pouvoir être traités par le *shading*, ce dernier est un opérateur flottant mais contraint à rester entre le chaînage et le dessin.

Optionalité La présence de certains opérateurs dans un module de style est obligatoire et peut donc servir de base structurante au pipeline. Les deux seuls opérateurs obligatoires sont le chaînage, qui est nécessaire au passage des arêtes aux traits, et l'opérateur de dessin. Des appels à tous les autres opérateurs sont optionnels.

Unicité On dit d'un opérateur qu'il est *unique* s'il ne peut apparaître qu'une seule fois dans tout le pipeline (dans le cas inverse, l'opérateur est *multiple*). Les seuls opérateurs uniques sont le chaînage (qui marque le passage d'un ensemble actif d'arêtes du graphe de vue à un ensemble actif de traits, prévenant du même coup tout autre chaînage), et l'opérateur de dessin.

5.3.2 Synchronisation des données

Nous nous intéressons maintenant au type de synchronisation à employer pour la circulation des données (c'est-à-dire des arêtes du graphe de vue, des traits, *etc.*...) dans le pipeline. Comme nous allons le voir, cette synchronisation décide également en partie du placement des opérateurs dans le pipeline.

Supposons que l'on travaille avec quatre opérateurs O_1, O_2, O_3 et O_4 et que ces opérateurs puissent être appelés séquentiellement dans cet ordre (c'est-à-dire que les sorties de O_i correspondent aux entrées de O_{i+1}). Soit $\{l_1, \ldots, l_n\}$ l'ensemble des éléments (*e.g.* arêtes) passé en entrée du pipeline. La figure 5.9 illustre ce pipeline.

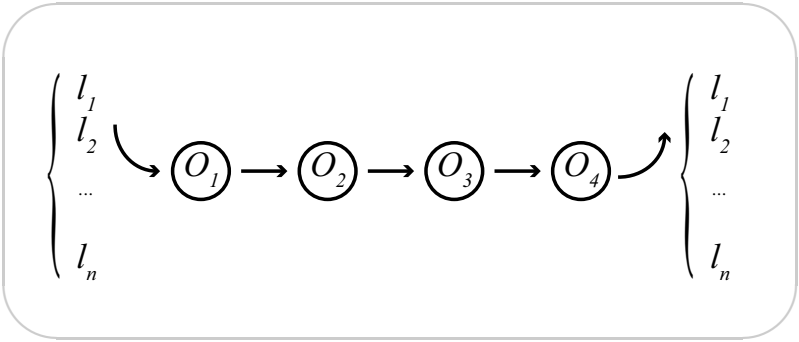


FIG. 5.9 – Synchronisation du pipeline d'opérateurs

L'étude de la question de synchronisation se fait sur ce pipeline théorique. n éléments l_i sont traités par les opérations $O_1, O_2, O_3 et O_4$. L'objet de cette étude est de déterminer l'ordonnancement des éléments au travers de la séquence d'opérateurs.

Dans la mesure où nous nous plaçons dans un schéma de fonctionnement séquentiel, à un instant t, un seul élément l_i est traité par un seul opérateur O_j. À ce même instant t, les éléments $\{l_k\}_k$ auront déjà été traités par les opérateurs $\{O_p\}_p$. Ce sont ces ensembles d'indices respectivement parcourus par k et p qui changent avec le mode de synchronisation choisi. Et c'est par conséquent l'ensemble des informations disponible lors du traitement de l_i par O_j qui change également, puisque chaque opération $O_p(l_k)$ rajoute une information différente (par exemple, si O_p est l'opérateur de sélection, le fait d'avoir appliqué O_p à l_k nous permet de savoir si l_k fait partie de la sélection ou pas). On décidera donc d'un schéma de synchronisation plutôt que d'un autre en fonction des informations qu'on veut pouvoir utiliser à un instant t, lors du traitement d'un élément l_i par un opérateur O_j.

Remarque : Dans notre présentation du pipeline, nous utilisons la notation l_i pour désigner un élément 1D, tout au long du pipeline. Il s'agit d'un abus de notation dans la mesure où, initialement, ces l_i correspondent à des arêtes du graphe de vue, alors qu'après un appel à l'opérateur de chaînage (et dans toute la suite du pipeline), ils correspondent aux traits ainsi construits. Ce raccourci nous a toutefois semblé améliorer la lisibilité du discours.

En supposant que l'on travaille de manière purement séquentielle, on distingue deux manières de synchroniser le flot de données au travers du pipeline, une synchronisation *par élément* et une synchronisation *par opération*.

Synchronisation *par élément*

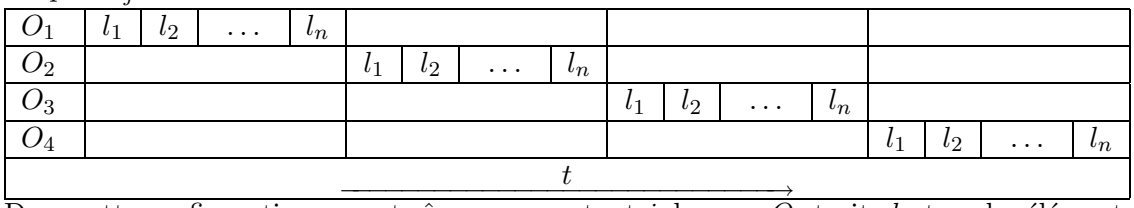
Dans ce schéma de synchronisation, l'élément l_i passe dans l'étage O_j directement après avoir été traité par O_{j-1} et l'élément l_{i+1} est injecté dans le pipeline dès que l_i en sort. En s'inspirant des diagrammes de Gantt, ce schéma de synchronisation se représente ainsi :

O_1	l_1			l_2			...	l_n						
O_2		l_1			l_2		...		l_n					
O_3			l_1			l_2	...			l_n				
O_4				l_1			l_2	...			l_n			
t ⟶														

Dans cette configuration, on est sûr que lorsque l_i entre dans le pipeline, tous les $\{l_k\}_{k<i}$ ont déjà traversé l'ensemble du pipeline. Donc, pour tout j, O_j, lorsqu'il traite l_i, peut en particulier utiliser les informations fournies par les traitements $\{O_p(l_k)\}_{p,k<i}$ s'il le souhaite.

Synchronisation par opérations

Dans ce schéma de synchronisation, les éléments sont tous traités par un étage avant d'être transmis à l'étage suivant. Ainsi, O_j traite l'élément l_0 dès que l_n a été traité par O_{j-1}. Les éléments sont en quelque sorte stockés entre deux étages O_j et O_{j+1} jusqu'à avoir tous été traités par O_j.



Dans cette configuration, on est sûr que, pour tout j, lorsque O_j traite l_i, tous les éléments $\{l_0, \ldots, l_n\}$ ont déjà été traités par l'ensemble des $\{O_p\}_{p<j}$. En particulier, lors du traitement de l_i par O_j, on peut utiliser l'ensemble des informations fourni par les traitements$\{O_p(l_k)\}_{p<i,k}$.

Quel schéma de synchronisation choisir ?

Le choix de l'un ou l'autre des schémas a des répercussions sur les possibilités offertes par le pipeline.

La synchronisation par opérations, on l'a vu, permet de disposer au sein d'un opérateur O_j de l'information fournie par le passage de l'ensemble des éléments par les opérateurs précédents. Par exemple, si on considère un pipeline comprenant un opérateur de sélection suivi d'un opérateur de chaînage, cette synchronisation permet de savoir, pendant le chaînage, si une arête quelconque du graphe de vue a été sélectionnée ou non. Cette information est nécessaire à l'implémentation d'un algorithme de chaînage permettant à l'utilisateur de contraindre les chaînes à rester dans la sélection. Ce type de synchronisation est, par ailleurs, incontournable dès que les décisions à prendre impliquent de travailler avec plusieurs éléments de l'ensemble considéré. En particulier, un opérateur tel que le tri manipule tous les éléments à la fois et ne peut fonctionner que sur ce mode de synchronisation.

En revanche, les opérateurs fonctionnant sur un mode de synchronisation par opérations ne peuvent bénéficier d'aucune information résultant d'une traversée complète du pipeline par un élément. Par exemple, toute information relative au dessin en cours n'est pas disponible pour les opérateurs se situant en amont de l'opérateur de dessin dans le pipeline. La synchronisation par élément en revanche, offre cette possibilité de disposer de l'information issue du passage des éléments précédant l'élément courant, par l'ensemble du pipeline. Tout opérateur voulant accéder à des informations provenant du dessin en cours doit fonctionner sur ce mode.

En pratique, on veut pouvoir fonctionner selon le premier schéma pour certaines opérations et selon le deuxième pour d'autres. Ces deux modes cohabiteront donc dans le pipeline, qui sera constitué de sous-pipelines fonctionnant chacun sur leur mode de synchronisation propre.

Les informations les plus utiles dont on bénéficie en synchronisation par éléments, sont les informations liées au dessin en cours. C'est donc sur ce critère que l'on évaluera le besoin d'un opérateur à fonctionner dans ce mode. L'opérateur de dessin par exemple, est justement celui qui met à jour l'image finale et doit, par conséquent, nécessairement fonctionner ainsi,

pour permettre à d'autres opérateurs de bénéficier d'informations sur le dessin en cours. On se pose maintenant la question de savoir, parmi les autres opérateurs, quels sont ceux qui requièrent un accès à l'information du dessin. Il est important de noter que ce choix n'a pas les mêmes répercussions pour tous les opérateurs. Pour un opérateur *fixe*, pouvant à priori bénéficier des informations disponibles dans un mode ou l'autre, choisir l'un des deux modes de synchronisation revient à renoncer à l'autre, et donc à l'accès à l'un des deux types d'informations. Un opérateur *flottant*, en revanche, pourra être appelé dans un sous-pipeline ou l'autre, en fonction des besoins, et un opérateur *flottant* et *multiple* pourra être appelé dans plusieurs sous-pipelines différents et ainsi utiliser les deux types de synchronisation dans le même module de style.

Opérateur de dessin Comme nous venons de le dire, l'opérateur de dessin doit nécessairement fonctionner en mode de synchronisation par élément. Cet opérateur étant *unique*, il fonctionnera uniquement sur ce mode. Par ailleurs, il s'agit d'un opérateur *fixe*, qui occupe le dernier étage du pipeline. On sait donc déjà que le dernier sous-pipeline fonctionnera en synchronisation par éléments. Ceci a pour conséquence que tous les opérateurs qui voudront pouvoir utiliser l'information provenant du dessin en cours devront appartenir au même sous-pipeline et fonctionner sur le même mode de synchronisation.

Opérateur de tri L'entrée de l'opérateur de tri est, en réalité, la séquence des éléments prise comme un tout, et non pas la séquence des éléments pris indépendamment les uns des autres. En effet, cet opérateur a besoin d'accéder à tous les éléments de la séquence afin de pouvoir produire le résultat. Il fonctionne donc nécessairement en synchronisation par opération.

Opérateur de sélection Pour la sélection, les deux modes de synchronisation peuvent être envisagés. Par exemple, pour pouvoir implémenter un style pour lequel les traits sont tracés uniquement si la densité du dessin en cours est suffisamment faible, il faut utiliser un opérateur de sélection s'appuyant sur l'information du dessin, et donc synchronisé par élément. De l'autre côté, pour qu'un opérateur tel que le chaînage, qui, en partant d'un élément (l'arête de départ) est amené à en considérer d'autres (les autres arêtes de la chaîne), puisse se restreindre aux arêtes de la sélection (et donc, en particulier ne pas les inclure dans les arêtes secondaires de la chaîne), il faut que tous les éléments soient passés par l'étage de sélection avant de poursuivre leur parcours, c'est-à-dire que cet opérateur doit fonctionner selon le schéma de synchronisation par opération. L'opérateur de sélection étant à la fois *flottant* et *multiple*, on pourra en envisager une version synchronisée par élément dans le sous-pipeline final, et d'autres versions synchronisées par opérations plus en amont.

Opérateur de chaînage Pour l'opérateur de chaînage, les choses se compliquent. En effet, il est la clé de voûte du module de style et il est à la fois *unique* et *fixe*. Il semble raisonnable de vouloir insérer cet opérateur dans le sous-pipeline contenant l'opérateur de dessin de manière à pouvoir prendre des décisions de chaînage s'appuyant sur les informations du dessin courant (typiquement, on pourrait vouloir stopper une chaîne à une arête si celle-ci se situe dans une zone déjà trop dense). Le problème qui se pose alors est qu'en étant dans le dernier sous-pipeline, l'opérateur de chaînage ne peut précéder aucun opérateur fonctionnant selon un schéma de synchronisation par opération, tel que le tri par exemple. C'est-à-dire que les seuls éléments qui seraient alors susceptibles d'être triés seraient les arêtes du graphe de vue, ce qui représente une limitation sévère. Les limitations résultant de l'absence de l'opérateur de chaînage dans le sous-pipeline de

dessin nous ont, en revanche, paru moins importantes, et nous avons finalement choisi la synchronisation par opération pour le chaînage.

Opérateur de découpage Le découpage pourrait fonctionner selon l'un ou l'autre des schémas de synchronisation. En effet, d'un côté, il semble raisonnable d'imaginer une synchronisation par élément (et donc un appartenance au sous-pipeline de dessin), pour, par exemple, permettre l'utilisation de l'information provenant du dessin en cours, dans les décisions de coupure (par exemple, afin de couper les traits en cas de changement abrupte de densité). De l'autre côté, on pourrait souhaiter adopter le mode de synchronisation par opération, ne serait-ce que pour pouvoir faire suivre un appel à cet opérateur de découpage d'un tri. Dans notre implémentation, nous avons opté pour une synchronisation par opération pour l'opérateur de découpage.

> **Remarque :** L'opérateur de découpage étant multiple, nous aurions pu également en prévoir des appels dans le sous-pipeline de dessin. Pour une question de temps, nous n'avons malheureusement pas pu implémenter cette fonctionnalité.

Opérateur de *shading* Il est clairement intéressant pour un utilisateur de pouvoir spécifier des attributs en fonction d'informations provenant du dessin en cours (*e.g.* une épaisseur de trait d'autant plus faible que la densité de l'image est importante). Nous avons donc choisi d'appliquer le mode de synchronisation par élément à cet opérateur.

> **Remarque :** Il existe pourtant également un intérêt à pouvoir entremêler des appels à l'opérateur de *shading* et des appels à l'opérateur de découpage, et donc à prévoir également une synchronisation par opérateur pour le *shading* : cet intérêt consiste dans la possibilité de découper un trait auquel on a déjà affecté des attributs et de pouvoir à nouveau lui affecter d'autres attributs après ce découpage. On peut de cette manière simuler de manière efficace une paramétrisation multiple des traits (cet aspect est discuté davantage dans la section 5.4.4). Ici encore, nous n'avons pas disposé de suffisamment de temps pour implémenter cette fonctionnalité, et le *shading* ne fonctionne actuellement que dans sa version synchronisée par élément.

> **Remarque :** Les séquences d'éléments (arêtes ou traits) ne sont jamais explicitement manipulées au sein d'un module de style. À un instant donné, en fonction des opérateurs ayant déjà été appelés, on a une séquence active d'éléments 1D. Initialement, la séquence active contient l'ensemble des arêtes du graphe de vue. Après une sélection, elle contient le sous-ensemble des arêtes respectant la contrainte fixée et après le chaînage, elle est composée des traits construits. C'est cette séquence active qui est implicitement manipulée dans le module de style. Le chapitre 7 montre des exemples de code source de modules de styles sur lesquels on peut voir les données manipulées par l'utilisateur dans une description de style.

5.3.3 Proposition de pipeline

Compte tenu des remarques faites dans les sections précédentes, nous proposons ici une organisation en pipeline des opérateurs. Le pipeline complet Φ est composé de deux sous-pipelines Φ_{op} et Φ_{el}, respectivement synchronisés par opération et par élément.

$$
\begin{aligned}
\Phi &= \Phi_{op}\Phi_{el} \\
\Phi_{op} &= \Big([tri][selection]^*\Big)^* \; chainage \; [selection]^*[decoupage]^*\Big([tri][selection]^*\Big)^* \\
\Phi_{el} &= [selection]^*[shade]^* dessin
\end{aligned}
$$

La figure 5.10 illustre ce pipeline sur trois exemples de module de style.

Ainsi, le squelette d'un module de style est toujours constitué d'appels séquentiels aux différents opérateurs fournis, dans un ordre qui respecte celui proposé ici. La figure 5.11 donne un aperçu de la structure du code des modules de style. Le code exhaustif de ces trois modules de style est donné section 7.2.

Résumé: Nous avons proposé un pipeline pour organiser les appels aux opérateurs au sein d'un module de style et remarqué que deux modes de synchronisation pouvaient être envisagés dans ce pipeline : un mode de synchronisation *par élément* et un mode de synchronisation *par opérateur*. En pratique, ces deux modes cohabitent dans deux sous parties différentes du pipeline.

5.4 Implémentation

Dans cette section, nous discutons un certain nombre d'aspects liés à l'implémentation des opérateurs.

5.4.1 Langage

Nous avons présenté un ensemble d'opérateurs ainsi que plusieurs structures destinés à être manipulés par l'utilisateur, dans un environnement de programmation, pour modéliser un style. Nous avons donc, jusqu'ici, spécifié une interface de programmation, sans préciser dans quel langage elle pouvait être proposée. Toutefois, un certain nombre de considérations limitent les possibilités quant aux langages possibles. Premièrement, la majorité des concepts que nous utilisons relève de la programmation objet, ce qui suppose donc que notre langage de description de style doit lui aussi être un langage objet. Ensuite, ce langage doit être suffisamment riche pour offrir une flexibilité suffisante à l'utilisateur dans la programmation. Finalement, afin de pouvoir visualiser interactivement les résultats de modifications faites dans un module de style, il faut que ce dernier soit interprété ou compilé à la volée.

Nous avons choisi d'utiliser le langage Python, qui respecte les contraintes fixées, pour programmer les modules de style. Ainsi, notre interface est fournie sous la forme d'une librairie Python.

En pratique, le système avec lequel nous avons expérimenté notre approche et produit les résultats de cette thèse se décompose en deux parties : l'application hôte et la description de style. L'application hôte se charge de la construction du graphe de vue à partir d'une scène 3D, et d'afficher les traits construits par la feuille de style. Elle intègre un interpréteur Python qui lui permet d'exécuter le code des modules de style. L'application hôte et la feuille de style s'appuient tous deux sur la librairie de descrption de style. Pour le premier, cette librairie

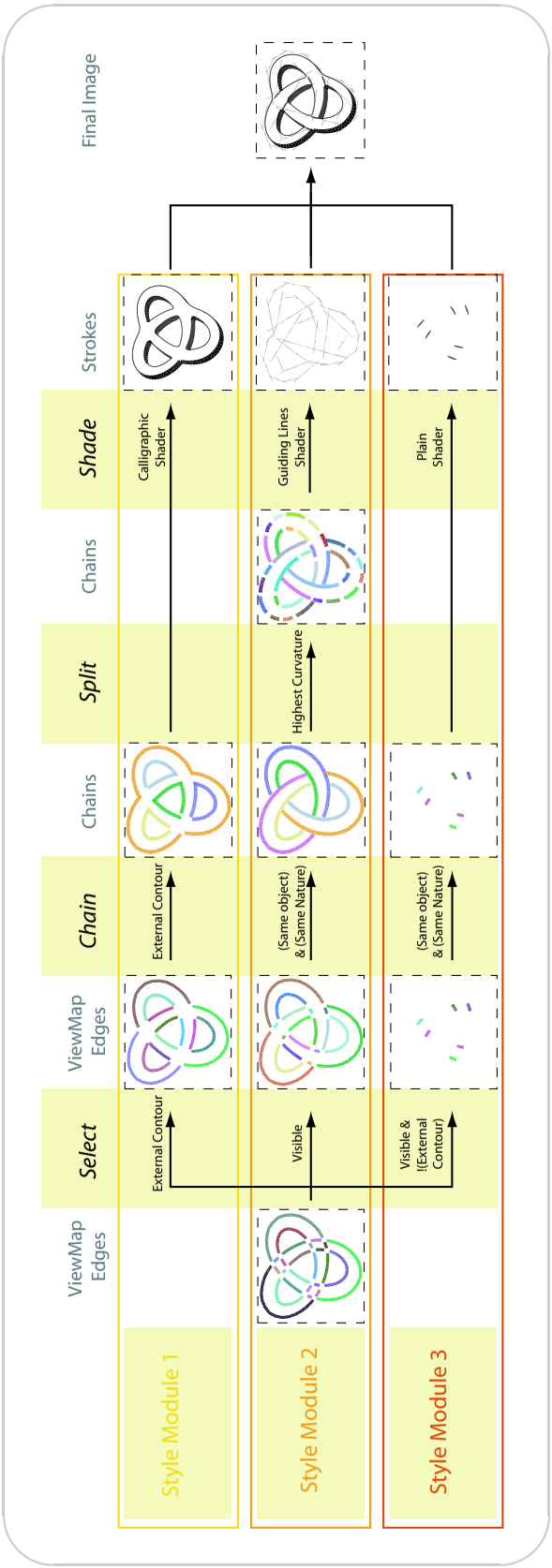


FIG. 5.10 – Exemples de trois modules de style

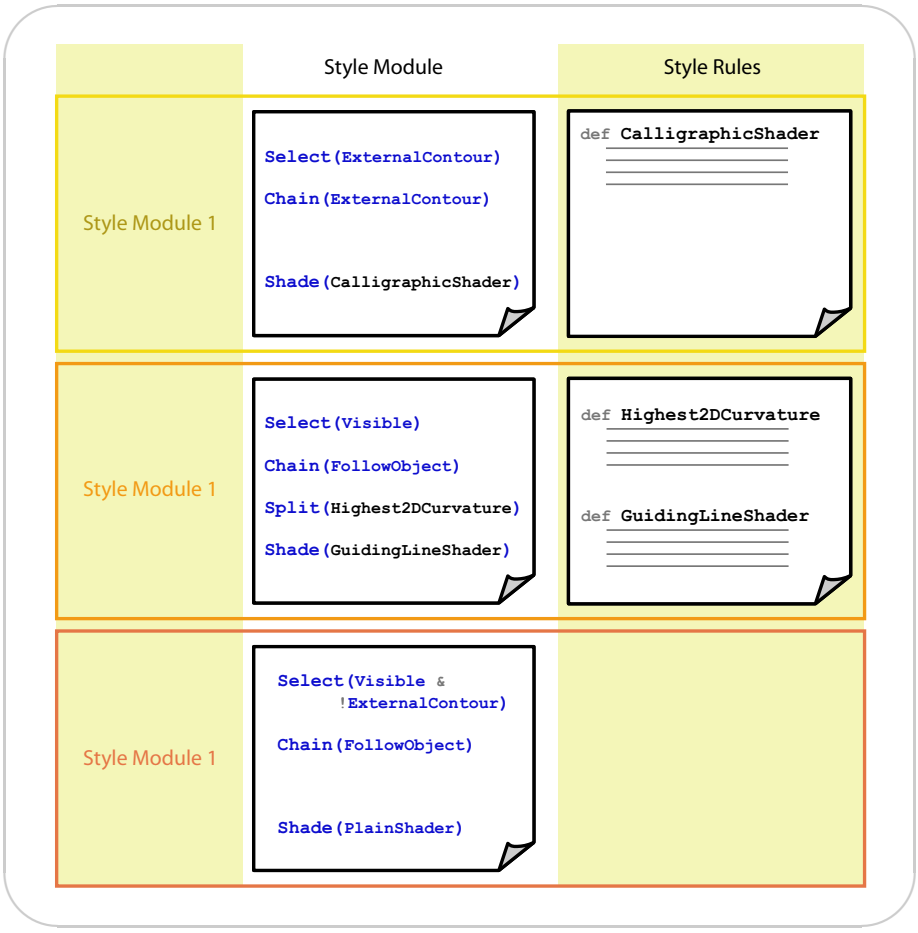


FIG. 5.11 – Exemples de pipelines de trois modules de style (2)

Cette figure montre les fichiers constitutifs de chacun des trois modules de style ainsi que la structure grossière de ces fichiers. Ainsi, chaque module de style est constitué d'un fichier contenant la séquence des appels aux opérateurs de description de style et qui est le module de style à proprement parler et d'un fichier contenant la définition des règles utilisées par le module de style. Cette séparation d'un module de style en plusieurs fichiers n'est pas obligatoire et sert ici uniquement un but pédagogique.

apporte la spécification des structures utilisées, telles que le graphe de vue et les traits. Pour le deuxième, c'est l'ensemble des opérateurs et des mécanismes d'accès à l'information qui sont utilisés. En outre, le partage des données (graphe de vue et traits), entre l'application hôte et l'interpréteur de style se fait également au travers de la librairie.

Nous avons choisi d'utiliser deux langages différents pour développer, d'un côté l'application de visualisation et la librairie de description de style et, de l'autre, pour les modules de style. Python est, comme on l'a vu, utilisé pour les derniers, tandis que, les premiers sont implémentés en C++. Ce choix permet, d'une part, de répondre aux exigences de performance d'opérations telles que la visualisation, et, d'autre part, de séparer clairement la programmation des modules de style des aspects techniques bas niveau du système. Ainsi, la librairie de description de style est, de manière native, en C++ (et est utilisée sous cette forme par l'application hôte) et a été interfacée pour Python (c'est cette interface qui est utilisée par

les modules de style). La figure 5.12 illustre les grandes lignes du fonctionnement de notre système. Nous détaillons dans la suite quelques uns des mécanismes d'implémentation liés au couplage de ces deux langages.

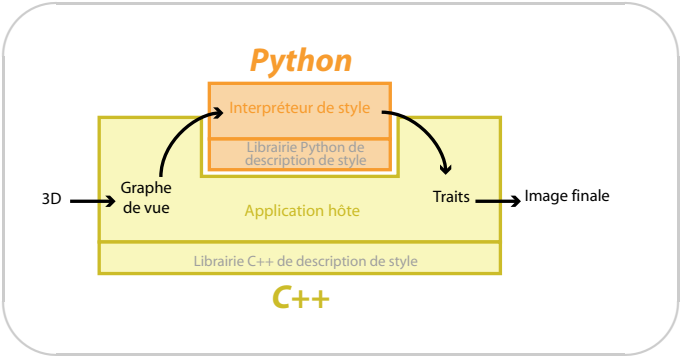


Fig. 5.12 – Implémentation du système

On distingue, d'un côté l'application hôte, qui se charge de calculer le graphe de vue à partir de la 3D et d'afficher les traits, et de l'autre, l'interpréteur de style, qui crée les traits à partir de la description Python du style. L'application hôte et la description de style d'appuient sur la même spécification d'interface (e.g. ensemble des opérateurs) mais en utilisent respectivement une mise en œuvre en C++ *et en* Python.

L'utilisation de deux langages différents n'est possible que s'il existe une interface suffisamment évoluée permettant de passer des objets, voire des modèles (de classes par exemple) de chaque environnement vers l'autre.

Si L_1 et L_2 désignent les langages objets utilisés respectivement pour le développement de l'application et d'un module de style, les contraintes minimales sont les suivantes :

Instanciation et utilisation dans L_2 d'objets définis dans L_1 Les structures de données définies dans L_1 doivent pouvoir être instanciées et manipulées dans L_2. Par exemple, les *functors* d'accès aux informations sont définis en C++ et doivent pouvoir être instanciés en Python. Cela implique que tous les mécanismes d'héritage ou de surcharge d'opérateurs existant dans L_1 soient gérés par L_2. De même les fonctions définies dans L_1 doivent pouvoir être appelées dans L_2. Ainsi, dans une procédure écrite en L_2, on pourra instancier un objet défini en L_1 et le passer comme argument à une fonction également définie en L_1.

Héritage dans L_2 de classes définies dans L_1 Si une classe a été définie dans L_1, on veut pouvoir l'hériter dans L_2. Par exemple, nous définissons en C++ une classe de base dont héritent les *functors* 0D. Grâce à cette contrainte, il est possible, pour l'utilisateur, d'hériter de cette classe en Python, afin de définir son propre *functor* 0D.

Utilisation dans L_1 d'objets définis dans L_2 Un objet défini dans L_2 et dérivant d'une classe définie dans L_1 par exemple doit pouvoir être manipulé en L_1. Ainsi, si on a une classe de base A définie en C++ et une classe B définie en Python qui hérite de A et que B surcharge une méthode m de A, alors un appel à $B.m()$ en C++ doit faire appel au code de m défini en Python.

Par exemple, L'opérateur de découpage, défini en C++, prend un *functor* 0D en argument. La première contrainte nous permet de savoir qu'on peut appeler cet opérateur en

python. La deuxième contrainte nous offre le choix d'utiliser un des *functors* pré-définis ou de définir notre propre *functor* en Python (celui-ci dérivant alors de la classe de base). La troisième contrainte nous permet de passer ce *functor* en argument à la fonction de découpage.

Il existe en Python un certain nombre de mécanismes permettant de développer des interfaces avec le C++. Nous utilisons la librairie *swig* [Bea96] qui exploite ces mécanismes de manière à permettre le développement rapide d'une telle interface.

Remarque : En pratique, plusieurs mécanismes du C++ (*e.g.* les *templates*) ne sont pas gérés par Python. Bien que *swig* propose une émulation de quelques uns de ces mécanismes manquant, il est souvent préférable de contourner ces derniers en s'appuyant sur des techniques mieux supportées (*e.g.* l'héritage).

Nous étendons ainsi Python par notre librairie de description de style, qui inclut des structures de données (*e.g.* le graphe de vue), des mécanismes d'accès à l'information à partir de ces structures (*cf.* chapitre 4) et l'ensemble des opérateurs dédiés à la description de style. En outre, notre système fournit un certain nombre de règles (*e.g.* prédicats, *shaders*) standard.

Le travail de code que doit fournir l'utilisateur consiste principalement à surcharger des fonctions, des prédicats ou des *shaders* et à utiliser les opérateurs et les composants standard fournis.

5.4.2 Règles de style

Comme nous l'avons vu, la plupart des opérateurs se décomposent en un structure algorithmique fixe d'un côté et un ensemble de règles variables, de l'autre côté. Ce sont ces règles, dont la programmation est à la charge de l'utilisateur, qui permettent de contrôler le comportement de chaque opérateur.

Nous parcourons dans cette section les différents types de règles et expliquons pour chacun d'eux sur quel modèle il est implémenté. Nous citons également à chaque fois quelques unes des règles standard fournies par le système.

Prédicats

Les prédicats sont implémentés par des *functors* dont le type de retour est un booléen. Les types de base des prédicats sont définis en C++. Trois types de prédicats peuvent être manipulés dans un module de style :

Les prédicats unaires 0D qui prennent un seul argument de type point (itérateur sur point en pratique, *cf.* section 4.5). Nous adoptons, dans notre librairie, la convention qui consiste à suffixer les noms de ces prédicats par "UP0D", pour *Unary Predicate 0D*.

Les prédicats unaires 1D qui prennent un seul argument de type arête du graphe de vue ou trait. Ils sont suffixés par "UP1D", pour *Unary Predicate 1D*.

Les prédicats binaires 1D qui prennent deux arguments de type arête du graphe de vue ou trait. Ces prédicats définissent souvent des fonctions de comparaison entre les deux éléments passés en argument. L'algorithme de tri par exemple utilise ce type de règle pour ordonner la séquence d'éléments. Le suffixe de ces prédicats est "BP1D", pour *Binary Predicate 1D*.

La plupart des fonctions d'informations sont naturellement associées à des prédicats (*e.g.* la fonction de discontinuité en profondeur est associée à un prédicat indiquant si celle-ci est inférieure à une distance donnée) qui sont, dans leur grande majorité, fournis par le système.

Parmi les prédicats plus avancés on trouve :

ExternalContourUP1D Ce prédicat permet de dire si un élément 1D appartient au contour extérieur d'une scène, c'est-à-dire si un contour donné, qui est une silhouette ou un bord, ne cache aucune surface. Pour un trait constitué de plusieurs arêtes, on renvoie *vrai* si au moins un segment de ce trait appartient au contour extérieur.

ContourUP1D Ce prédicat permet de dire si un élément 1D appartient au contour d'un objet. Pour un trait constitué de plusieurs arêtes, on renvoie *vrai* si au moins un segment de ce trait appartient au contour.

Fonctions d'information

Les fonctions d'information sont également implémentées par des *functors*, comme nous avons pu le voir à la section 4.5. Le type de retour d'une fonction dépend du type de l'information à laquelle elle permet d'accéder. Les fonctions travaillent sur des itérateurs 0D ou sur des éléments 1D. Pour chaque requête d'information, une fonction est fournie par le système.

La fonction est en quelque sorte la base de toute règle, dans la mesure où un prédicat quelconque s'appuiera en général sur une fonction pour déterminer la valeur à retourner.

Itérateurs

Dans notre système, plusieurs règles liées à des parcours sont implémentées à l'aide d'itérateurs. Ces derniers ont l'avantage d'encapsuler les fonctions de parcours du côté du conteneur, permettant ainsi d'écrire des algorithmes génériques et lisibles. Nous avons vu à la section 4.5 que des itérateurs 0D étaient utilisés pour représenter les points munis de leur contexte 1D.

Deux autres types d'itérateurs sont manipulés dans un module de style et sont associés à l'opération de chaînage. Le premier est associé à un sommet et permet d'itérer sur les arêtes du graphe de vue adjacentes à ce sommet (on itère sur les arêtes dans l'ordre inverse des aiguilles d'une montre). Cet itérateur permet d'encapsuler un certain nombre de contraintes (*e.g.* la contrainte de n'itérer que sur des arêtes faisant partie de la sélection courante) qui seront ainsi cachées à l'algorithme de chaînage.

Le deuxième type d'itérateur manipulé correspond à la règle la plus caractéristique du chaînage. Il travaille en collaboration avec le premier et permet d'itérer sur l'ensemble des arêtes du graphe en partant d'une arête donnée et en suivant le graphe. La fonction *traverse* utilisée par les opérateurs d'incrémentation et de décrémentation de cet itérateur de chaînage fait elle-même appel au premier type d'itérateur : sur une arête donnée, pour déterminer la prochaine arête à suivre, on itère sur les arêtes connectées au sommet extrémité. Tous les itérateurs de chaînage peuvent optionnellement être contraints à rester dans la sélection et à ne pas considérer les arêtes ayant déjà été chaînées.

Quelques itérateurs de chaînages classiques sont mis à la disposition de l'utilisateur :

NatureChainingIterator Cet itérateur de chaînage construit un trait en suivant les arêtes de même nature le long d'un objet. Cet itérateur implémente en quelque sorte le chaînage le plus naturel (respect de la topologie des objets). Ainsi, en partant d'une arête donnée

E, le chaînage se poursuivra le long d'une arête E', si E et E' sont connectées en 3D et si elles sont de même nature. On chaîne en priorité les arêtes de silhouette. Un exemple de traits construits à l'aide de ce chaînage est montré sur la figure 5.14 (b).

ChainPredicateIterator Cet itérateur permet de chaîner les arêtes respectant certaines conditions définies par l'utilisateur. Ces conditions peuvent porter sur l'arête seule ou sur le couple formé par l'arête candidate et l'arête précédente ou sur les deux. Il prend en arguments un prédicat unaire Π^1 et un prédicat binaire Π^2. La fonction *traverse* itère alors sur les arêtes candidates et teste pour chacune d'elles les prédicats Π^1 et Π^2. La première arête vérifiant ces deux conditions est retournée. La fonction *traverse* est implémentée selon l'algorithme suivant :

Algorithme 5.7 Itérateur de chaînage : *ChainPredicateIterator*

Fonction *traverse*

Entrée: Π^1 : un prédicat unaire opérant sur des éléments 1-dimensionnels, Π^2 : un prédicat binaire opérant sur des éléments 1-dimensionnels, $\{E_i\}_i$: les arêtes candidates à poursuivre la chaîne

Sortie: E_{next} : l'arête par laquelle se poursuit la chaîne

1. $E \leftarrow$ l'arête courante
2. **pour** $E_i \in \{E_i\}_i$
3. **si** $\Pi^1(E_i) = true$ et $\Pi^2(E, E_i) = true$
4. **alors** $E_{next} \leftarrow E_i$
5. $E_{next} \leftarrow nil$

Cet itérateur permet par exemple d'implémenter les chaînages suivants :

Chaînage en suivant le contour extérieur de la scène Le prédicat unaire passé à l'itérateur vérifie qu'une arête appartient au contour extérieur d'une scène. tandis que le prédicat binaire est toujours vrai (aucune contrainte par rapport à l'arête précédente n'est nécessaire). Ce chaînage est illustré par la figure 5.14 (a).

Chaînage en suivant le contour d'un objet Le prédicat unaire vérifie cette fois que l'arête considérée est bien une arête de contour, tandis que le prédicat binaire vérifie que cette arête appartient bien au même objet que l'arête précédente.

Chaînage en suivant les différents plans d'une scène Ce chaînage permet, par exemple, de construire des traits délimitant un premier plan, un deuxième plan, un arrière plan, *etc.*... d'une scène. Le prédicat unaire vérifie que l'arête considérée est une arête de contour et que sa discontinuité en profondeur moyenne est suffisamment importante. Le prédicat binaire permet de tester que deux arêtes consécutives sont à une profondeur proche. La figure 5.13 illustre un chaînage utilisant cet itérateur.

SketchyChainingIterator Cet itérateur permet de construire une chaîne passant plusieurs fois par les mêmes arêtes. En perturbant la géométrie de cette chaîne de manière pseudo-aléatoire, on obtient un effet "esquissé" particulièrement réaliste. Cet itérateur peut en réalité être n'importe lequel des itérateurs présentés précédemment pour lequel le chaînage multiple d'une même arête est actif (*cf.* section 5.2.3). Il s'agit ensuite d'autoriser cette multiplicité un nombre de fois fini. L'itérateur de chaînage "sketchy" est illustré par la figure 5.14 (c).

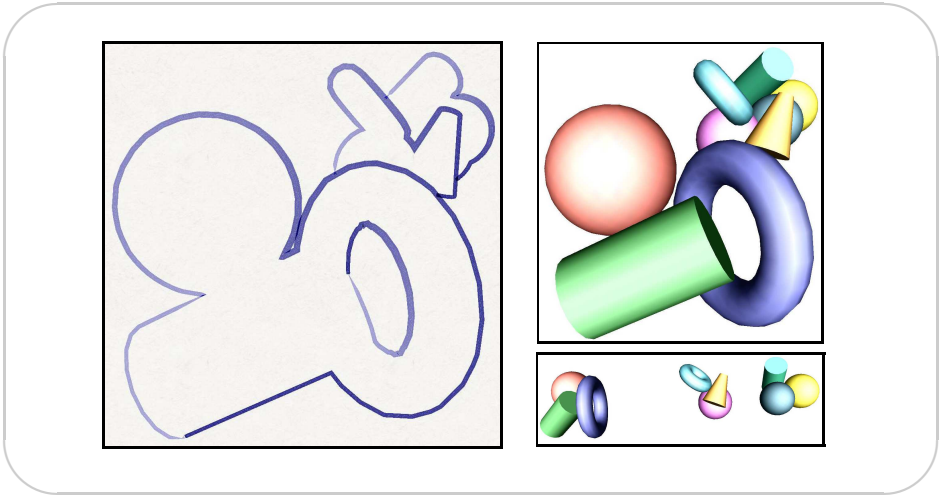


FIG. 5.13 – Itérateurs de chaînage : contour des plans

Gauche : *Chaînage basé sur l'information de profondeur pour dessiner des traits autour des premier, deuxième et arrière plans.* **Droite (haut) :** *La scène 3D sous le même point de vue.* **Droite (bas) :** *La scène sous un point de vue supérieur mettant en évidence les distances entre objets.*

shaders standard

Un certain nombre de *shaders* standard sont fournis par le système. Nous listons ici les plus classiques :

SamplingShader Ce *shader* permet de rééchantillonner les sommets d'un trait (*cf.* section 4.3). Le rééchantillonnage peut être exprimé en distance entre deux sommets ou en nombre de sommets pour le trait.

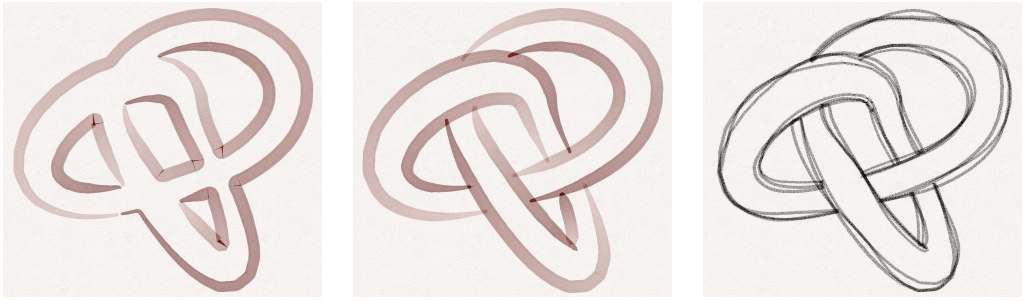


FIG. 5.14 – Exemples d'itérateurs de chaînage simples

Sur cette figure, le chaînage est à chaque fois appliqué à l'ensemble des arêtes appartenant au contour extérieur de la scène : (a) suit le contour extérieur, (b) suit la topologie de l'objet et la silhouette, (c) suit la silhouette sur le même objet et autorise le chaînage multiple des arêtes. Notons que dans les cas (b) et (c) l'opération de chaînage a inclus des arêtes ne faisant pas partie de la sélection initiale.

ConstantThicknessShader Ce *shader* basique permet d'assigner une épaisseur constante à l'ensemble d'un trait. (*cf.* figure 5.15 (b).)

ConstantColorShader C'est cette fois une couleur constante que l'on peut assigner à un trait. (*cf.* figure 5.15 (c).)

StrokeTextureShader Nous avons mentionné que le système de marques était simulé à l'aide de textures (ce point sera détaillé à la section 5.4.5), utilisées pour moduler les valeurs de transparence en chaque pixel d'un trait. Ce *shader* permet à l'utilisateur de spécifier le fichier de texture à utiliser. Outre le nom de fichier contenant l'image de la texture, ce *shader* prend en arguments le type de médium simulé (de manière à choisir le mode de mélange à utiliser) ainsi qu'un booléen indiquant si l'image contient également des textures spécifiques pour les extrémités des traits. (*cf.* figure 5.15 (d).)

VaryingThicknessShader Ce *shader* prend en arguments deux valeurs d'épaisseur : une épaisseur pour les extrémités et une pour le milieu du trait. L'interpolation pour les valeurs intermédiaires suit un schéma d'interpolation linéaire sphérique (SLERP). (*cf.* figure 5.15 (e).)

VaryingColorShader De manière similaire au précédent *shader*, celui-ci prend deux couleurs en arguments, une pour les extrémités et une pour le milieu du trait, les couleurs intermédiaires étant, cette fois, interpolées linéairement. (*cf.* figure 5.15 (f).)

BezierCurveShader Ce *shader* permet de lisser la géométrie d'un trait en le déformant selon son approximation par une courbe de bézier.

SmoothingShader Ce *shader* permet également de lisser la géométrie d'un trait, mais cette fois à l'aide d'une méthode itérative. (*cf.* figure 5.16.)

SpatialNoiseShader Ce *shader* permet de perturber la position des sommets d'un trait à l'aide d'un bruit de Perlin. L'amplitude et la fréquence du bruit font partie des arguments. Cette fonctionnalité est essentielle pour donner à l'image finale un aspect crédible de dessin. Les figures 5.17 (a) et (b) illustrent le *shader* de bruit spatial.

Les fonctions de bruit de Perlin utilisées pour programmer les différents *shaders* de bruit sont également mises à la disposition de l'utilisateur.

ThicknessNoiseShader Ce *shader* permet de perturber les valeurs de l'épaisseur du trait en ses sommets en suivant un bruit de *Perlin* [Per85, EMP+94]. Les figures 5.17 (c) et (d) illustrent le *shader* de bruit d'épaisseur.

ColorNoiseShader Ce *shader* permet de perturber la couleur du trait aux sommets en suivant un bruit de *Perlin*. Les figures 5.17 (e) et (f) illustrent ce *shader*.

TipRemoverShader Ce *shader* de "rognage" permet d'éliminer des portions de trait aux extrémités (*cf.* figure 5.18 (a)).

BackboneStretcherShader Ce *shader* permet d'étirer un trait. Il est en particulier utilisé pour ajouter un aspect "esquissé" à un dessin (*cf.* figure 5.18 (b)).

GuidingLinesShader De même que pour le *shader* précédent, celui-ci permet d'obtenir un effet d'esquisse, en approximant chaque trait par sa tangente en son milieu (*cf.* figure 5.18 (c)).

CalligraphicShader Ce *shader* permet d'assigner à un trait une épaisseur d'autant plus importante que la direction de celui-ci se rapproche d'un vecteur spécifié par l'utilisateur, donnant ainsi au trait un aspect calligraphique (*cf.* figure 5.18 (d)).

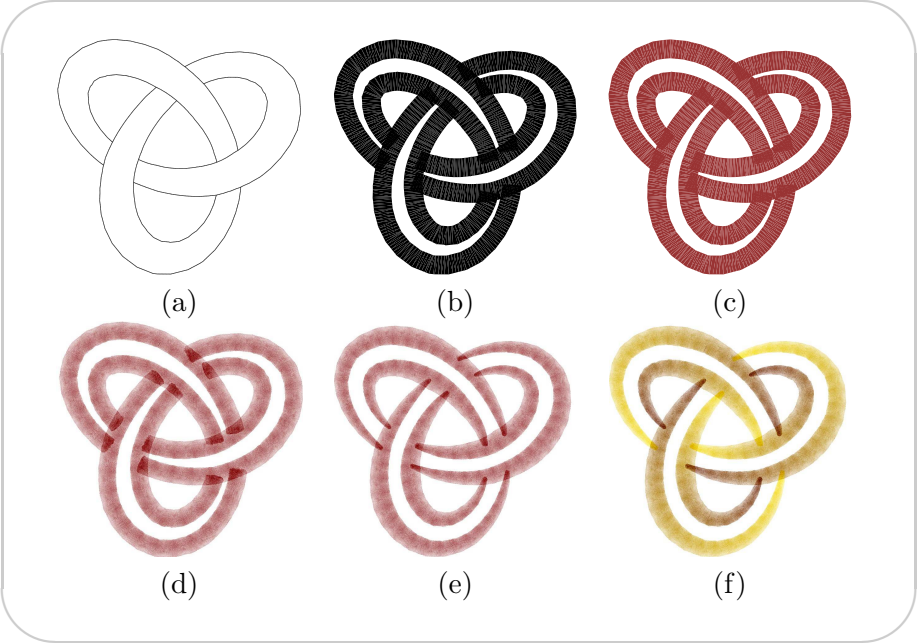


FIG. 5.15 – *Shaders* standard : *shaders* basiques

Ces illustrations montrent le résultat de l'application séquentielle des différents shaders *basiques fournis par le système. (a) : L'ensemble initial de traits avec leurs attributs par défaut. (b) : Une épaisseur plus importante a été affectée uniformément à l'aide du* shader *d'épaisseur constante. (c) : Une couleur a été affectée uniformément à l'aide du* shader *de couleur constante. (d) : Une texture a été affectée. La texture ne modifie que les valeurs de transparence du trait. La couleur et l'épaisseur restent inchangées. (e) : On fait ensuite varier l'épaisseur des lignes à l'aide du* shader *de variation d'épaisseur non uniforme. (f) : Finalement, c'est la couleur qui est modifiée pour évoluer linéairement entre deux couleurs.*

5.4.3 Pipeline d'opérateurs

L'implémentation du pipeline d'opérateurs mérite qu'on s'y arrête un instant. La principale question qui y est associée concerne la mise en œuvre des différents types de synchronisation. Nous avons vu section 5.3.2 qu'il était intéressant de proposer deux modes de synchronisation différents pour l'ordonnancement des éléments au sein du pipeline d'un module de style et qu'un tel pipeline pouvait en réalité être divisé en deux sous-pipelines, chacun fonctionnant sur un mode de synchronisation propre. Comment gérer ces deux modes de fonctionnement dans un module de style ? Une première option pourrait consister à introduire une instruction de programmation supplémentaire qui permettrait à l'utilisateur de spécifier dans quel mode de synchronisation il se place avant une série d'appels aux opérateurs. L'inconvénient de cette approche est qu'elle ne facilite pas la prise en main du système, pour deux raisons : premièrement, elle suppose, de la part de l'utilisateur, une bonne compréhension de mécanismes de bas niveau (alors même qu'en n'exposant pas explicitement la liste des éléments manipulés à un instant donné, on tente de masquer de tels mécanismes) et, deuxièmement, elle représente une source d'erreurs importante dans le processus de programmation de style.

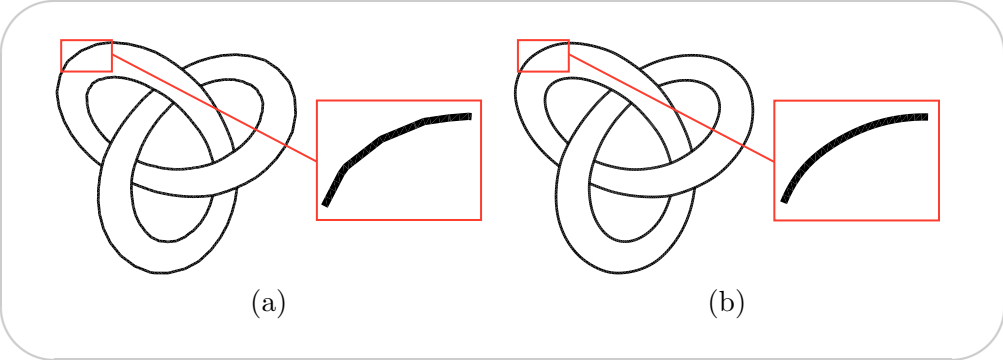


FIG. 5.16 – *Shader* de lissage

Les lignes caractéristiques d'un modèle sans lissage (a) et avec lissage (b).

Nous avons opté pour une autre solution qui consiste à encapsuler l'ensemble du sous-pipeline final (qui contient l'opérateur de dessin et est synchronisé par éléments) dans un opérateur unique : l'opérateur *création*.

L'opérateur *création*

Avec cette mise en œuvre, la liste des opérateurs fournie dans notre implémentation diffère donc légèrement de celle présentée dans la section précédente.

Dans la suite de cette section, nous nous attachons donc à décrire cet opérateur. L'implémentation du premier sous-pipeline d'opérateurs est, quant à elle, directe.

Principe L'opérateur de création fonctionne sur le mode de synchronisation par éléments et combine trois appels séquentiels aux opérateurs de sélection, de *shading* et de dessin :

$$creation = [selection][shade]^* dessin$$

Entrées/sorties Les entrées de cet opérateur sont les traits et la sortie est l'image finale.

Règles Les règles à spécifier sont les mêmes que pour les différents opérateurs encapsulés dans cet appel. Ainsi, on peut spécifier un prédicat unaire 1D, utilisé pour la sélection, ainsi qu'un ensemble de *shaders*.

Algorithme L'algorithme de l'opérateur création est montré sur l'algorithme 5.8.

Une autre conséquence de cette mise en œuvre est qu'on ne manipule jamais vraiment d'opérateurs de *shading* (bien qu'on implémente des *shaders*) ou de dessin.

5.4.4 Paramétrisation multiple

Il est parfois souhaitable de programmer un *shader* de manière à ce qu'un attribut varie le long d'une chaîne "virtuelle". Sur la figure 5.19 (a) par exemple, l'épaisseur du trait traçant la

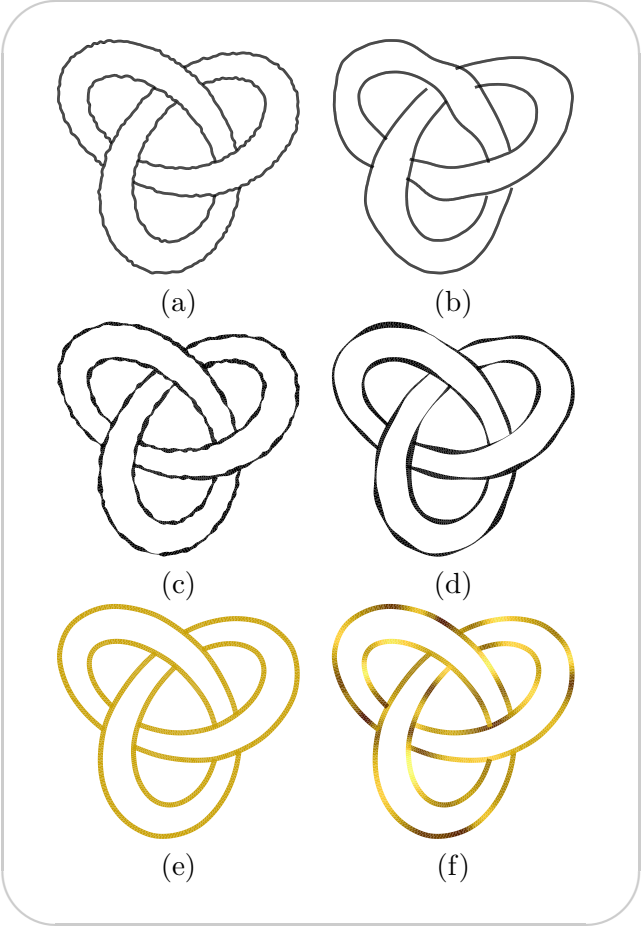


FIG. 5.17 – *Shaders* standard : *shaders* de bruit

Plusieurs shaders *de bruit sont fournis par le système et permettent de perturber la géométrie, l'épaisseur ou la couleur des traits. (a) et (b) : Le* shader *de bruit spatial permet de perturber la géométrie des traits. Sur cette figure, on voit deux exemples d'utilisation de ce* shader *avec des paramètres (amplitude et fréquence) différents. (c) et (d) : De même, le* shader *de bruit d'épaisseur permet de perturber l'épaisseur des traits. Ces deux exemples ont été réalisés avec des paramètres (amplitude et fréquence) différents. (e) et (f) : Les traits sont tracés en couleur sans* shader *de bruit (e) et avec* shader *de bruit (f).*

silhouette de la sphère varie continûment au travers des occultations. Le *shader* programmant cet attribut s'appuie donc sur une paramétrisation couvrant une chaîne s'étendant au travers des parties cachées, bien que le trait ne soit, lui, tracé que sur les parties visibles.

Cette idee peut être étendue à la notion de *paramétrisation multiple* pour laquelle, dans un même module de style, plusieurs *shaders* différents peuvent s'appuyer sur plusieurs chaînes virtuelles différentes et, donc, sur plusieurs paramétrisations. Sur la figure 5.19 (b) par exemple, on fait varier deux attributs (l'épaisseur et la couleur) selon deux paramétrisations différentes.

Ce type de paramétrisation multiple est réalisable grâce à la possibilité de programmer les itérateurs de chaînage et les *shaders*. En revanche, une telle programmation reste quelque peu fastidieuse avec notre implémentation actuelle.

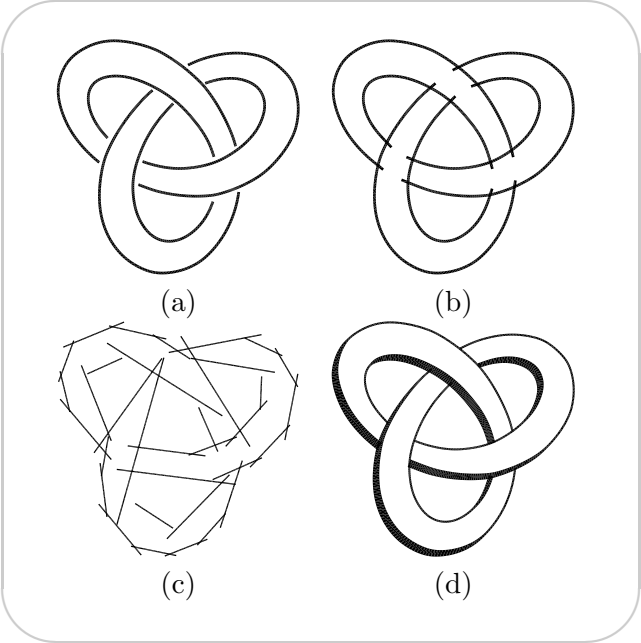


FIG. 5.18 – *Shaders* standard : rognage, étirement, lignes directrice et calligraphie

*(a) : Les portions extrémales des traits ont été supprimées sur une longueur spécifiée par l'utilisateur (*TipRemoverShader*). (b) : La géométrie des traits est étirée d'une longueur spécifiée par l'utilisateur (*BackboneStretcherShader*). (c) : Les traits sont déformés de manière à correspondre à une approximation de leur tangente en son milieu (*GuidingLinesShader*). (d) : Ce* shader *(*CalligraphicShader*) permet d'assigner au trait une épaisseur variable en fonction de la direction de manière à imiter un effet calligraphique.*

Une manière plus naturelle d'obtenir cette paramétrisation multiple (et également d'organiser le pipeline) aurait été de permettre des appels à un opérateur de *shading* à plusieurs endroits du pipeline (en particulier dans le sous-pipeline synchronisé par opérateur). Il aurait alors suffit, pour réaliser l'illustration de la figure 5.19 (b), d'appeler cet opérateur une première fois après un chaînage ne tenant pas compte de l'information de visibilité pour affecter au trait ainsi construit la couleur et la géométrie, puis une deuxième fois, après un

Algorithme 5.8 Opérateur de création

Fonction *Create*

Entrée: Π : un prédicat unaire opérant sur des éléments 1-dimensionnels, $\{S_p\}_p$: un ensemble de *shaders*, $\{T_i\}_i$: un ensemble de traits

Sortie: le dessin

1. $\{T_i\}_i \leftarrow$ ensemble actif de traits
2. **pour** $T_i \in \{T_i\}_i$
3. **si** $\Pi(E) = true$
4. **alors pour** $S_p \in \{S_p\}_p$
5. $S_p(T_i)$
6. $dessiner(T_i)$
7.

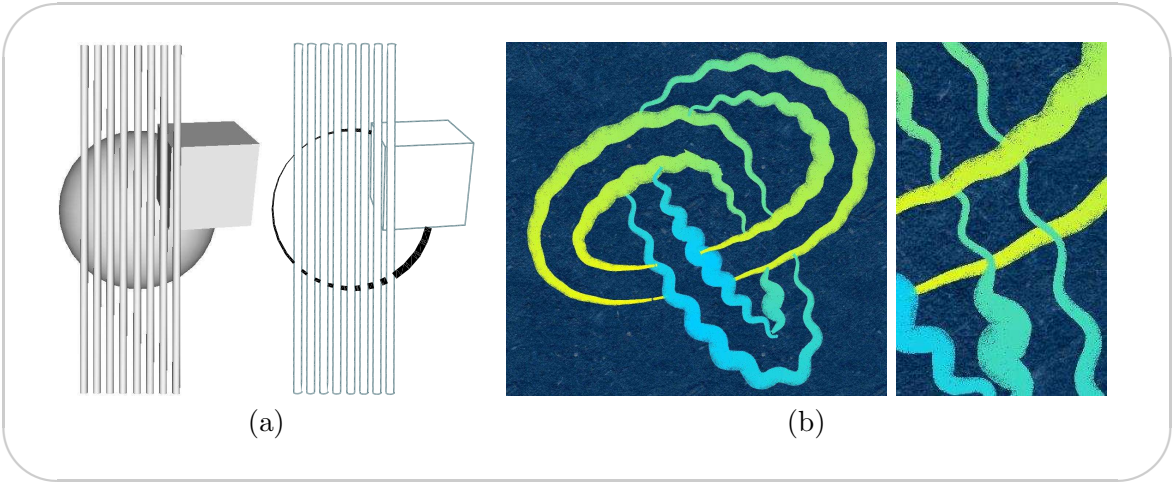


Fig. 5.19 – Multiple paramétrisation des traits

(a) : L'épaisseur du trait décrivant la silhouette de la sphère varie continûment au travers des petites occultations. (b) : Dans cette image, différents shaders *utilisent deux paramétrisation distinctes pour contrôler les attributs : l'une couvre la totalité de la ligne de silhouette, portions cachées incluses ; et l'autre sépare et couvre les segments visibles de la silhouette.* **Gauche :** *la couleur et la géométrie varient selon la paramétrisation complète, tandis que l'épaisseur du trait varie selon la paramétrisation s'appuyant sur les segments visibles.* **Droite :** *Gros plan sur la partie en bas à droite de la scène, dans laquelle nous avons ajouté les parties cachées de la silhouette. En comparant cette image avec l'image de gauche, on voit bien que le détail géométrique sinusoïdal traverse continûment les parties invisibles.*

découpage et une sélection conservant uniquement les parties visibles des traits, pour affecter l'épaisseur.

Une extension logique et intéressante du pipeline consisterait donc à ajouter un opérateur *shade*, prenant un *shader* en argument. Il serait ainsi possible de manipuler des traits à qui certains attributs auraient déjà été assignés avant l'étape de dessin. L'illustration de la figure 5.19 (b) par exemple pourrait être réalisée à l'aide d'une séquence simple d'opérations incluant notamment des *shaders* modifiant les attributs de manière relative. Ainsi, les *shaders* de déformation géométrique et de variation de couleur seraient appliquées après une première étape de chaînage ne tenant pas compte de l'information de visibilité ; une opération de découpage serait ensuite appliquée à cette chaîne (munie de ses nouveaux attributs de géométrie et de couleur), permettant de séparer les segments de visibilité différente ; les segments visibles seraient alors conservés par sélection et, finalement, un nouvel opérateur de *shading* modifierait ces nouvelles chaînes (munies de leur nouvelle paramétrisation) en leur affectant la variation d'épaisseur souhaitée.

5.4.5 Rendu des traits

Du point de vue de l'architecture logicielle, notre approche se situe en amont du rendu et peut se brancher sur un système de marques permettant la génération de l'image finale à partir des traits munis de leurs attributs. Le développement d'un système programmable de rendu de marques constitue d'ailleurs un projet excitant pour des travaux futurs.

Notre système intègre toute fois un composant simple de rendu qui s'appuie sur la bibliothèque graphique standard OpenGL et permet une visualisation du résultat. Les traits sont transformés en bandes de triangles à partir de l'échantillonnage de leur géométrie et de leur épaisseur puis rendus. La figure 5.20 montre la bande de triangles associé à un trait donné.

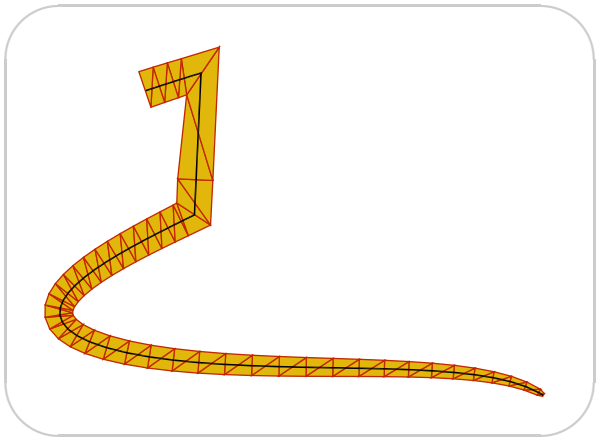


FIG. 5.20 – Rendu des traits : bande de triangles

Sur cette illustration, on voit la bande de triangles construite pour un trait donné dont l'épine dorsale est tracée en noir. C'est cette bande qui est rendue à l'aide de l'API OpenGL dans le dessin.

Des techniques établies permettent d'éviter les singularités aux points de haute courbure, *e.g.* [SS02] chapitre 3 et [HL94].

Nous utilisons des textures de traits réels comme des cartes de transparence (*alpha maps*) pour améliorer la qualité visuelle (*cf.* figure 3.7 page 63). Cette technique nous permet d'imprimer à nos traits l'apparence du trait réel tout en conservant le contrôle des attributs tels que la couleur et l'épaisseur. L'utilisation des textures nous permet également d'imiter les interactions entre le medium et le support sans avoir recours à une simulation physique qui empêcherait un affichage interactif.

Nous utilisons les modes de mélange (*blending modes*) d'OpenGL pour émuler différents types de médiums physiques. En pratique, nous rendons l'inverse de l'image (pour laquelle un support blanc correspond à la couleur RGB (0,0,0)). Ceci nous permet en particulier de simuler la nature soustractive de la plupart des média. Pour les média épais tels que la peinture à l'huile, nous utilisons un simple mode de remplacement (*cf.* figure 5.21 (a)). Le mélange additif (qui devient soustractif dans notre contexte inverse) est bien adapté pour les matériaux humides tels que l'encre (*cf.* figure 5.21 (b)). Enfin, le mode de mélange *minimum* fourni par OpenGL 1.2 [W+99] peut imiter le graphite et d'autres média secs (*cf.* figure 5.21 (c)). Une texture de papier peut également être appliquée. Elle n'est cependant utilisée que pour le rendu final et n'affecte pas les calculs de densité.

En outre, nous proposons un composant de rendu Postscript qui permet de créer des fichiers au format *Encapsulated Postscipt* dans lesquels les traits sont, au choix, sauvés sous forme de bandes de triangles ou alors sous forme de lignes (auquel cas les variations d'épaisseur ne sont pas prises en compte). La seule limitation de ce type de rendu est qu'il ne gère pas les textures, qui ajoutent beaucoup à la qualité de l'image finale.

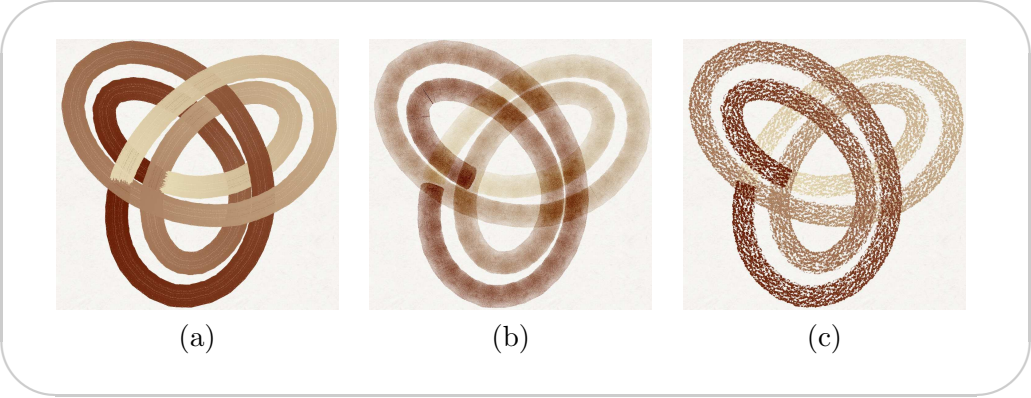


FIG. 5.21 – Rendu des traits : simulation des media

(a) : Simulation de media épais (e.g. peinture à l'huile) en utilisant le mode de mélange par remplacement. (b) : Simulation de matériaux humides (e.g. encre) avec le mode de mélange additif. (c) : Simulation de media secs (e.g. graphite) avec le mode de mélange minimum.

Résumé: Dans notre mise en œuvre de l'approche, nous avons utilisé deux langages différents (*C++* et *Python*) pour l'application principale et la description de style. Le système fournit de nombreuses règles standard qui permettent de faciliter la tâche de l'utilisateur. Dans notre implémentation, le sous-pipeline final, comprenant l'opérateur de dessin, est encapsulé dans un unique opérateur *create*. Finalement, le rendu des traits est fait en s'appuyant sur la librairie graphique OpenGL.

chapitre 6

Mesures de densité pour la simplification de dessins au trait

Dans ce chapitre, nous présentons une approche pour contrôler l'encombrement de lignes dans le rendu NPR de dessins au trait. Cette approche s'appuie sur une information de densité calculée dans la vue et sur le dessin. Nous définissons deux types d'information de densité : la densité *a priori* et la densité *causale* et les utilisons pour contrôler quelles parties du dessin doivent être simplifiées. La densité *a priori* est une mesure de la complexité visuelle du dessin potentiel et est calculée sur l'arrangement complet des lignes de la vue (en pratique, sur les arêtes visibles du graphe de vue). Cette mesure permet de développer une approche systématique pour caractériser la structure des régions encombrées en terme de géométrie, d'échelle, et de directionnalité. La densité *causale* mesure la complexité du dessin dans son état courant pendant que les traits y sont ajoutés, permettant le contrôle de l'encombrement par omission ou stylisation de lignes.

Nous montrons comment ces mesures de densité permettent de simplifier un dessin de plusieurs manières pour lesquelles la complexité est réduite soit uniformément soit d'une façon spatialement variable au travers de l'*indication*.

L'information de densité s'inscrit dans le cadre des informations utiles au style étudiées au chapitre 4. De même, l'utilisation qu'on en propose dans ce chapitre est plus particulièrement dédiée à un système programmable. En effet, l'utilisateur doit pouvoir disposer d'une grande flexibilité pour spécifier les règles de sélection et de tri qui y sont associées et qui peuvent être compliquées. Toutefois, de par sa complexité, la densité occupe une place particulière dans la liste des informations et il nous semblé pertinent de détacher au maximum sa description du système présenté dans les précédents chapitres. Le ton adopté ici se veut donc plus général que pour les autres informations.

6.1 Introduction

Dans le rendu NPR de dessins au trait, lorsque la complexité de la scène augmente, les images générées peuvent souffrir d'encombrement lorsque trop de lignes sont dessinées sur une petite surface. Ce problème est en particulier soulevé par la représentation de structures complexes telles que les murs de briques, les toits de tuiles, les arbres, et s'accentue lorsque ces structures sont observées sous des angles rasants. Par opposition à la photographie, les dessins permettent l'omission de détails ou l'abstraction et les artistes ont développé un grand

nombre de techniques picturales pour éviter l'encombrement tout en préservant la forme et l'information. Par exemple, il est classique d'omettre les structures trop petites, d'exploiter les répétitions dans la scène et d'omettre les détails de texture. La quantité locale de traits (la *densité*) est contrôlée avec soin, de manière à éviter l'encombrement, focaliser l'attention ou créer du dynamisme.

En particulier, les structures répétitives ou semi-répétitives telles que les textures, la végétation, ou les groupes d'objets similaires soulèvent des questions cognitives et picturales intéressantes du fait de l'importance de l'encombrement qu'ils génèrent et parce que la similarité qui les caractérise peut être soulignée ou exploitée. Nous identifions deux stratégies picturales utilisées par les artistes pour aborder le problème de l'encombrement dans le dessin au trait de structures répétitives ou pseudo-régulières. Elles diffèrent par leur perspective (emphase *vs.* exploitation de la répétition) et par leur style visuel (complexité visuelle uniforme *vs.* spatialement variable).

L'élagage uniforme assure une faible complexité en omettant les lignes de façon homogène. On obtient ainsi une image de densité uniforme sur laquelle la complexité de la vue originale est complètement cachée. La régularité est soulignée, car la représentation de la structure régulière est également régulière. En pratique, l'élagage uniforme peut être réalisé au travers de techniques liées aux niveaux de détails pour lesquels chaque motif est simplifié en omettant les traits secondaires ; ou encore par le biais d'un sous-échantillonnage des motifs, où des motifs entiers sont omis, comme, par exemple, dessiner une ligne sur deux dans une grille. Les figures 6.1 (a) et (b) illustrent cette stratégie de simplification.

L'indication exploite les structures répétitives et s'appuie sur une simplification non-uniforme pour réduire la complexité globale tout en la suggérant de manière exhaustive dans de petites régions [WS94]. L'artiste dessine en détail seulement quelques parties d'une structure répétitive de manière à suggérer sa complexité globale, par exemple, quelques tuiles sur un toit. L'objectif est de préserver suffisamment de motifs pour transmettre l'information dans toute sa complexité. L'indication est illustrée par les figures 6.1 (c) et (d).

Ces stratégies peuvent être combinées avec des informations sémantiques pour souligner les parties importantes au travers d'une simplification sélective.

Nous pensons qu'un contrôle précis de l'encombrement est fondamental pour produire un rendu convaincant. Pour cela, il est crucial d'imaginer des stratégies de simplification, mais aussi des outils systématiques pour estimer la complexité dans la vue et dans le dessin. Des approches antérieures ont présenté des méthodes puissantes de contrôle de ton et de complexité dans un dessin au trait. Cependant, elles s'appuient souvent sur une spécification manuelle de stratégies d'omission et de seuils de densité. Notre travail en revanche se focalise sur la détermination automatique de la complexité, de la répétition ainsi que sur l'élaboration de stratégies pour le contrôle de l'encombrement dans le dessin au trait.

Contributions présentées dans ce chapitre

Nous présentons une approche générale pour contrôler la simplification de dessin au trait basée sur l'omission et la stylisation de lignes. Nous introduisons des mesures de densité pour quantifier la complexité visuelle dans un dessin. Nous commençons par définir une information de densité *a priori* qui capture quelle complexité visuelle aurait le dessin si toutes les lignes

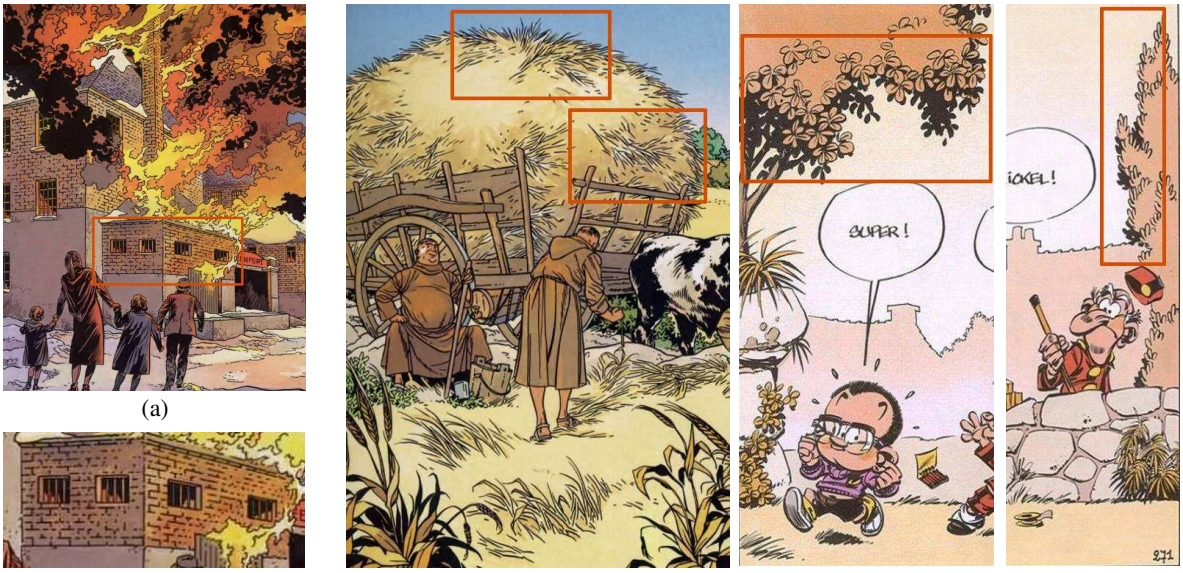


FIG. 6.1 – Élagage uniforme et indication

Avec l'élagage uniforme, les traits sont omis de façon homogène. La simplification d'une structure régulière est ainsi également régulière. (a) : Dans cette illustration, l'artiste a omis des traits selon un élagage uniforme dans sa représentation des murs de briques. Une illustration dans laquelle apparaît un mur de brique. (b) : Un agrandissement sur l'un des murs sur lequel on voit clairement que des traits ont été conservé sur l'ensemble de la surface du mur. © Valles et Van Hamme 1993 - Glénat [VH93]. Avec l'indication, la complexité est suggérée à l'aide de quelques régions seulement qui sont dessinées en détail. (c) : Des pailles ont été dessinées seulement à quelques endroits pour signifier la complexité de la motte de foin. © Valles et Van Hamme 1995 - Glénat [VH95]. (d) : Sur ces deux images, l'artiste a utilisé des indications pour dessiner tout un feuillage à l'aide de quelques feuilles positionnées sur les bords. © Tome et Janry 1994 - Dupuis [TJ94].

de la vue courante étaient dessinées. Celle-ci est calculée à plusieurs échelles et pour plusieurs directions et peut être utilisée pour analyser la structure locale de manière à exploiter et éviter les structures répétitives chargées. Ensuite, pendant le rendu, la complexité du dessin en cours est estimée au travers d'une densité *causale*. Cette information de densité est mise à jour chaque fois qu'un trait est ajouté de façon similaire à Salisbury *et al.* [SWHS97]. Des décisions d'omission ou de stylisation peuvent alors être prises à partir de cette information pour contrôler finement la complexité de l'image finale.

Nous montrons également qu'il peut être particulièrement important, dans ce contexte, de faire précéder le rendu d'un appel à l'opérateur de tri, puisque l'ordre dans lequel les traits sont dessinés importe. Cet aspect est lié aux textures de traits à priorité (*prioritized stroke textures*) [WS94, SWHS97] et aux *tonal art maps* [WPFH02, KKL+00] pour lesquelles l'ordre est défini manuellement. Nous montrons comment ces outils peuvent être utilisés pour réaliser différents types de contrôles de l'encombrement. En particulier, nous réalisons à la fois l'élagage uniforme et l'indication.

Résumé: Nous introduisons deux mesures de densité, la densité *a priori* et la densité *causale* pour automatiser le processus de simplification de dessin au trait. Nous nous focalisons plus spécifiquement sur deux stratégies de simplification classiques : l'élagage uniforme et l'indication.

6.2 Travaux antérieurs

Les questions liées à l'encombrement dans le dessin au trait ont été abordées très tôt par la communauté NPR. Winkenbach *et al.* par exemple ont introduit la notion d'*indication*, qui consiste à dessiner les détails d'une structure complexe seulement à quelques endroits pour en suggérer la complexité, tout en réduisant la surcharge. Ils introduisent la notion de traits à priorité [WS94, SWHS97], qui peuvent être vus comme des motifs de textures en demi-tons. Les traits qui forment une texture sont manuellement triées par ordre d'importance jusqu'à ce que la densité correcte soit atteinte. Les priorités assurent que les caractéristiques les plus pertinentes de la texture sont dessinées en premier. Notre approche s'appuie sur ce travail et l'étend des textures aux scènes générales.

En 1995, Preim *et al.* [PS95] s'intéressent également à la question de simplification de dessins au trait. Leur système permet à l'utilisateur de spécifier interactivement sur le dessin, quelles régions doivent être affichées avec plus de détails. Les lignes peuvent alors être omises ou au contraire renforcées, raccourcies ou laissées à leur longueur originale, de façon à "enrichir" ou "épurer" telle ou telle zone du dessin. Dans cette perspective, les auteurs définissent la *pertinence* (*relevance*) d'une ligne, comme une fonction de sa longueur, de sa distance au point de vue et de la densité de lignes dans l'environnement. De manière similaire, notre approche peut utiliser l'omission ou des effets stylistiques pour simplifier un dessin, et elle intègre également la notion de priorité sur les lignes. En revanche, notre but n'est pas de simplifier une zone à la demande de l'utilisateur, mais de détecter automatiquement les lignes à omettre, en fonction de la stratégie de simplification choisie.

Les notions d'encombrement, de densité et de ton sont reliées entre elles, et des travaux de NPR [Ost99] se sont appuyées sur les techniques de demi-ton pour produire un ton à l'aide de primitives de traits.

Deussen *et al.* [DS00] proposent une approche de simplification dédiée à la végétation et aux arbres. En se basant sur la hiérarchie d'un arbre, ils remplacent les groupes complexes d'objets tels que les feuilles, par des primitives plus simples. De plus, un seuil sur le *Z-Buffer* leur permet de ne rendre que les arêtes marquant une forte discontinuité en profondeur. Cette approche permet une simplification puissante mais est fortement spécialisée dans la mesure où elle s'appuie sur la représentation hiérarchique de la végétation.

Le travail le plus proche du notre est celui de Wilson *et al.* [WM04]. Ils utilisent l'omission de lignes pour représenter des scènes complexes de manière plus légère. Comme nous, ils utilisent une estimation de la densité potentielle du dessin et s'appuient sur l'ordonnancement des traits pour diriger l'omission des lignes. Nous introduisons une information de densité, ainsi qu'une analyse de la structure locale, plus élaborées, et montrons comment la combinaison des densités causales et *a priori* fournit un contrôle complet sur la simplification.

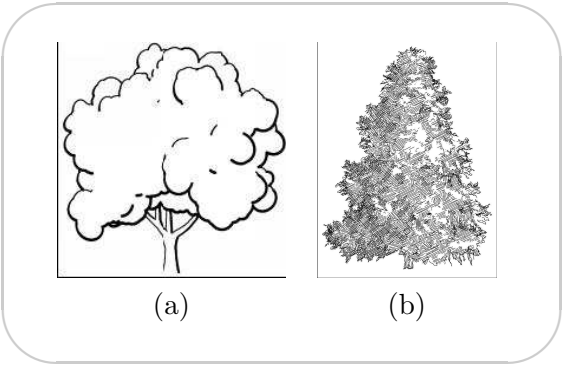


FIG. 6.2 – Simplification de dessins au trait : travaux antérieurs

Deux dessins au trait simplifiés obtenus à partir de modèles 3D d'arbres. (a) : Deussen et al. simplifie directement le modèle 3D de manière à réduire le nombre de primitives à dessiner. © Deussen et al. [DS00]. (b) : Wilson et al. utilisent l'omission de lignes pour simplifier le dessin. © Wilson et al. [WM04].

6.3 Vue d'ensemble de l'approche

Notre approche pour le contrôle de l'encombrement s'inscrit dans le cadre général du rendu NPR de dessins au trait. La méthode prend en entrée un grand ensemble de primitives de lignes qui est un sur-ensemble des lignes qui apparaîtront dans le dessin final. Ces lignes peuvent provenir d'une source 2D ou 3D (silhouettes, *etc.*...). Nous verrons que l'analyse de cet ensemble initial de lignes nous permet de planifier la simplification et d'extraire des motifs localement. Dans le cadre de notre système programmable, l'ensemble initial des lignes est l'ensemble des arêtes visibles du graphe de vue. Comme nous avons pu le voir dans les chapitres précédents, le problème de la simplification de dessin est abordé sous la perspective de l'omission de lignes (via l'opérateur de sélection en pratique) ou de la modification des attributs des lignes (via les *shaders*). Nous ne traitons pas ici la modification topologique de traits ou la création de nouveaux traits. D'une manière générale, cette approche nécessite de travailler dans un contexte applicatif de rendu pour lequel les lignes sont traitées de manière séquentielle : chaque ligne est transformée en trait, stylisée et rendu avant de procéder à la suivante (mode de synchronisation par éléments *cf.* section 5.3.2). Cette séquentialité correspond au processus de dessin où l'artiste voit l'état courant du dessin avant de prendre des décisions au sujet du prochain trait. Nous utilisons une notion causale de densité où la complexité courante locale peut être évaluée, et affecte la décision de dessiner le prochain trait ou pas, ainsi que la manière de le dessiner. C'est le rôle de la *densité causale*. Cette dernière est mesurée dans l'image dans laquelle les traits sont rendus (*cf.* figure 6.3) et est causale dans la mesure où elle est mise à jour à chaque fois qu'un nouveau trait est rendu, et où un trait donné influencera uniquement les décisions portant sur les traits qui le suivent. Par exemple, la densité causale peut être utilisée pour omettre un trait donné si la densité locale est trop importante. La densité causale est décrite à la section 6.5.

La causalité donne ici à l'ordre dans lequel les traits sont dessinés, et donc à l'opérateur de tri, une importance particulière. En effet, il peut être crucial de traiter les traits dans un ordre approprié de manière à ce que les traits "importants" soient dessinés en premier et soient moins à même d'être omis.

Cette information de densité causale est essentielle pour contrôler la complexité courante du dessin. Cependant, elle est limitée sans sa capacité à planifier et à exploiter le traitement de structures répétitives. Or certaines stratégies, telles que l'indication par exemple, qui suggèrent la complexité globale en dessinant seulement quelques régions en détail, notamment souvent aux frontières de la structure, requièrent ces possibilités. Par conséquent, nous introduisons l'information de densité *a priori* qui mesure la complexité visuelle du dessin "potentiel", constitué de l'ensemble des lignes considérées en entrée. La densité *a priori* est évaluée au début du processus de dessin, avant que le rendu séquentiel ne commence (*cf.* figure 6.3). Une interprétation informelle de cette densité est qu'elle correspond à la connaissance préliminaire de la scène que l'artiste possède. L'information de densité *a priori* ainsi que l'analyse structurelle qu'elle permet sont décrites à la section 6.4.

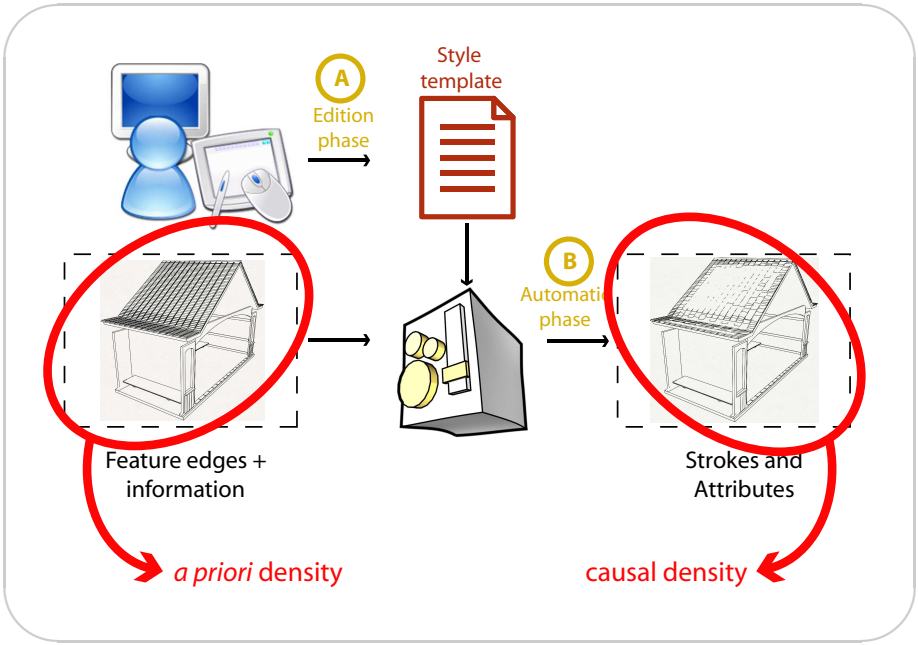


Fig. 6.3 – Mesures de densité : vue d'ensemble

La densité a priori *est mesurée sur l'ensemble des arêtes visibles du graphe de vue, c'est-à-dire sur l'ensemble des lignes qui, potentiellement, peuvent apparaître dans l'image finale. Cette mesure permet d'estimer la complexité du dessin si toutes les lignes étaient dessinées. La densité causale est mesurée sur l'image dans laquelle les traits sont rendus, avant d'ajouter chaque nouveau trait. Elle permet en particulier de décider d'omettre ou non un trait en fonction de la complexité courante du dessin.*

Il est important de remarquer que ces quantités peuvent être évaluées (dans leurs espaces respectifs) en tout point, à n'importe quelle échelle et pour n'importe quelle direction (dans le cas de la densité *a priori*). Les décisions de simplification et de stylisation peuvent s'appuyer sur un simple critère de densité mesurée sur un trait, mais elles peuvent aussi impliquer une analyse plus complexe basée sur des requêtes de densité à différentes échelles, positions et directions. Dans ce chapitre, nous montrons comment des outils d'analyse, simples mais puissants, peuvent être définis à partir de telles requêtes.

Finalement, dans la philosophie inhérente à notre approche programmable, l'objectif est encore ici de séparer les décisions stylistiques des décisions techniques. Nous fournissons la

densité parmi les autres informations et laissons à l'utilisateur la décision de comment exploiter cette information, dans le contexte de modules de style. Dans nos expériences, nous contrôlons l'encombrement en utilisant l'omission de lignes ou en contrôlant les attributs des traits, toutefois d'autres stratégies de simplification pourraient exploiter nos mesures de densité [SP03a].

6.4 Densité a priori

Bien que la notion de complexité (ou de densité) d'un dessin soit intuitive, sa définition précise requiert un soin particulier ; des notions de normalisation et d'échelle doivent notamment, être traitées avec attention.

En outre, du fait de la nature 1-dimensionelle des lignes et des traits, leur orientation joue un rôle important. Par exemple, dans le cas où une ligne horizontale se trouve au milieu d'un grand nombre de lignes verticales, on peut vouloir traiter cette ligne horizontale différemment d'une ligne verticale. Sur la figure 6.11 par exemple, nous utilisons la directionnalité pour distinguer les barreaux horizontaux des barreaux verticaux de la grille, de manière à éviter de supprimer les lignes horizontales à cause du grand nombre de lignes verticales. Nous définissons un estimateur pour la densité de lignes à partir de ces considérations.

Pour aborder ces contraintes, nous nous inspirons de techniques de décomposition d'image telles que les pyramides orientables (*steerable pyramids*) [FA91], et représentons la notion de densité à différentes échelles et orientations.

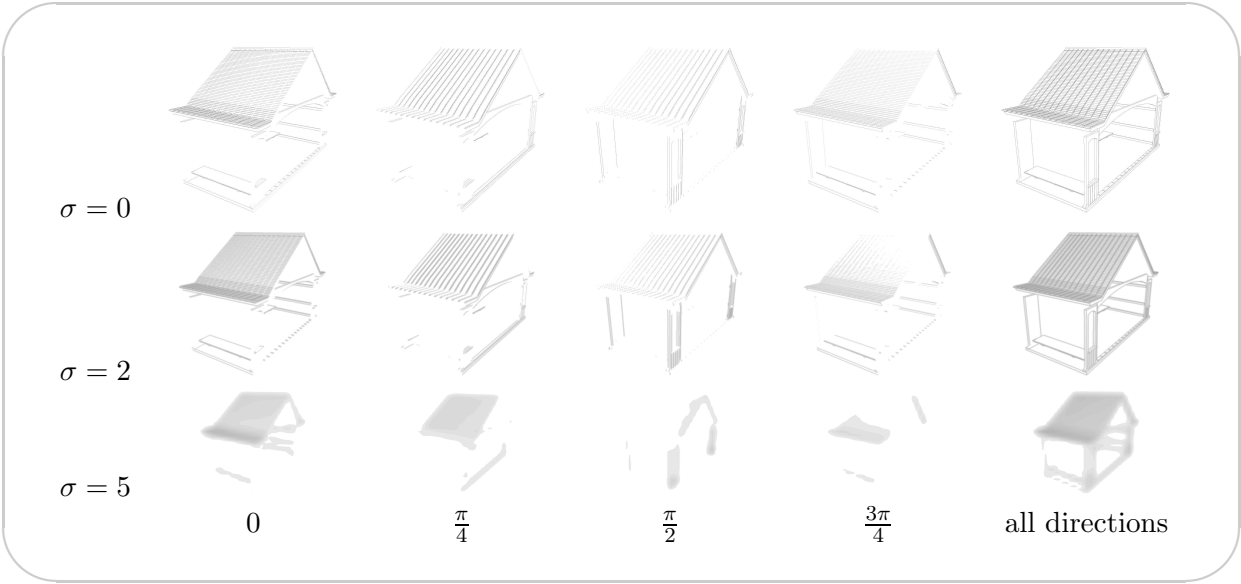


FIG. 6.4 – Pyramides des cartes de densité *a priori*

Les pyramides de cartes de densité a priori *pour le modèle de la maison (pour des raisons de confort visuel, ces cartes sont dessinées en mode négatif et avec un gamma plus grand). Chaque colonne correspond à une direction différente et montre trois niveaux (la base, $\sigma = 2$ et 5 pixels) de la pyramide gaussienne correspondante. La cinquième colonne montre la carte de densité* a priori *a-directionnelle.*

6.4.1 Un estimateur de la densité de lignes

Intuitivement, la densité est définie en un point donné et à une échelle s comme la somme des longueurs des lignes incluses dans un cercle de rayon s, normalisée par l'aire de ce cercle. Cette normalisation est importante pour assurer l'indépendance à l'échelle dans le cas de structures périodiques. Cette densité mesure une longueur par unité de surface. Pour comprendre l'effet de l'échelle s, considérons une structure régulière de lignes verticales. Lorsque le rayon s augmente, la longueur de chaque ligne dans le disque augmente linéairement, et le nombre de lignes qui intersectent le disque augmente aussi linéairement. L'augmentation quadratique qui en résulte est compensée par la normalisation par l'aire du disque et la densité est principalement indépendante de s (pour ce type de structures périodiques).

En pratique, nous calculons une moyenne avec pondération spatiale à l'aide d'un noyau gaussien de variance σ. Par ailleurs, nous découplons l'information donnée par les différentes orientations et définissons la densité pour une direction donnée $\vec{u}$ en utilisant une fonction de pondération w_o sur la direction de la ligne.

La densité d'un ensemble de lignes $\mathcal{L}$ en un point Q, pour une échelle σ et une orientation $\vec{u}$ est alors :

$$d(Q, \sigma, \vec{u}) = \int_{P \in \mathcal{L}} w_d(P, Q, \sigma) w_o(P, \vec{u}) \, dl \tag{6.1}$$

où w_d est la fonction gaussienne circulaire normalisée d'écart type σ :

$$w_d(P, Q, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{\|\overrightarrow{PQ}\|^2}{2\sigma^2}} \tag{6.2}$$

et w_o la fonction de pondération pour l'orientation qui dépend de $\theta(P)$, l'angle entre la tangente à la ligne au point P et $\vec{u}$:

$$w_o(P, \vec{u}) = \begin{cases} |cos(\frac{n\theta(P)}{2})|, \text{ if } \theta(P) \in [-\frac{\pi}{n}, \frac{\pi}{n}], \\ 0 \text{ sinon} \end{cases} \tag{6.3}$$

n contrôle la gamme des angles à laquelle un point donné de $\mathcal{L}$ contribue. Cette fonction de pondération est utilisée pour assurer une normalisation correcte quand les directions sont discrétisées. Elle est reliée aux poids d'interpolation directionnels (*steerable interpolation weights*) [FA91].

Comme nous l'avons mentionné plus haut dans le contexte simple du disque et de lignes verticales, nous utilisons une version normalisée de la fonction gaussienne pour cet estimateur de manière à assurer que la valeur moyenne de la densité des lignes pour le dessin est la même quelque soit le σ utilisé pour définir l'échelle de chaque estimation ponctuelle.

6.4.2 Cartes de densité *a priori*

De manière à être efficace en terme de calcul et d'accès, nous adoptons les approches de la pyramide [BA83] et de la pyramide directionnelle [FA91] et stockons la mesure de densité pour des échelles dyadiques et un nombre donné d'orientations. Les illustrations qui suivent dans ce chapitre ont été réalisées en utilisant quatre directions, dont les angles avec l'horizontale sont $(0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4})$. Les expériences ont montré que ce nombre était suffisant pour caractériser la plupart des aspects structurels. Nous avons également choisi $n = 4$ dans la formule 6.3, égal au nombre de directions ; ainsi un segment contribue au plus à deux cartes directionnelles consécutives.

Par conséquent, chaque carte est associée à une échelle et une direction spécifiques et stocke, pour chaque position de pixel, la densité de lignes mesurée pour cette échelle et cette direction.

La figure 6.4 montre un exemple d'un sous-ensemble de telles cartes pour le modèle de la maison, dont l'arrangement initial de lignes peut être vu sur la figure 6.13. En plus des quatre pyramides directionnelles d'images, nous stockons une pyramide supplémentaire pour la densité *a priori* a-directionnelle qui encode les contributions de toutes les lignes, indépendamment de leur orientation.

Le rendu des cartes constituant les bases des pyramides utilise OpenGL : pour chacune des quatre directions, une couleur est attribuée à chaque segment de ligne, celle-ci reflétant son poids directionnel w_o mesuré par rapport à la direction considérée (*cf.* section 6.4.1) et ce segment est rendu en utilisant un mode de mélange additif. Les pyramides gaussiennes sont alors construites à partir de chacune des quatre cartes de base.

L'utilisation la plus directe de cette information est l'identification des régions denses d'un dessin, par rapport à une échelle et une orientation données. Cette information est disponible au travers de simples requêtes. De telles régions correspondent aux zones sombres sur la figure 6.4.

Le densité *a priori* pourrait déjà être utilisée pour contrôler de l'omission de lignes, par exemple en assignant à chaque trait une "probabilité d'omission" inversement proportionnelle à sa densité *a priori*, comme illustré par la figure 6.5. Cette technique permet effectivement de contrôler la densité finale moyenne, mais n'offre aucun contrôle fin sur le placement des traits, tel que celui fournit par la densité causale (*cf.* section 6.5).

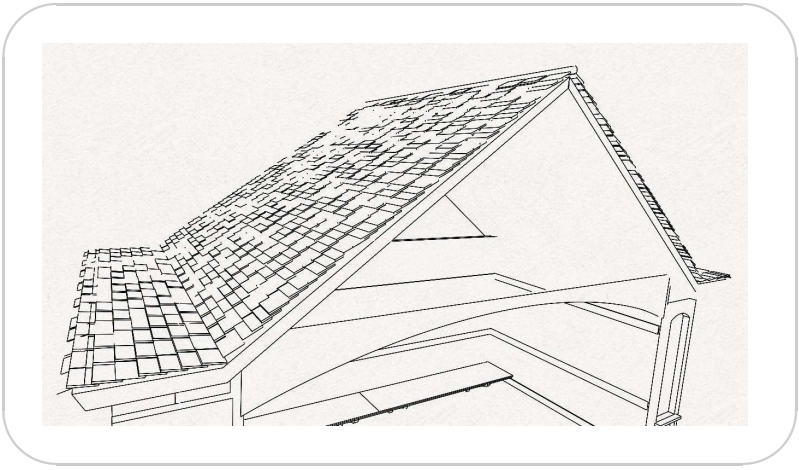


Fig. 6.5 – Simplification s'appuyant uniquement sur la densité *a priori*

La probabilité pour chaque ligne d'être éliminée est inversement proportionnelle à sa densité. On a ainsi un contrôle grossier sur la densité finale moyenne mais aucun contrôle fin sur le placement des traits, produisant cet aspect "aléatoire". La densité causale (section 6.5) est dédiée à cet objectif, comme le démontre la figure 6.10 par exemple.

6.4.3 La densité en fonction de l'espace, de l'échelle et de l'orientation

Nous montrons maintenant comment l'information orientée et multi-échelle contenue dans nos pyramides de densité *a priori* peut être utilisée pour analyser les structures et permettre des stratégies de simplification telles que l'indication. Dans ce but, nous nous inspirons des techniques d'analyse d'image et de motif. Comme on l'a vu dans la formule 6.1, la densité *a priori* est une fonction 3-paramétrée de l'espace (x,y), de l'échelle (σ) et de la direction ($\vec{u}$). D'importants aspects structurels de la complexité du dessin peuvent être révélés en étudiant les variations de la densité dans la dimension de chaque paramètre. Pour un meilleur aperçu, nous fixons deux des trois paramètres et étudions la réponse de la densité aux variations du troisième paramètre.

Frontières spatiales

Dans de nombreux schémas de simplification, les bords des régions denses reçoivent un traitement spécifique. En particulier, l'indication tend à être positionnée à ces endroits (*cf.* section 6.6). Les bords peuvent être identifiés en étudiant la dérivée spatiale de la densité $d_{\sigma_0,\vec{u_0}}(P)$, à une échelle donnée σ_0 et pour une direction donnée $\vec{u_0}$. De manière similaire à la détection de contours, on peut calculer l'image des gradients de la carte d'échelle σ_0 et de direction $\vec{u_0}$. Des schémas de détection plus avancés issus de l'analyse d'image peuvent aussi être utilisés.

Distribution directionnelle

Comme nous l'avons mentionné plus haut, différentes stratégies picturales peuvent être utilisées selon que la complexité d'une région dense est due à des lignes dirigées selon quelques directions privilégiées ou qu'aucune direction principale n'en ressort. On dit que la densité est *anisotropique* si une ou quelques directions dominent, *isotropique* sinon. Le degré d'isotropie peut être caractérisé en étudiant $d_{\sigma_0,P_0}(\vec{u})$, à une échelle σ_0 et pour un point P_0 donné. Sur la figure 6.6, on montre comment le profil de la fonction $d_{\sigma_0,P_0}(\vec{u})$ varie en fonction de l'arrangement des lignes avoisinantes, permettant ainsi de déterminer si une densité est isotropique ou anisotropique. En particulier, nous mettons en évidence les différences entre deux profils anisotropiques mesurés pour des pixels du toit, et les mettons en opposition avec un profil isotropique observé pour un pixel de l'arbre.

Échelle

Lorsqu'on travaille avec des structures répétitives, il est souvent important de caractériser l'échelle du motif constituant cette structure. C'est également crucial dans notre cas parce que les lignes sont des Diracs, et que l'échelle à laquelle la densité est évaluée peut avoir une forte influence sur le résultat : à la plus petite échelle, l'image est une fonction binaire qui indique la présence de lignes. Nous montrons qu'une analyse fine de l'échelle peut être menée de manière à permettre une sélection pécise des lignes à omettre. On peut par exemple, dessiner un toit avec une tuile sur deux, ou avec quelques groupes de n tuiles. Pour celà il est essentiel de posséder une approche systématique pour extraire l'échelle caractéristique de manière à rendre le critère de simplification indépendant de la scène.

L'échelle de régions denses et répétitives peut être caractérisée en étudiant $d_{P_0,\vec{u_0}}(\sigma)$ en un point P_0 donné et pour une direction $\vec{u_0}$ donnée. Dans le cas d'une région complexe faite

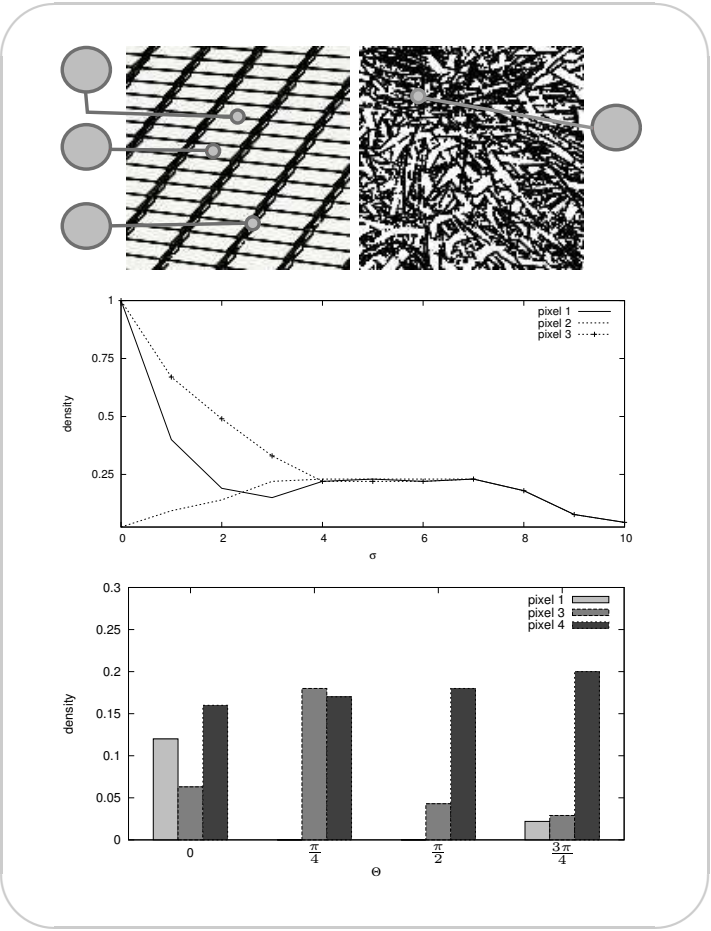


FIG. 6.6 – Profils de la densité *a priori* selon l'échelle et la direction

Haut : quatre pixels choisis à des positions particulières dans les images du toit et de l'arbre. Milieu : Le profil d'échelle de la densité a priori *pour les pixels 1,2,3. Bien qu'ayant des densités très différentes à basse échelle, les trois courbes convergent vers un plateau à $\sigma = 4$. Bas : Profils directionnels pour les pixels 1,3,4. Les deux premiers profils (provenant de l'image du toit) caractérisent une densité anisotropique importante selon la direction horizontale et $\frac{\pi}{4}$. Le dernier profil (provenant de l'image de l'arbre) correspond par contre à une haute densité isotropique.*

de petits motifs répétitifs, cette fonction contient un plateau : puisque notre estimateur est normalisé par l'aire de sa région support (*cf.* section 6.4.1), il donne le même résultat à multiples échelles pour les structures périodiques, dès que l'échelle est plus grande que la taille du motif et tant qu'elle est plus petite que la distance au bord de la région. Ainsi, la taille d'un motif peut être inférée à partir de la valeur de σ à laquelle ce plateau commence (ainsi que la fréquence de la distribution de lignes dans cette région). De même la distance au bord de la structure peut être obtenue à partir de la valeur σ à laquelle le plateau se termine. La figure 6.6 montre comment les profils d'échelle, pour trois points différents du toit du modèle de la maison, convergent vers un tel plateau, bien qu'étant très distincts pour les petites échelles. Sur la figure 6.10, nous utilisons cette propriété pour extraire la taille d'une tuile de manière à contrôler de l'omission de lignes à deux échelles différentes. Plus de détails à propos de cette

illustration sont donnés à la section 6.6.

Comme pour les autres informations, la densité, que nous avons jusqu'ici étudiée dans le contexte de requêtes 0-dimensionelles, peut être agrégée le long d'un élément 1-dimensionel en utilisant les mêmes outils statistiques (*cf.* section 4.4.3).

Résumé: Nous avons défini un estimateur pour la densité *a priori*, qui est calculée sur l'ensemble complet des lignes. Cet estimateur permet de mesurer la densité en un point, dans une direction et pour une échelle données. Ainsi, cet estimateur est un outil prédictif puissant quant à l'organisation des structures complexes qui apparaîtront potentiellement dans le dessin. Il permet donc de définir des stratégies de simplification qui anticipent sur cette complexité.

6.5 Densité causale

La densité *a priori* discutée jusqu'ici permet l'analyse du dessin qu'on aurait si toutes les lignes étaient dessinées. Elle fournit des outils d'analyse puissants, mais malheureusement, elle ne fournit pas de contrôle fin sur l'apparence finale du dessin, comme l'illustre la figure 6.5. En particulier, elle ne peut pas garantir que la densité du dessin courant n'excède pas un seuil donné, ni qu'aucune paire de traits est trop proche dans l'image. Pour cette raison, nous utilisons également une densité *causale* qui complémente la densité *a priori* et reflète l'état courant du dessin. Elle est mise à jour après le dessin de chaque trait et peut être utilisée pour styliser ou pour décider d'omettre les traits suivants.

6.5.1 Un estimateur de densité de traits

L'estimateur de densité causale travaille sur l'arrangement des traits rendus dans le dessin courant. Nous choisissons d'utiliser la fonction gaussienne normalisée standard, d'écart type σ, convoluée avec l'image de la luminance I du dessin comme estimateur.

De manière similaire à la définition de la densité *a priori* dans l'équation 6.1, l'estimation de la densité des traits dans l'image I en un point Q peut s'écrire comme :

$$d(Q,\sigma) = \int_{P \in I} w_d(P,Q,\sigma) I(P)\, dP \tag{6.4}$$

où w_d est la fonction gaussienne définie par la formule 6.2. Cet estimateur indique à quel point le dessin est localement "sombre" à l'échelle définie par σ. Chaque trait ajouté contribue à "assombrir" l'image d'une quantité qui dépend de sa couleur, de sa taille, de son épaisseur et de l'échelle σ.

De même que pour l'estimateur de densité de lignes, nous utilisons une version normalisée de la fonction gaussienne de manière à ce que la densité moyenne de traits du dessin ne dépende pas de l'échelle à laquelle ont été faites les requêtes.

Dans notre approche, la densité causale peut être évaluée à plusieurs échelles. Toutefois, parce que la densité causale est rafraîchie après chaque trait, nous n'avons pas trouvé bénéfique

de la stocker dans une pyramide. Les requêtes sont implémentées en intégrant l'information autour d'une région et les requêtes à large échelle sont plus coûteuses.

Pour des raisons de performance, nous avons choisi de ne pas encoder la densité causale pour de multiples orientations. Par opposition avec l'estimateur de densité de lignes, l'estimateur de densité de traits ne prend pas la directionnalité en compte. Néanmoins, une dépendance directionnelle peut être implémentée en utilisant l'information directionnelle fournie par la densité *a priori*.

6.5.2 Causalité et ordonnancement des traits

Comme l'avaient remarqué Winkenbach *et al.* [WS94] et Salisbury *et al.* [SWHS97], l'ordonnancement des traits est crucial lorsque l'omission de trait est utilisée pour contrôler le ton ou la densité de l'image finale. Notre seconde mesure de densité étant causale, les premiers traits ont plus d'influence sur le reste du dessin et il peut être fondamental de s'assurer que les traits les plus importants sont dessinés en premier. Par exemple, lorsqu'on utilise de l'omission de lignes, les lignes les plus importantes doivent être dessinées en premier pour minimiser leur chance d'être éliminés.

Cet ordonnancement dépend des choix stylistiques. Dans notre approche, il peut être défini à l'aide de l'opérateur de tri en s'appuyant sur l'ensemble des informations défini au chapitre 4 et en particulier sur la densité *a priori*. La figure 6.7 montre deux versions simplifiées obtenues en utilisant l'opérateur de sélection pour décider d'omettre des traits ou non en fonction de la valeur de la densité causale. Pour la première, la plus haute priorité est attribuée aux traits marquant de fortes discontinuités en profondeur. Par contraste, pour la deuxième image, les traits sont dessinés dans un ordre aléatoire. On peut en particulier remarquer que dans la version ordonnée, les limites du cœur de la fleur sont plus prononcées et la forme des graines est mieux suggérée.

FIG. 6.7 – Ordonnancement des traits

Il est essentiel de trier les lignes avant d'utiliser la densité causale. (a) Toutes les lignes visibles. (b) Utilisation de la densité causale pour l'omission de lignes lorsque les lignes ont été triées selon la discontinuité en profondeur, et (c), sans ordonnancement.

6.5.3 Omission de traits, densité et régularité

Une large classe de stratégies de simplification est basée sur l'omission de lignes. Cette technique peut être implémentée en utilisant la densité causale dans le cadre d'une opération de sélection pour laquelle la densité mesurée pour chaque trait est comparée avec un seuil τ afin de décider de dessiner ce trait ou non. L'algorithme naïf correspondant est :

pour chaque trait t
  d = densité du dessin sous t à l'échelle σ
 si d < τ
  dessiner(t)

Remarque : En pratique, ce test sera encapsulé sous forme d'un prédicat qui sera passé comme argument à l'opérateur de sélection.

Deux paramètres contrôlent cet algorithme : l'échelle σ et le seuil τ. Le dessin résultant peut être qualifié en deux termes, la *densité finale* moyenne (intuitivement le nombre de lignes dessinées) et la *régularité* de la distribution des lignes.

L'objectif de la simplification est d'obtenir une densité plus faible que celle initiale. L'ensemble final des traits peut être vu comme un sous-ensemble des lignes initiales. Comme avec n'importe quelle approche d'échantillonnage, la notion de régularité est importante pour caractériser à quel point la distribution finale est spatialement uniforme. Pour un seuil de densité donné τ, testé à une échelle σ, différents dessins peuvent être obtenus à partir du même arrangement de lignes, en fonction de l'ordre dans lequel les lignes sont traitées. Un dessin où les lignes apparaissent plus uniformément espacées est dit être plus régulier qu'un exhibant des groupes de lignes serrées.

La figure 6.8 illustre cette propriété. Cette notion de régularité est reliée au spectre de Fourier de motifs d'échantillonnage non uniforme utilisé pour l'*antialiasing* [MAZD85].

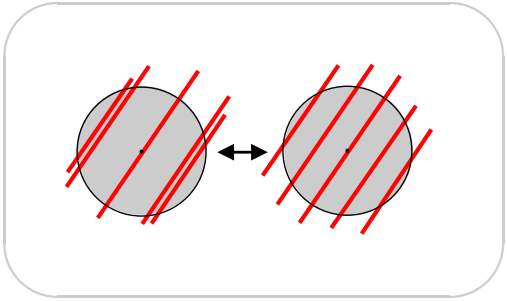


FIG. 6.8 – Régularité dans la distribution de lignes

Malgré la différence dans les distributions des lignes observées sur ces deux images, la densité (calculée à l'échelle du cercle gris en son centre) est la même. La distribution des lignes de l'image de droite est dite plus régulière *que celle des lignes de l'image de gauche.*

Les effets des deux paramètres σ et τ sur le résultat, c'est-à-dire la densité finale moyenne et la régularité du dessin, sont importants et méritent d'être discutés. Considérons une situation où un simple seuil τ est utilisé dans le cadre de requêtes de densité causale à une échelle σ.

Le tableau 6.1 résume les dépendances entre ces différentes variables en montrant comment la régularité et la densité finale moyenne varient lorsque, soit le seuil τ, soit l'échelle σ augmentent.

	$test : d < \tau$		$test : d < \frac{\tau}{\sigma^2}$	
paramètre	régularité	densité	regularité	densité
τ ↗	↘	↗	↘	↗
σ ↗	↘	–	–	↘

TAB. 6.1 – Régularité et densité finale moyenne : effet des paramètres σ et τ

Nous discutons de manière informelle les résultats présentés dans la dernière ligne du tableau 6.1, qui sont obtenus lorsque σ augmente tandis que τ reste constant. Il est tout d'abord important de noter que, puisque nous utilisons une version normalisée de la densité, l'estimation de celle-ci ne change pas avec σ lorsque l'échelle est au-dessus de l'échelle caractéristique d'une structure répétitive (*cf.* section 6.4.3).

Nous observons que lorsque l'échelle augmente, la position précise des lignes est moins contrainte, menant à des ensembles de lignes plus irréguliers. En effet, le seuil τ exerce à présent son influence sur des voisinages plus grands définis par σ. Cet effet est illustré sur la figure 6.9.

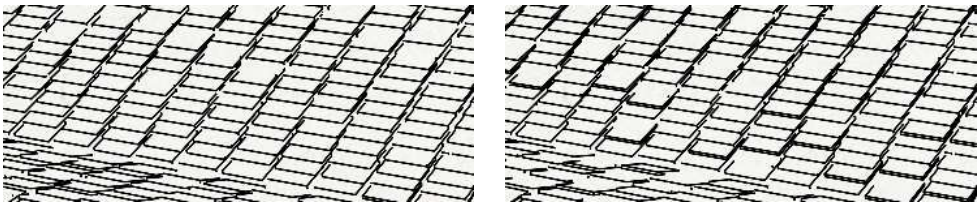


FIG. 6.9 – Influence de l'augmentation de σ sur la régularité de la distribution des lignes.
Droite : $\sigma = 2$, $\tau = 0.2$. Gauche : $\sigma = 7$, $\tau = 0.2$. La densité finale est la même pour les deux illustrations, mais seule l'image de droite contient des paires de lignes rapprochées.

De l'autre côté, si le seuil est inversement proportionnel à σ^2, ce qui revient à travailler avec une version non normalisée de la fonction gaussienne, alors augmenter σ diminue la densité finale du dessin, puisque le nombre de traits permis dans une région donnée est le même quelque soit la taille de cette région. Dans ce cas, la régularité reste globalement constante, puisque diminuer le nombre de traits autorisés sur une aire donnée réduit les chances d'encombrement pour cette région.

Le résultat attendu à l'issue d'une simplification basée sur l'omission de lignes consiste souvent à avoir une certaine quantité d'espace libre autour de chaque ligne, ce qui est obtenu en choisissant une échelle correspondant à l'espace minimum entre deux traits ainsi qu'un seuil bas.

Résumé: La densité *causale* est mesurée sur le dessin, au fur et à mesure que les traits sont ajoutés, et permet, en particulier, un contrôle fin sur la densité des traits dans ce dessin lorsqu'elle est utilisée pour diriger de l'omission de traits. La causalité de cet information implique de pouvoir trier les traits, de manière à afficher les plus importants en premier.

6.6 Applications et résultats

Nous présentons ici les résultats obtenus à la suite des expérimentations menées à l'aide de notre système programmable et spécifiquement liés à la simplification de dessins au trait à l'aide des mesures de densité. Ces résultats démontrent que les deux densités, causale et *a priori*, se complémentent l'une l'autre et permettent l'implémentation de stratégies de simplification automatiques. Des résultats plus généraux (certains utilisant également la densité) sont présentés dans le chapitre 7.

Les temps de calcul requis pour générer les illustrations de ce chapitre varient entre quelques secondes et quelques minutes : utiliser la densité *a priori* est rapide grâce au précalcul des cartes de densité, en revanche, la densité causale est évaluée sur demande chaque fois qu'un trait doit être dessiné, et est, par conséquent, l'opération la plus coûteuse (en particulier à large échelle).

Élagage uniforme

Nous commençons par décrire une stratégie de simplification où nous soulignons la régularité et omettons les lignes uniformément.

Dans ces exemples, nous décidons de considérer l'uniformité dans l'espace objet de manière à préserver la régularité de la scène.

La figure 6.10 montre deux niveaux d'élagage uniforme sur le toit à tuiles d'une maison. Dans les deux cas, les lignes qui appartiennent à de grandes régions visuellement complexes, sont, dans un premier temps, identifiées à l'aide de la carte de densité *a priori* a-directionnelle, pour l'échelle $\sigma = 4$. Ensuite, à partir du profil d'échelle (section 6.4.3) de la densité *a priori*, nous initialisons le noyau de l'estimateur de densité causale de manière à correspondre à environ deux fois la taille d'une tuile pour l'image de gauche et deux fois plus pour celle du milieu. Ainsi, les images sont simplifiées uniformément en gardant respectivement toutes ou la moitié des tuiles.

La figure 6.11 démontre le bon contrôle sur l'omission de lignes en simulant l'effet 3D de perspective au travers de requêtes multi-échelle de la densité causale ; notre but est de dessiner une version simplifiée (haut) d'une grille (bas) de la manière la plus naturelle : chaque barreau est représenté avec une seule ligne au lieu de deux, et seulement la moitié des barreaux verticaux sont dessinés. Les barreaux sont distribués uniformément le long de la grille, avec un espacement décroissant à mesure que l'on s'éloigne du point de vue, de manière à respecter l'impression de perspective. Pour ce faire, les lignes ont été soumises au test de la densité causale, la taille du noyau gaussien dépendant de la profondeur de la ligne considérée (la taille du noyau vaut approximativement deux fois l'espacement entre deux barreaux dans l'image simplifiée). De plus, nous nous appuyons sur l'information directionnelle fournie par

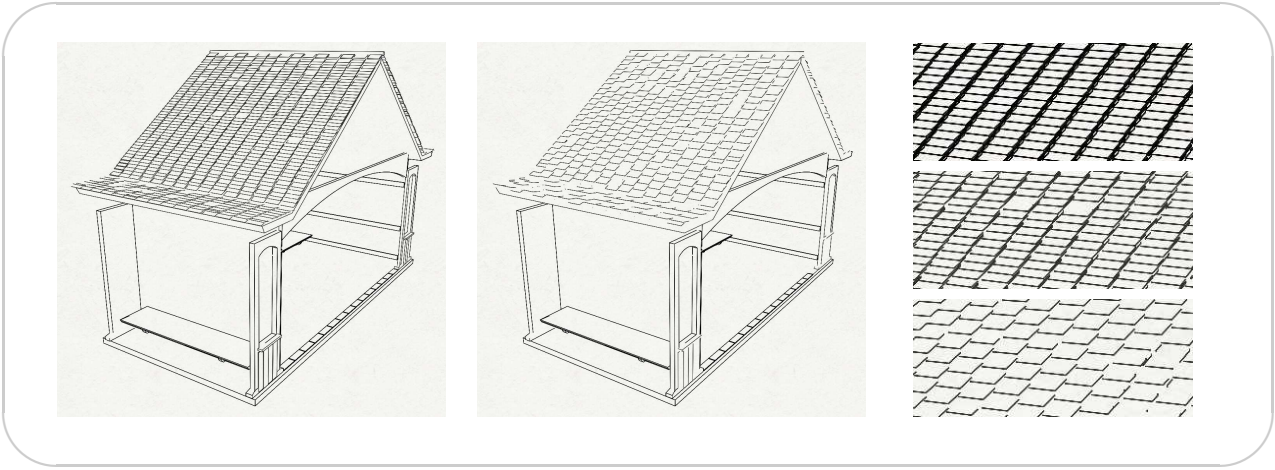


Fig. 6.10 – Élagage uniforme d'un toit à tuiles

À partir du profil d'échelle (section 6.4.3) de la densité a priori*, on infère la taille appropriée pour le noyau de l'estimateur de densité causale, de manière à dessiner un toit simplifié avec toutes les tuiles (gauche) ou la moitié des tuiles (milieu). La colonne de droite montre un zoom sur l'ensemble initial de lignes (haut), sur l'image de gauche (milieu), sur l'image du milieu (bas). On remarque que, bien que toutes les tuiles y soient dessinées, l'image du milieu contient une complexité moindre que l'image initiale.*

la densité *a priori* pour assigner des seuils plus hauts lorsque la densité *a priori* est faible dans la direction du trait courant. Ceci permet d'éviter l'omission systématique des lignes horizontales, dont la densité causale est haute du fait du nombre, potentiellement grand, de lignes verticales qui les croisent.

La figure 6.12 illustre l'utilisation des diverses mesures de densité pour contrôler l'encombrement mais préserver l'impression visuelle de masse. L'objet complexe comprend beaucoup de lignes presque parallèles (non nécessairement rectilignes) et nous utilisons abondamment de l'information de densité *a priori* directionnelle pour traiter les lignes en fonction de la présence ou non de voisins presque parallèles. Nous commençons par sélectionner les lignes qui sont entourées de lignes presque parallèles, c'est-à-dire ayant une haute densité anisotropique, en exploitant le profil directionnel de la densité *a priori* (section 6.4.3). Ces lignes sont ensuite triées par longueurs décroissantes, et dessinées en utilisant la densité causale pour diriger l'omission de lignes (en bas, à gauche). Dans une seconde passe, nous nous occupons des lignes appartenant à des régions denses isotropiquement (aucune direction privilégiée dans l'ensemble des lignes voisines) et éliminons celles appartenant aux régions les plus denses à grande échelle. On peut remarquer que le dessin est simplifié avec succès (l'image en bas à droite montre toutes les lignes qui sont totalement omises) tout en gardant une signification fidèle.

Indication

Comme nous avons pu le voir, cette stratégie de simplification exploite la répétition en suggérant la complexité globale à l'aide de quelques parties détaillées bien positionnées.

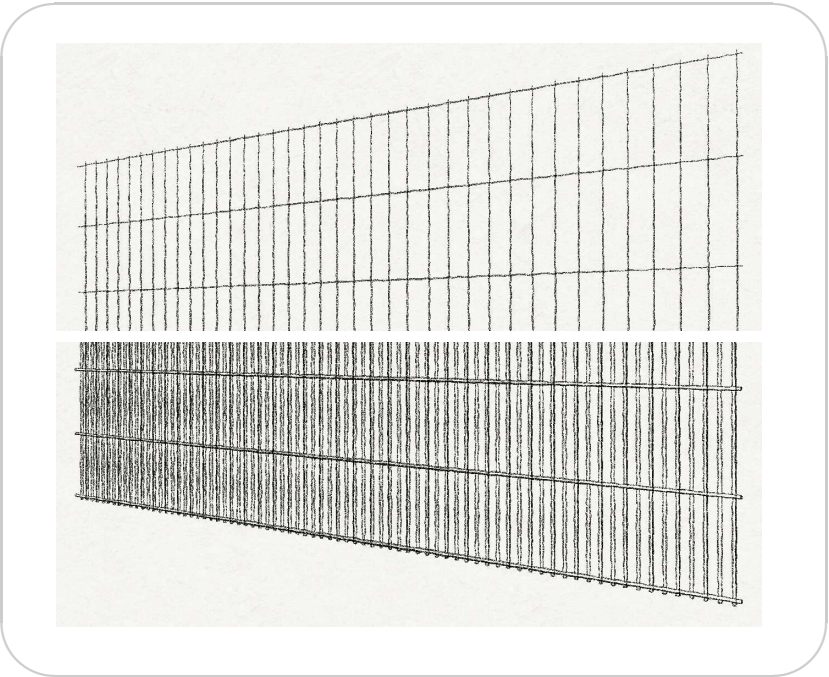


FIG. 6.11 – Simplification uniforme d'une grille

L'illustration simplifiée (haut) de la grille (bas) est obtenue en utilisant la densité causale. Chaque barreau est dessiné à l'aide d'une seule ligne et exactement la moitié des barreaux a été éliminée de manière uniforme. Les seuils d'élimination sont, en outre, dépendants de la direction, afin d'éviter l'omission systématique des lignes horizontales, dont la densité causale est plus haute.

Nous avons noté que ces indications sont plus utiles aux bords des régions denses et utilisons cette propriété pour produire les images simplifiées de la maison et de l'arbre, figure 6.13.

Sur l'illustration de la maison, nous commençons par sélectionner toutes les lignes appartenant à des larges régions de haute densité au travers de requêtes dans la carte a-directionnelle de densité *a priori*. La suite est identique pour les illustrations de la maison et de l'arbre.

Les lignes sont soumises à la densité causale avec un seuil d'omission proportionnel à la valeur du gradient calculé sur la carte a-directionnelle de densité *a priori*, à une échelle relativement petite. Ainsi, les lignes sont gardées de préférence dans les régions de fort gradient, c'est-à-dire proches des bords des régions de haute densité. La figure 6.14 montre la carte de densité *a priori* qui a été utilisée pour produire l'illustration de la maison et l'image de gradient correspondante.

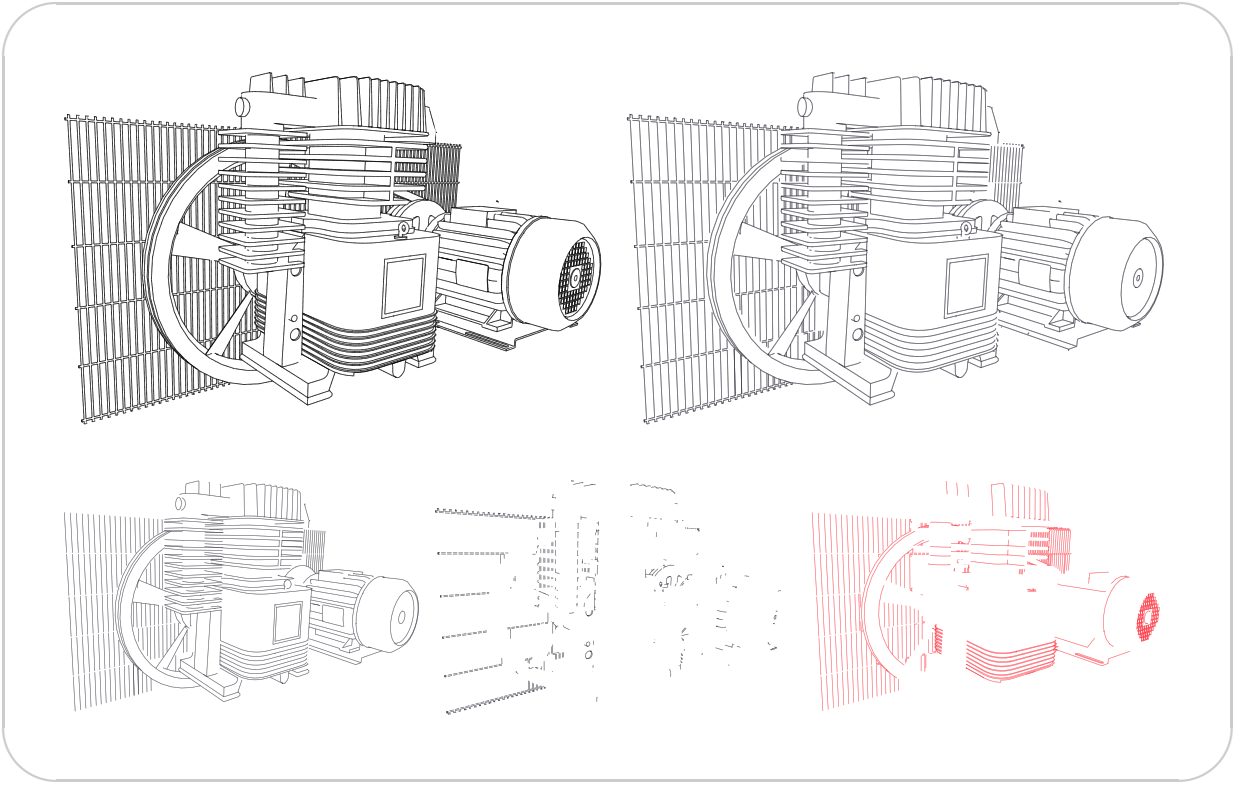


FIG. 6.12 – Utilisation du profil directionnel de la densité *a priori*

On utilise ici le profil directionnel de la densité a priori *pour simplifier l'encombrement du à des lignes presque parallèles.* **En haut, à gauche :** *toutes les lignes visibles.* **En haut, à droite :** *rendu final.* **En bas, à gauche :** *longues lignes à haute densité anisotropique.* **En bas, au centre :** *lignes courtes ou lignes à haute densité isotropique.* **En bas, à droite :** *lignes omises dans l'image finale.*

Résumé: Dans ce chapitre, nous avons introduit deux mesures de densité pour le dessin, pour contrôler et prévenir l'encombrement. La densité *a priori* est calculée une fois pour toute, avant le rendu, sur l'ensemble complet des lignes composant la vue. Elle peut être évaluée pour différentes positions, échelles et orientations et permet une analyse puissante de la structure locale. Par contraste, la densité *causale* est mise à jour au fur et à mesure que le dessin est créé, fournissant ainsi des informations à propos de l'état courant du dessin, de manière à offrir un contrôle plus fin sur le résultat final. Utilisées séparément, ces deux mesures fournissent des informations utiles pour diriger des stratégies de simplification et d'omission. Mais c'est en réalité lorsqu'elles sont exploitées ensemble qu'elles offrent un contrôle puissant et fin sur la complexité finale, la régularité, et le style du dessin. En particulier, nous avons montré qu'elles facilitent l'exploitation ou la mise en emphase de la régularité dans la vue en utilisant des stratégies picturales telles que l'élagage uniforme ou l'indication.

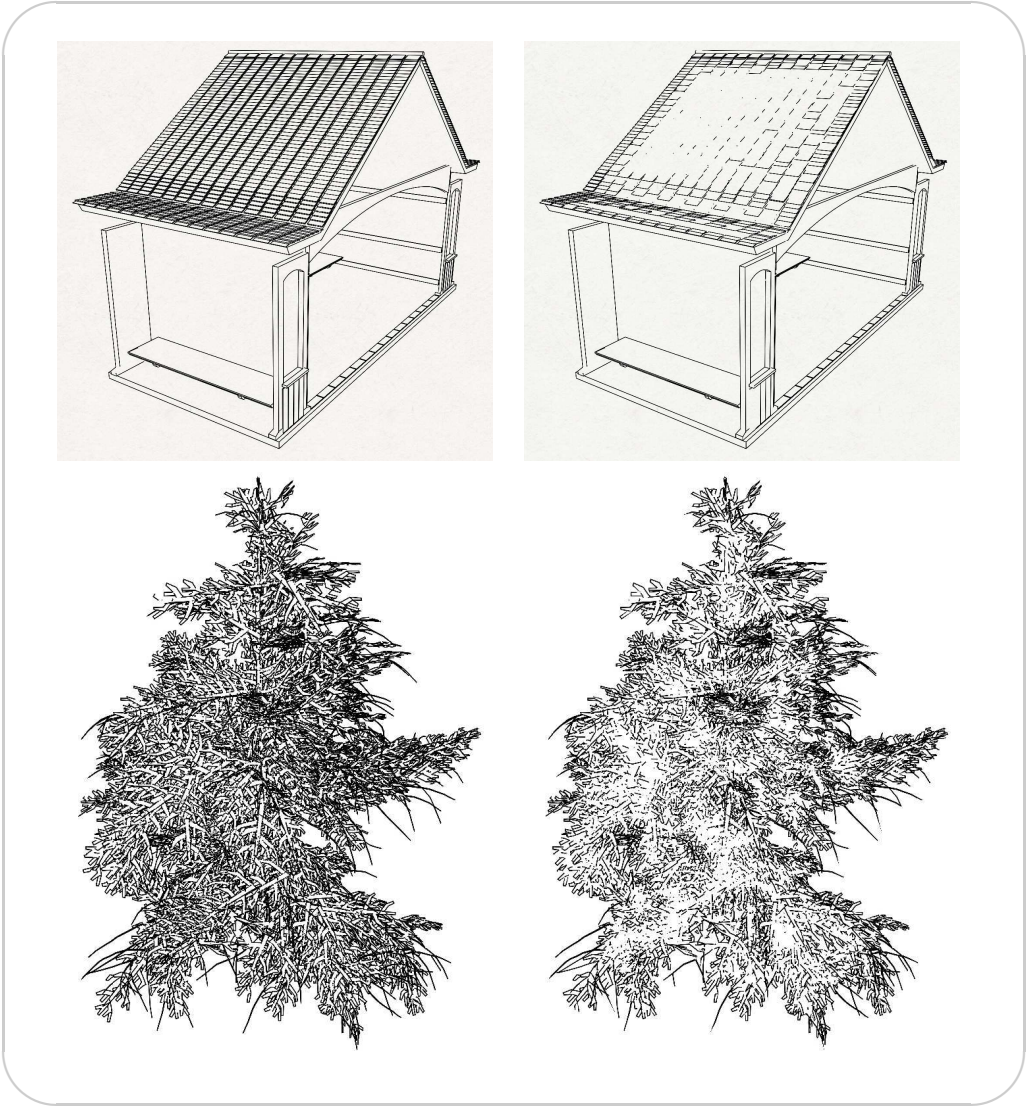


FIG. 6.13 – Simplification suivant la stratégie d'*indication*

Les versions initiale et simplifiée sont respectivement sur la gauche et sur la droite. L'objectif est de garder quelques régions complexes aux bords des régions visuellement denses pour suggérer leur complexité globale.

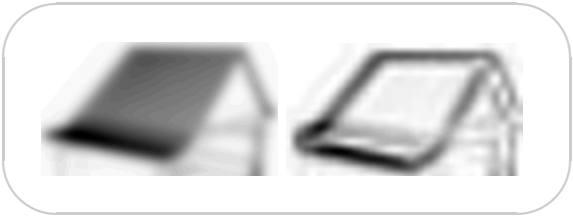


FIG. 6.14 – Image des gradients calculés sur la densité *a priori*

Ces cartes intermédiaires ont été utilisées pour produire l'illustration de la maison sur la figure 6.13. Gauche : la carte de densité a priori *a-directionnelle. Droite : L'image des gradients calculée sur cette carte. Cette image n'est pas stockée, les calculs de gradient étant en réalité faits au vol.*

chapitre 7

Résultats

Ces travaux de recherche ont mené à l'implémentation d'un système d'expérimentation développé suivant notre approche. Ce système est distribué sous la dénommination *Freestyle*, sous license GPL, et peut être téléchargé à l'adresse `http://freestyle.sourceforge.net`.

Dans ce chapitre, nous présentons les résultats obtenus avec ce système, lors de nos expérimentations. Nous démontrons notre approche en présentant ces résultats sous deux formes différentes. Premièrement, nous illustrons de manière plus concrète le processus de formulation d'un style en donnant un ensemble d'exemples de code source de modules de style ou de règles de style. Ces codes source sont des exemples du type de programmation que doit faire l'utilisateur. Ensuite, nous commentons des images générées en rendant des exemples de scènes 3D dans des styles variés. Pour chacune, les attributs, les informations et les règles de style les mettant en relation sont décrits.

7.1 Code source de règles de style

Avant de détailler un exemple de code source de module de style, nous montrons dans cette section, des exemples de code source pour chacune des règles auxquelles l'utilisateur a accès dans le cadre de la programmation d'un module de style, à savoir : les fonctions (0D/1D), les prédicats unaires (0D/1D), les prédicats binaires (1D), les itérateurs et les *shaders*. Tous ces exemples sont donc programmés en Python et utilisent, de manière plus ou moins importante, les mécanismes définis par notre système. Il est important de souligner que la programmation que nous allons voir, et qui est attendue de l'utilisateur, consiste la plupart du temps à surcharger un ou deux opérateurs tout au plus, et que de nombreux mécanismes rendent cette tâche relativement aisée.

7.1.1 Fonctions

Comme nous l'avons mentionné dans le chapitre 4, il existe deux grandes familles de fonctions : les fonctions 0D et les fonctions 1D, que nous illustrons chacune par un exemple de code.

Fonctions 0D

L'exemple de code que nous allons montrer ici correspond à l'implémentation Python d'une fonction 0-dimensionnelle. Ce type de programmation est typique de celle que doit

réaliser un utilisateur. En pratique, les fonctions 0D servent surtout à accéder à des informations sur des éléments 0-dimensionnels ; toutefois, il est courant pour l'utilisateur de définir ses propres fonctions 0D lorsque des requêtes plus complexes mettant en jeu plusieurs informations différentes sont nécessaires.

Il est important de prendre connaissance des deux ou trois grands principes relatifs à cette implémentation avant de l'étudier. D'abord, la classe *UnaryFunction0D* est la classe de base de tous les *functors* 0D, qu'ils soient définies en C++ ou en Python. En pratique, en Python, on trouve plusieurs *functors* de base en fonction du type de retour de la fonction. Ainsi, la classe *UnaryFunction0DDouble* est la classe de base de tous les *functors* 0D qui renvoient un réel. Ensuite, pour définir le comportement de sa propre fonction 0D, deux méthodes doivent être surchargées :

La méthode *__init__* Cette méthode correspond au constructeur du *functor* et contient toutes les initialisations nécessaires.

La méthode *__call__* C'est l'équivalent de l'opérateur () du C++, dont la surcharge est le propre des *functors* dans la mesure où elle permet de simuler l'utilisation d'une fonction (à partir d'un objet). L'élément passé en argument à cette méthode est un point, encapsulé dans un itérateur (cet itérateur permet de parcourir les points d'une chaîne 1D et sert à connaître, si besoin est, le contexte 1D auquel appartient le point, *cf.* chapitre 4 pour plus de détails à ce sujet). Cette méthode contient les instructions à appliquer à cet itérateur et qui définissent le comportement de la fonction.

Finalement, pour faciliter la lecture des exemples de code, nous faisons quelques rappels sur les bases de Python :

- Python est un langage objet,
- les variables ne sont pas typées à la déclaration (mais dynamiquement),
- il n' a pas de pointeur et les champs d'une classe sont accédés avec l'opérateur ".",
- le mot-clef *self* est équivalent au *this* du C++ et est requis pour accéder à une donnée membre d'une classe, y compris dans la définition d'une méthode de cette même classe,
- le groupement d'instructions est uniquement défini par l'indentation (pas de "{}" comme en C++)

Nous considérons maintenant l'exemple de code suivant :

```
## estimateur d'anisotropie de la densite causale
class pyDensityAnisotropyF0D(UnaryFunction0DDouble):
    ## level : l'echelle a laquelle la requete de densite est faite
    def __init__(self,level):
        ## Appel au constructeur de la classe de base
        UnaryFunction0DDouble.__init__(self)
        ## Construction des functors 0D de requete de densite a-priori
        ## pour les quatre directions et dans la carte a-directionnelle.
        self._IsoDensity = ReadCompleteViewMapPixelF0D(level)
        self._d0Density  = ReadSteerableViewMapPixelF0D(0, level)
        self._d1Density  = ReadSteerableViewMapPixelF0D(1, level)
        self._d2Density  = ReadSteerableViewMapPixelF0D(2, level)
        self._d3Density  = ReadSteerableViewMapPixelF0D(3, level)

    ## Surcharge de l'operateur ()
    ##  iter : iterateur sur les points de la chaine
    def __call__(self, iter):
        c_iso = self._IsoDensity(iter)
```

```
        c_0 = self._d0Density(iter)
        c_1 = self._d1Density(iter)
        c_2 = self._d2Density(iter)
        c_3 = self._d3Density(iter)
        cMax = max( max(c_0,c_1), max(c_2,c_3))
        cMin = min( min(c_0,c_1), min(c_2,c_3))
        if ( c_iso == 0 ):
            v = 0
        else:
            v = (cMax-cMin)/c_iso
        return (v)
```

Il s'agit de la définition de la fonction 0D *pyDensityAnisotropyF0D* qui évalue l'anisotropie de la densité *a priori* (*cf.* chapitre 6) en un point. Ce *functor* dérive de la classe *UnaryFunction0DDouble* et retourne donc un réel.

À l'initialisation de cette fonction (dans la fonction *__init__*), cinq *functors* permettant de faire des requêtes 0D dans les différentes cartes de densité *a priori*, sont instanciés. On les instancie à l'initialisation pour pouvoir les utiliser par la suite dans la méthode *__call__*. Dans la surcharge de la méthode *__call__* (c'est-à-dire de l'opérateur ()), des requêtes sont faites dans les différentes cartes au point concerné, de manière à renvoyer une valeur entre 0 et 1 mesurant dans quelle mesure la densité *a priori* en ce point est anisotropique.

La plupart des règles que nous allons maintenant voir s'appuient sur les mêmes concepts.

Fonctions 1D

De même que pour les *functors* 0D, les *functors* 1D dérivent tous de la classe $UnaryFunction1D$, dont la méthode *__call__*, et éventuellement la méthode *__init__*, doivent être surchargées.

En outre, les fonctions 1D sont souvent construites comme une série d'appels à la fonction 0D correspondantes. Par exemple, une fonction 1D calculant la courbure moyenne d'un trait, fera un appel à la fonction 0D de calcul de courbure en chaque point du trait, et calculera ensuite la moyenne des valeurs obtenues. Pour cette raison, il est possible d'automatiser une telle implémentation qui, dans notre système, consistera souvent à utiliser les mécanismes standard d'agrégation (comme nous l'avions vu section 4.5, un certain nombre d'opérations standard, telles que la moyenne ou le minimum, permettent de construire, à partir d'un ensemble de valeurs obtenues par requêtes 0D, une unique valeur, pour l'ensemble de l'élément 1D).

Ces mécanismes sont illustrés dans l'exemple de code suivant :

```
## Fonction 1D estimant l'anisotropie de l'ensemble des points d'un element 1D
class pyDensityAnisotropyF1D(UnaryFunction1DDouble):
    ## constructeur
    ##  level       : echelle de la requete a la densite
    ##  integrationType     : type d'agregation a utiliser
    ##  sampling        : distance entre deux points d'echantillonnage
    def __init__(self,level,  integrationType=MEAN, sampling=2.0):
        UnaryFunction1DDouble.__init__(self, integrationType)
        ## La fonction pour chaque requete 0D est instanciee
        self._func = pyDensityAnisotropyF0D(level)
        self._integration = integrationType
        self._sampling = sampling
```

```
    def __call__(self, inter):
        ## Appel au mecanisme standard d'agregation de plusieurs valeurs
        return integrateDouble( self._func,
                                inter.pointsBegin(self._sampling),
                                inter.pointsEnd(self._sampling),
                                self._integration)
```

Ce *functor*, *pyDensityAnisotropyF1D*, dérive de la classe *UnaryFunction1DDouble* qui est la classe de base de tous les *functors* 1D qui renvoient des réels. La classe de base des *functors* 1D prend en particulier en argument le type d'"intégration" (*e.g.* moyenne, maximum) à utiliser pour agréger les valeurs 0D (correspondant aux valeurs renvoyées par les appels de la fonction 0D correspondante en chaque point de l'élément 1D) ainsi qu'une valeur pour l'échantillonnage auquel faire ces requêtes 0D. Dans le constructeur, on instancie le *functor* 0D correspondant : *pyDensityAnisotropyF0D*, défini au-dessus. La méthode *__call__* prend en argument un élément 1D (qui peut être une arête du graphe de vue ou un trait) et consiste en un appel au mécanisme standard d'intégration mentionné précédemment. On peut remarquer que l'implémentation d'une telle fonction 1D à partir de la fonction 0D correspondante est relativement directe et rapide dans la mesure où elle peut largement bénéficier des mécanismes standard fournis par le système.

7.1.2 Prédicats

La deuxième famille de règles que nous illustrons dans ce chapitre sont les prédicats. Les prédicats sont des types spéciaux de *functors* qui ne renvoient que des booléens. Ils peuvent être unaires, s'ils ne prennent qu'un seul argument, ou binaires, s'ils prennent deux arguments, et travailler sur des éléments 0D (de type itérateur sur points d'un élément 1D) ou des éléments 1D. En général, un prédicat unaire sert à tester si une certaine condition est vérifiée sur un élément donné, tandis qu'un prédicat binaire sert à comparer deux éléments.

De même que pour les *functors*, il existe des classes de base pour chacun de ces types de prédicats, dont chacun d'entre eux doit dériver et le travail consiste à surcharger le constructeur *__init__* (si besoin est) et la méthode *__call__*.

Nous commençons par donner un exemple de prédicat unaire 0D.

Prédicats unaires 0D

Ces prédicats prennent en argument un unique élément de type itérateur sur points d'un élément 1D.

UnaryPredicate0D est la classe de base de tous les prédicats 0D.

```
## Predicat 0D pour tester si la courbure 2D en un point
## est plus grande qu'un seuil donne.
class pyHigherCurvature2DUP0D(UnaryPredicate0D):
    ## Constructeur
    ##  a : seuil sur la courbure
    def __init__(self,a):
        UnaryPredicate0D.__init__(self)
        # Le functor de requete sur la courbure est instancie
        self._func  = Curvature2DAngleF0D()
        self._a     = a
    ## Operateur ()
```

```
    ##  inter : element 1D
    def __call__(self, inter):
        # Si la courbure est plus grande que le seuil,
        # on retourne vrai.
        return ( self._func(inter) > self._a)
```

Ce prédicat utilise l'information de courbure ; pour ce faire, le *functor* de requête à cette information est instancié dans le constructeur. Ensuite, dans la méthode *__call__*, la valeur de la courbure en un point est comparée à un seuil : si la courbure est plus grande que ce seuil, le prédicat renvoie *vrai*, *faux* sinon.

Prédicats unaires 1D

Ces prédicats prennent en argument un unique élément de type élément 1D.

UnaryPredicate1D est la classe de base de tous les prédicats 1D.

```
## Predicat 1D testant si un element 1D est d'une
## certaine nature
class pyNatureUP1D(UnaryPredicate1D):
    ## Constructeur
    def __init__(self,nature):
        UnaryPredicate1D.__init__(self)
        # On instancie le functor 1D associe a ce predicat
        self._func   = CurveNatureF1D()
        self._nature     = nature

    ## Surcharge de l'operateur ()
    ##  inter : element 1D
    def __call__(self, inter):
        if(self._func(inter) & self._nature):
            return 1
        return 0
```

Dans cet exemple, le prédicat utilise le *functor* 1D *CurveNatureF1D* pour évaluer la nature d'un élément 1D. Si, pour l'élément passé en argument, cette nature inclut notamment la nature passée au constructeur, le prédicat renvoie *vrai*.

Prédicats binaires 1D

Un tel prédicat binaire prend en argument deux éléments 1D et est la plupart du temps utilisé comme opérateur de comparaison.

La classe de base de tous les prédicats binaires 1D est *BinaryPredicate1D*.

Le prédicat binaire montré en exemple compare les profondeurs moyennes de deux éléments 1D. Lorsqu'il est utilisé avec ce prédicat, l'opérateur de tri place les éléments les plus proches en premier dans la séquence.

```
## Predicat de comparaison des profondeurs moyennes de deux aretes
class pyZBP1D(BinaryPredicate1D):
    ## Constructeur
    def __init__(self):
        BinaryPredicate1D.__init__(self)
        # On instancie le functor 1D d'evaluation de la profondeur
        # (par defaut le type d'integration est la moyenne)
```

```
        self._func = GetZF1D()
    ## Surcharge de l'operateur ()
    ##  i1 : premier element 1D
    ##  i2 : deuxieme element 1D
    def __call__(self, i1, i2):
        return (self._func(i1) < self._func(i2))
```

7.1.3 Itérateurs de chaînage

le troisième type de règle que nous considérons sont les itérateurs de chaînage. Comme nous l'avons expliqué dans le chapitre 5, ces itérateurs servent à définir la topologie d'un trait en spécifiant quel chemin suivre dans le graphe de vue, en partant d'une arête donnée. L'utilisateur spécifie un tel chemin en surchargeant la fonction d'itération de l'itérateur afin d'indiquer quelle arête choisir parmi les arêtes candidates.

Les iterateurs de chaînage dérivent de la classe de base *ChainingIterator*. Le travail de l'utilisateur consiste ici à surcharger le constructeur *__init__*, si besoin est, une fonction *init* (différente du constructeur) d'initialisation de chaînage, qui est appelée avant de commencer chaque nouvelle chaîne et la fonction *traverse*, qui définit l'arête à choisir parmi les candidats pour continuer le chaînage.

En outre, le constructeur de cette classe prend en argument deux booléens indiquant respectivement s'il faut chaîner en restant dans la sélection ou non et si les arêtes déjà chaînées restent de potentielles candidates.

Cet exemple de code montre un itérateur de chaînage permettant de produire des styles "sketchy" en chaînant plusieurs fois la même arête. Un mécanisme de *time stamp* est utilisé pour gérer le nombre de passages par une même arête. En particulier, dans sa méthode *init()*, un *time stamp* est initialisé de manière à n'autoriser que trois chaînages de la même arête. Dans la méthode *traverse()*, on parcourt l'ensemble des arêtes candidates et une arête est arbitrairement choisie pour poursuivre la chaîne. Parmi les arêtes candidates, seules les arêtes ayant déjà été chaînées plus du nombre de fois autorisé sont rejetées.

```
## Iterateur de chainage produisant un style "sketchy".
## Cet iterateur peut chainer plusieurs fois la meme arete
## du graphe de vue de maniere a produire des traits multiples
## par arete.
## Sinon, le choix de l'arete a suivre est arbitraire.
class pySketchyChainingIterator(ChainingIterator):
    ## Constructeur
    ##  nRounds          : le nombre de fois qu'une arete peut etre chainee.
    ##  stayInSelection      : indique si le chainage peut s'etendre en dehors de
    ##                 de la selection.
    def __init__(self, nRounds=3, stayInSelection=1):
        # On autorise le chainage a chainer plusieurs fois la meme arete (0)
        ChainingIterator.__init__(self, stayInSelection, 0,None,1)
        self._timeStamp = GetTimeStampCF()+nRounds
    ## Fonction d'initialisation du chainage
    def init(self):
        # On initialise la valeur du time stamp de maniere
        # a chainer chaque arete nRounds fois au max.
        self._timeStamp = GetTimeStampCF()+nRounds
    ## Fonction d'iteration sur les aretes du graphe
```

```
##  iter : Iterateur sur les aretes adjacente au sommet courant
def traverse(self, iter):
    winner = None
    # AdjacencyIterator : type de iter
    it = AdjacencyIterator(iter)
    # On itere sur les aretes candidates pour choisir la prochaine
    while(it.isEnd() == 0):
        ve = it.getObject()
        # L'arete sur laquelle on se trouve est ecartee :
        if(ve.getId() == self.getCurrentEdge().getId()):
            it.increment()
            continue
        winner = ve
        it.increment()
    # Si on n'a rien trouve, on repasse par la meme
    if(winner == None):
        winner = self.getCurrentEdge()
    # Si l'arete a deja ete chainee nRounds fois, on arete la chaine.
    if(winner.getChainingTimeStamp() == self._timeStamp):
        return None
    return winner
```

On remarque, dans la méthode *traverse*, l'utilisation d'un autre itérateur, *AdjacencyIterator* (*cf.* section 5.4.2). Il s'agit d'un mécanisme standard permettant d'itérer sur les arêtes candidates à poursuivre la chaîne. Cet itérateur encapsule un certain nombre de contraintes de manière à faciliter l'implémentation de l'itération : par exemple, si l'utilisateur a spécifié que le chaînage devait rester dans la sélection, seules les arêtes de la sélection seront parcourues par l'itérateur d'adjacence.

7.1.4 *Shaders*

La dernière règle qu'il nous reste à illustrer est celles des *shaders*.

La classe de base pour les *shaders* est *StrokeShader*. L'utilisateur doit ici surcharger le constructeur *__init__* (si besoin est) et la fonction *shade*. Cette dernière prend en argument un trait et peut en modifier les attributs. La structure classique de cette fonction est une boucle sur les sommets du trait dans laquelle les attributs associés à chaque sommet sont modifiés.

> **Remarque :** L'itérateur sur les points d'un trait est de type *StrokeVertexIterator*. C'est un type spécialisé permettant d'accéder plus facilement aux attributs spécifiques des sommets de trait. Pour des raisons d'implémentation, ce type n'est pas une spécialisation du type de base utilisé pour l'ensemble des itérateurs sur points (*InterfaceODIterator*). Pour pouvoir faire une requête d'information 0D (à l'aide des fonctions 0D définies par le système) sur un objet de type *StrokeVertexIterator*, il est donc nécessaire d'invoquer une méthode de typage (*castToInterface0DIterator()*).

Cet exemple est un *shader* qui affecte à chaque point d'un trait une épaisseur inversement proportionnelle à la valeur de la profondeur de ce point.

```
## Shader assignant une epaisseur fonction de la profondeur
class pyZThicknessShader(StrokeShader):
```

```
## Constructeur
##  min : epaisseur min (pour z = infini)
##  max : epaisseur max (pour z = 0)
def __init__(self, min, max):
    StrokeShader.__init__(self)
    self._min = min
    self._max = max
    # On instancie le functor de requete de profondeur
    self._func = GetProjectedZF0D()
## On surcharge la methode shade()
##  stroke : un trait
def shade(self, stroke):
    # it est du type StrokeVertexIterator
    it = stroke.strokeVerticesBegin()
    while it.isEnd() == 0:
        # it doit etre caste pour etre passe en argument d'une function 0D
        z = self._func(it.castToInterface0DIterator())
        # On calcule l'epaisseur en fonction du z
        thickness = (1 - z) * self._max + z * self._min
        # v est le sommet pointe par l'iterateur (getObject() est l'operateur
        # de dereferencement)
        v = it.getObject()
        # attribute() renvoie l'attribue associe au sommet v
        v.attribute().setThickness(thickness, thickness)
        it.increment()
```

Remarque : En Python, il n'est pas possible de déréférencer un itérateur en invoquant l'opérateur "*", comme on le ferait en C++. Dans notre système, c'est la méthode *getObject* qui joue ce rôle.

7.2 Style simple

Maintenant que nous nous sommes familiarisés avec l'allure du code pour chaque règle, c'est-à-dire avec les briques de base, nous allons regarder un exemple complet, constitué de trois modules de style, de manière à acquérir une vision d'ensemble de la formulation d'un style.

C'est le style que nous avons eu l'occasion d'utiliser plusieurs fois précédemment (*cf.* figure 5.1 page 116) pour illustrer notre approche qui nous sert ici d'exemple. Nous utilisons la scène montrée sur la figure 7.1 (a) pour montrer le résultat de l'application de chaque module de style et, également, du style complet (figure 7.1 (b)).

Dans les exemples de code qui suivent, les règles qui ont été programmées par l'utilisateur sont préfixées par "py" (pour Python). Tous les autres éléments sont des composants standard fournis par le système.

7.2.1 Premier module de style : lignes de construction

Ce module de style définit des traits correspondant à des lignes de construction (*cf.* figure 7.2).

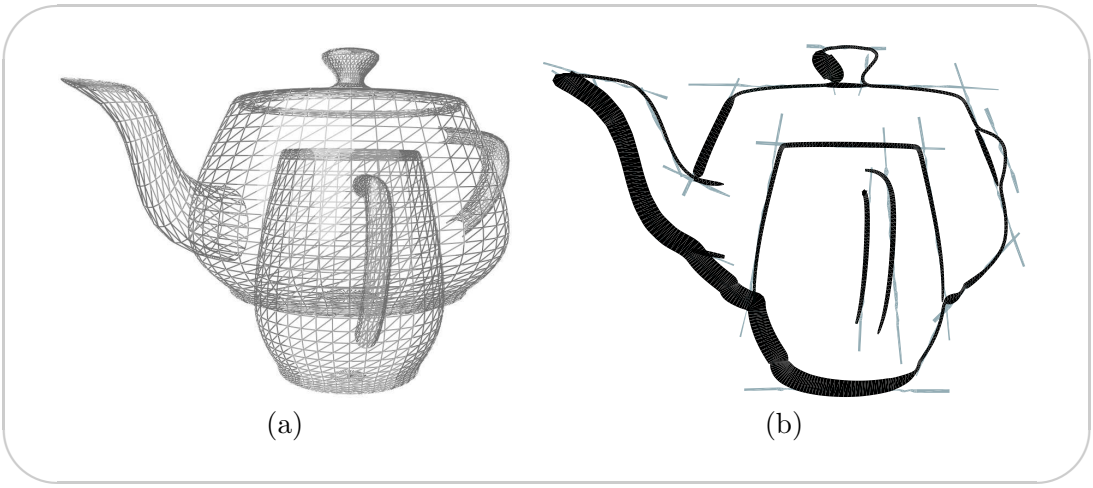


FIG. 7.1 – Exemple de style simple

(a) : La scène 3D. (b) : Le dessin final résultant de la combinaison de trois modules de style.

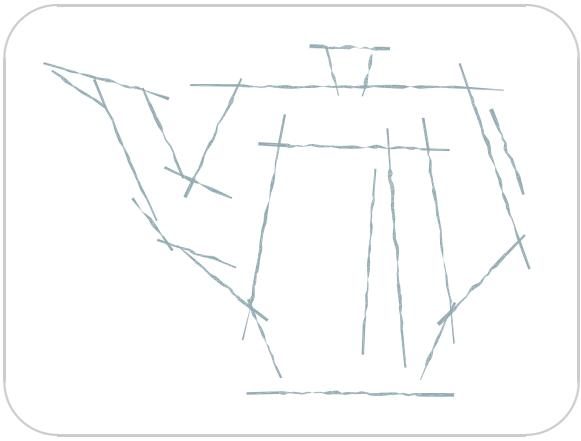


FIG. 7.2 – Premier module de style : lignes de construction

On commence par donner le code du module de style, puis les règles définies par l'utilisateur qui y apparaissent.

Corps du module de style

```
## ////////////////////////////////////////////////////////
##           Module de style
## ////////////////////////////////////////////////////////
##-- Selectionne les aretes visibles du graphe de vue
Operators.select(QuantitativeInvisibilityUP1D(0)) {\textcolor{red}{(1)}}
```

```
## -- Chaine en utilisant l'iterateur de chainage standard
## Par defaut le chainage reste dans la selection.
## On s'arrete lorsque l'arete trouvee n'est pas visible
Operators.bidirectionalChain(ChainSilhouetteIterator(),
                             NotUP1D(QuantitativeInvisibilityUP1D(0)))
## -- Decoupe recursivement aux points de plus haute courbure
## Les chaines sont coupees jusqu'a atteindre une certaine longueur
## Nous empechons les points de coupure d'etre proches des extremites
Operators.recursiveSplit(
            Curvature2DF0D(),## La fonction 0D a evaluer
            pyParameterUP0D(0.2,0.8),## Le predicat 0D qui preselectionne
                                     ## les points de coupure candidats
            NotUP1D(LengthHigherUP1D(75)),## Le predicat 1D d'arret de la recursion
            2)## Echantillonnage
## -- liste des shaders
shaders_list = [
        ConstantColorShader(0.6,0.6,0.6),## affecte une couleur constante
        ConstantThicknessShader(2.0),## affecte une epaisseur constante
        pyGuidingLineShader(),## Modifie la geometrie du trait selon
                              ## sa tangente
        SpatialNoiseShader(1, 30, 2, 1, 1),## bruit spatial
        ThicknessNoiseShader(2, 10),## bruit d'epaisseur
        BackboneStretcherShader(0.2)## stretche le trait
            ]
## -- cree les traits
Operators.create(TrueUP1D(),shaders_list)
```

On voit ainsi la structure d'un module de style qui correspond au pipeline d'opérateurs défini au chapitre 5 : on a ainsi une séquence d'appels aux opérateurs de sélection, de chaînage, de découpage récursif et de création. C'est dans ce dernier que l'affectation d'attributs, définie par la liste de *shaders*, *shaders_list*, est effectué. En particulier, le découpage récursif est utilisé en combinaison avec la fonction de courbure de manière à définir des traits commençant et s'arrêtant aux points de plus forte courbure. Le *shader* caractéristique de ce style déforme finalement la géométrie de chaque trait de manière à l'approximer par un trait rectiligne.

Règles utilisées

Les règles utilisées dans ce module de style, qui ont été programmées par l'utilisateur, et que nous allons détailler dans la suite, sont le prédicat unaire *pyParameterUP0D* et le *shader pyGuidingLineShader*.

Prédicat unaire 0D Ce prédicat teste à partir de son abscisse curviligne, la position d'un point sur une chaîne. Il peut ainsi permettre d'identifier les points se trouvant près des extrémités d'un trait.

```
## Predicat unaire 0D. Retourne vrai si le parametre u du trait appartient
## a un segment donne
## -- Derive de UnaryPredicat0D
class pyParameterUP0D(UnaryPredicate0D):
    ## Constructeur
    ##  pmin : valeur inferieure pour le segment
    ##  pmax : valeur superieure pour le segment
    def __init__(self,pmin, pmax):
```

```
        UnaryPredicate0D.__init__(self)
        self._m = pmin
        self._M = pmax
    ## Surcharge de l'operateur ()
    ##  iter : Interface0DIterator
    def __call__(self, iter):
        return ((iter.u()>=self._m) and (iter.u()<=self._M))
```

Shaders Ce *shader* est la version Python du *shader* de lignes de construction. Il déforme la géométrie du trait de manière à la projeter sur une approximation de sa tangente.

```
## Shader pour deplacer les sommets du trait le long
## de la tangente en son milieu
## -- Derive de StrokeShader
class pyGuidingLineShader(StrokeShader):
    ## methode shade
    ##  stroke : un trait
    def shade(self, stroke):
        it = stroke.strokeVerticesBegin()   ## Premier sommet
        itlast = stroke.strokeVerticesEnd() ##
        itlast.decrement()                  ## Dernier sommet
        ## t <- direction tangente
        t = itlast.getObject().getPoint() - it.getObject().getPoint()
        ## On cherche le sommet milieu du trait
        itmiddle = StrokeVertexIterator(it)
        while(itmiddle.getObject().u()<0.5):
            itmiddle.increment()
        ## On deplace tous les sommets sur la tangente
        while(it.isEnd() == 0):
            it.getObject().SetPoint(itmiddle.getObject().getPoint() \
            +t*(it.getObject().u()-itmiddle.getObject().u()))
            it.increment()
```

7.2.2 Deuxième module de style : contour extérieur calligraphique

Ce module de style définit des traits correspondant au contour extérieur de la scène (*cf.* figure 7.3). Il s'appuie principalement sur le prédicat standard permettant de savoir si un élément 1D appartient au contour extérieur, sur l'itérateur de chaînage *ChainPredicatIterator* et sur le *shader* calligraphique.

Corps du module de style

```
## ////////////////////////////////////////////////////////
##          Module de style
## ////////////////////////////////////////////////////////
## -- On instancie un predicat permettant de tester si un element 1D
## est visible et appartient au contour exterieur.
predicate1D =  AndUP1D(QuantitativeInvisibilityUP1D(0),ExternalContourUP1D())
## -- Selectionne les aretes visibles du contour exterieur
Operators.select(predicate1D)
## -- Chaine a l'aide de l'iterateur de chainage ChainPredicateIterator.
```

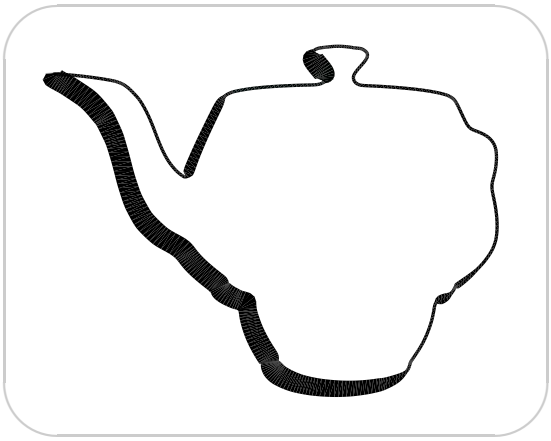


FIG. 7.3 – Deuxème module de style : contour extérieur calligraphique

```
## On chaine le long des aretes visibles du contour exterieur.
## Aucune contrainte ne porte sur la paire (arete courante - arete
## suivante). Ainsi, le predicat binaire de l'iterateur repond vrai
## en permanence (TrueBP1D).
Operators.bidirectionalChain(ChainPredicateIterator(predicate1D, TrueBP1D()),
                             NotUP1D(predicate1D))
## -- liste de shaders
shaders_list = [
        ConstantColorShader(0.2,0.2,0.2),## Affecte une couleur RGB constante
        ## Affecte une epaisseur calligraphique
        PyCalligraphicThicknessShader(6,          ## Epaisseur mininmale
                                      40,         ## Epaisseur maximale
                                      Vec2f(1, 1))## Orientation
                ]
## -- cree les traits
Operators.create(TrueUP1D(),shaders_list)
```

Règles utilisées

La seule règle codée par l'utilisateur est ici le *shader pyCalligraphicThicknessShader*.

Shaders Ce *shader* affecte une épaisseur calligraphique aux traits. Il s'appuie sur l'information de normale 2D pour calculer l'épaisseur appropriée en chaque point.

```
## Affecte une epaisseur calligraphique aux traits.
## -- Derive de StrokeShader
class pyCalligraphicThicknessShader(StrokeShader):
    ## Constructeur
    ##  m : epaisseur minimum
    ##  M : epaisseur maximum
    ##  V : direction calligraphique
    def __init__(self, m, M, V):
        StrokeShader.__init__(self)
```

```
        self._m = m
        self._M = M
        self._V = V
        # On instancie le functor 0D de requete
        # de l'information de normale 2D :
        self._getNormal2D = Normal2DF0D()

    ## Methode shade()
    def shade(self, stroke):
        it = stroke.strokeVerticesBegin()
        # On itere sur les sommets
        while(it.isEnd() == 0):
            ## Recupere la normale a la courbe
            n2d = self._getNormal2D(it).normalize()
            ## Calcule de l'epaisseur
            t = self._m + fabs(n2d*self._V)*(self._M-self._m)
            ## Affecte la nouvelle epaisseur
            it.getObject().attribute().setThickness(t/2.0,t/2.0)
            it.increment()
```

7.2.3 Troisième module de style : lignes visibles intérieures

Ce troisième et dernier module de style s'appuie surtout sur l'opérateur de sélection pour définir des traits correspondant aux lignes de la scène qui sont visibles et qui n'appartiennent pas au contour extérieur (*cf.* figure 7.4). On utilise notamment un *shader* permettant d'affecter aux traits une épaisseur variant de manière non linéaire.

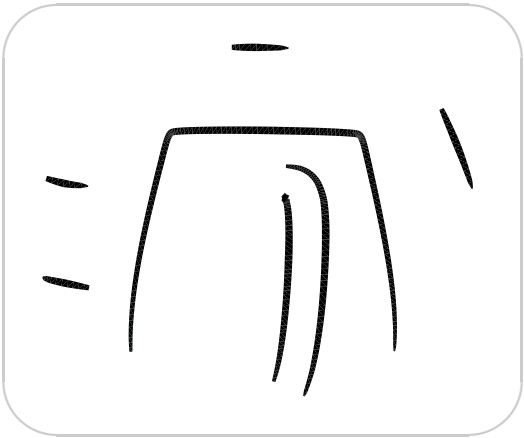


FIG. 7.4 – Troisième module de style : lignes visibles internes

Corps du module de style

```
## ////////////////////////////////////////////////////////
```

```
##                    Module de style
## ////////////////////////////////////////////////////////
## On instancie un predicat permettant de tester si un element 1D est
## visible et n'appartient pas au contour exterieur.
predicate1D = AndUP1D(QuantitativeInvisibilityUP1D(0),
                      NotUP1D(ExternalContourUP1D()))
## -- Selectionne les aretes qui sont visibles et non exterieures
Operators.select(predicate1D)
## -- Chaine a l'aide de l'iterateur de chainage standard.
## Par defaut le chainage reste dans la selection.
Operators.bidirectionalChain(ChainSilhouetteIterator(),
                             NotUP1D(predicate1D))
## -- Liste des shaders
shaders_list = [
        ConstantColorShader(0.2,0.2,0.2),## Affecte une couleur constante
        ## Affecte une epaisseur variant de maniere non lineaire
        pyNonLinearVaryingThicknessShader(2,     ## epaisseur minimum
                                          10,    ## epaisseur maximum
                                          0.33), ## exposant
                ]
## -- Cree les traits
Operators.create(TrueUP1D(),shaders_list)
```

Règles utilisées

Ici encore, seul un *shader* a été programmé (en plus du module de style) par l'utilisateur.

Shaders Ce *shader* affecte une épaisseur variant de manière non linéaire le long du trait.

```
## Affecte une epaisseur qui augmente puis diminue de
## maniere non lineaire.
## -- Derive de StrokeShader
class pyNonLinearVaryingThicknessShader(StrokeShader):
    ## Constructeur
    ##  m : epaisseur minimum
    ##  M : epaisseur maximum
    ##  e : exposant de non linearite
    def __init__(self, m, M, e):
        StrokeShader.__init__(self)
        self._m = m
        self._M = M  # M est atteint au milieu du trait
        self._e = e
    ## Methode shade()
    def shade(self, stroke):
        n = stroke.strokeVerticesSize()
        i = 0
        it = stroke.strokeVerticesBegin()
        while it.isEnd() == 0:
            att = it.getObject().attribute()
            if(i < float(n)/2.0):
                c = float(i)/float(n)
            else:
                c = float(n-i)/float(n)
            c = pow(float(c),self._e)*pow(2.0,self._e)
```

```
t = (1.0 - c)*self._M + c * self._m
att.setThickness(t/2.0, t/2.0)
i = i+1
it.increment()
```

7.2.4 Application à d'autres modèles

Nous avons beaucoup insisté sur l'aspect générique que se devait d'avoir une feuille de style pour pouvoir être réutilisable. En regardant attentivement les modules de style et les règles définies précédemment, on s'aperçoit que les instructions ne s'appuient à aucun moment sur des éléments spécifiques à la scène particulière (la théière et la tasse) utilisée pour modéliser le style. Il est donc possible d'appliquer ce style à n'importe quelle scène 3D, comme on peut le voir sur la figure 7.5.

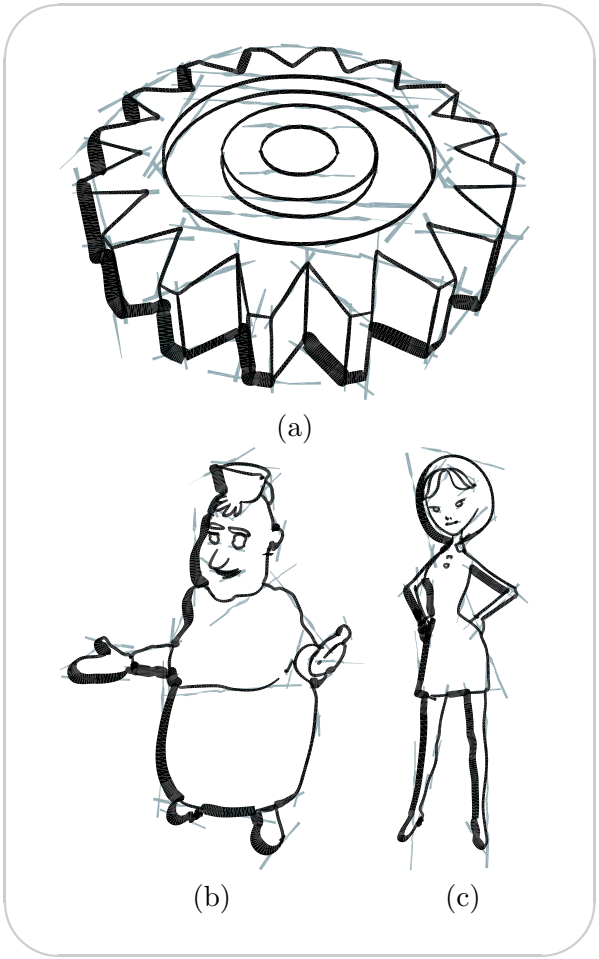


FIG. 7.5 – Généricité de la feuille de style

On applique la même feuille de style à plusieurs modèles différents.

7.3 Images de styles

Nous montrons à présent les résultats sous la forme d'images, ces dernières ayant été générées par notre système. Ces images illustrent la variété de styles accessibles par notre approche, la précision qui est offerte dans le contrôle de ces styles et la pertinence du postulat selon lequel un style peut être formulée de manière générique sous la forme de relations entre attributs de style et informations.

De par sa flexibilité, notre système permet la réalisation d'effets visuels inédits en NPR ; *toutefois, il est important de remarquer que notre principale contribution n'est pas dans ces effets, qui ne sont montrés qu'à titre d'illustration, mais bien dans l'approche unifiée et flexible pour le rendu de dessins au trait stylisés.*

Le temps de calcul nécessaire à un rendu stylisé avec notre système varie entre quelques secondes et quelques minutes pour un modèle d'environ 50K polygones. L'étape la plus coûteuse est le calcul du graphe de vue, en particulier en raison du calcul de l'information de visibilité qui se fait de manière exhaustive et par lancer de rayons. L'utilisation de l'information de densité peut également accroître le coût de l'étape de construction des traits de manière significative (pour atteindre un temps de calcul de l'ordre de la minute), ceci en raison du délai induit par la lecture dans le *framebuffer*. Par ailleurs, les faibles performances du langage Python à l'interprétation contribuent à éloigner notre rendu du temps réel. Toutefois, ces temps de calcul sont rarement prohibitifs et permettent, la plupart du temps, une exploration confortable lors de la modélisation d'un style. En outre, ils pourraient être fortement réduits si un effort était fourni dans ce sens. En effet, premièrement aucune optimisation d'envergure n'est à ce jour incluse dans le système, ensuite les prochaines cartes graphiques promettent des performances similaires pour l'écriture et la lecture dans le *framebuffer* et la vitesse d'exécution de Python augmente au fur et à mesure que des nouvelles versions sont distribuées.

Nous pensons finalement que le nombre de lignes de code nécessaires à la formulation d'un style donné, ou le temps requis pour qu'un utilisateur implémente un style constituent des mesures plus pertinentes de l'efficacité de notre système. Le lecteur trouvera ces chiffres dans le chapitre 8, en particulier sur la figure 8.1.

Avant de présenter les images de style les unes après les autres en les accompagnant des commentaires explicatifs nécessaires, nous proposons de considérer ces résultats sous une autre perspective, celle des fonctionnalités de notre approche.

7.3.1 Organisation par fonctionnalités

Dans cette section, nous proposons une liste des fonctionnalités principales et originales de notre système et, pour chacune d'elles, nous référençons les images qui en permettent l'illustration.

Segmentation en calques avancée Le système de calques s'appuyant sur l'opérateur de sélection, et cet opérateur étant programmable, les possibilités de regroupement des lignes de la scène en calques est infinie. Cette fonctionnalité est illustrée par les figures 7.7, 7.9, 7.10 et 7.11.

Contrôle de la toplogie des traits Notre système est le premier à offrir à l'utilisateur un contrôle sur la topologie des traits construits. Ce contrôle est en outre particulièrement flexible dans notre cas, dans la mesure où deux types d'opérateurs différents (chaînage et découpage) lui sont dédiés et que ces opérateurs sont programmables. Le contrôle sur

la topologie des traits est illustré par les figures 7.10 (chaînage au travers des petites occultations), 7.11 (extrémités des traits aux points de haute courbure), 7.12 (construction de traits passant plusieurs fois par les mêmes endroits) et 7.13 (création de traits se recouvrant).

Dépendance attributs/informations Tous les styles que nous montrons ici expriment à un niveau plus ou moins élevé des dépendances entre les attributs de style et les informations. Par exemple, les règles de chaînage, dans leur ensemble, s'appuient au minimum sur les informations de nature des lignes et d'identité des objets, pour chaîner ensemble des arêtes de même nature en suivant la topologie de l'objet. D'autres dépendances sont plus particulièrement visibles pour les illustrations 7.6 (information de couleur du matériau), 7.7 (information de densité), 7.9 (informations de nature des lignes et d'occultation), 7.10 (informations de densité et d'occultation) et 7.11 (information de courbure 2D).

Déformation géométrique des traits Une autre fonctionnalité difficilement accessible aux systèmes non programmables, est la possibilité de déformer la géométrie des traits. Outre les mécanismes standard de bruitage de la géométrie, qui permettent d'obtenir un aspect plus humain, c'est aussi une gamme de déformations plus radicales qui est ainsi offerte à l'utilisateur, comme l'illustrent les figures 7.7, 7.8, 7.11, 7.12 et 7.13.

Simplification par usage de la densité Finalement, l'intégration de formes avancées de l'information de densité, couplée à la présence d'opérateurs programmables de tri et de sélection, permet de spécifier des stratégies de simplification élaborées. Outre les exemples du chapitre 6, cette fonctionnalité est illustrée par les figures 7.7 (concentration des traits dans les zones déjà denses) et 7.10 (élagage uniforme d'une structure régulière).

7.3.2 Organisation par styles

L'organisation que nous adoptons maintenant pour présenter ces images s'appuie sur le type d'illustration auquel l'image appartient. Ainsi, les intitulés des sections ressemblent-ils à "dessin animé" ou "illustration technique". Cette catégorisation ne s'appuie sur rien de formel et ne sert qu'à structurer le document pour en améliorer la lisibilité.

Style "dessin animé"

Nous avions mentionné au chapitre 3 qu'il était très courant dans les dessins animés de s'appuyer sur la couleur du matériau pour choisir la couleur des traits (*cf.* figure 3.3 page 58). Ce type de dépendance attribut/information rentre tout à fait dans le cadre de notre approche programmable. Nous avons donc écrit un *shader MaterialColorShader* qui affecte à un trait une couleur qui est une variation de celle de son matériau. Pour ce faire, nous travaillons dans l'espace LUV des couleurs [WS82b] et translatons le vecteur couleur du matériau dans la direction L, dans le sens négatif si la luminance est supérieure à un seuil spécifié par l'utilisateur, de manière à obtenir une couleur plus sombre, et dans le sens positif, dans le cas inverse, pour que la couleur obtenue soit cette fois plus claire. La figure 7.6 illustre ce *shader*. En outre, sur ces illustrations, la géométrie des traits a été lissée et leur épaisseur varie afin que l'aspect obtenu soit plus fluide.

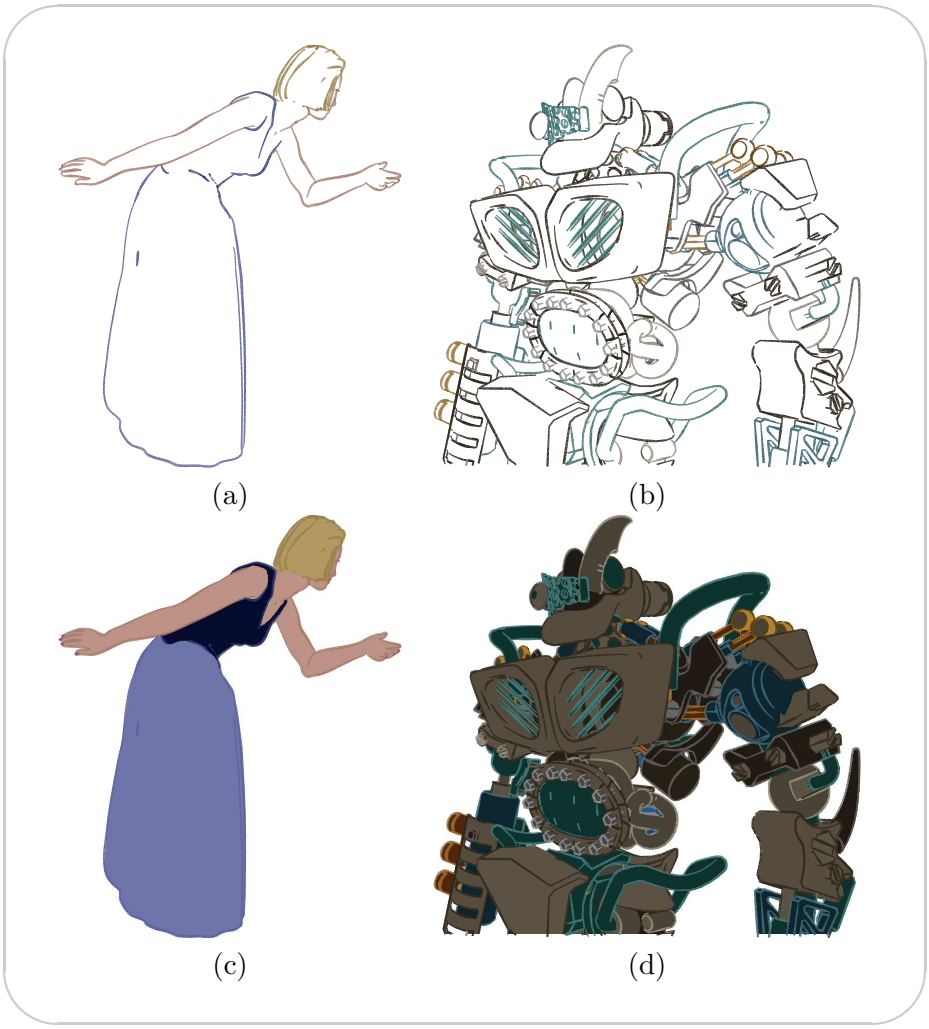


FIG. 7.6 – Style dessin animé

Sur ces illustrations, la couleur des traits a été déterminée automatiquement dans un shader *à partir de la couleur du matériau correspondant afin d'imiter le style des films d'animation traditionnels. La couleur d'un trait est obtenue en translatant, dans l'espace* LUV*, la couleur du matériau selon L dans un sens ou dans l'autre, selon que le matériau est clair on foncé. (a) et (b) : Les traits seuls. (c) et (d) : les traits superposés à un rendu uniforme du modèle 3D.*

Style "Renaissance"

La figure 7.7 montre un exemple complexe construit à partir de huit modules de style et appliqué à un modèle de statue de la vierge. Cet exemple illustre notamment l'utilisation du système des modules de style pour construire des styles riches, ainsi que l'exploitation de l'information de densité pour mettre en valeur des sous parties du modèle. Trois des modules construisent un patron du dessin (*cf.* figure 7.7 (b)) : les deux premiers tracent des "boites englobantes" à partir des traits, donnant une approximation très grossière de la forme du modèle, et le troisième dessine des traits de crayon temporaires, imitant un style esquissé en utilisant par exemple des *shaders* de bruit et de lissage. Les modules suivants marquent

le début de ce qui est supposé être le dessin final en dessinant les traits les plus longs dans un ton plus clair (*cf.* figure 7.7 (c)) affirmant ainsi les grandes courbes du modèle. Les deux modules de style restant ajoutent les traits de détail (*cf.* figure 7.7 (c)) : les petits traits se trouvant dans des régions de haute densité sont sélectionnés de manière à accentuer le tracé des zones complexes telles que les mains ou les visages. Le résultat de la combinaison de tous ces modules rappelle les dessins inachevés de la période de la Renaissance. L'omission contrôlée de lignes joue notamment un rôle important dans l'emphase des régions d'intérêt pour lesquelles plus de détails ont été gardés.

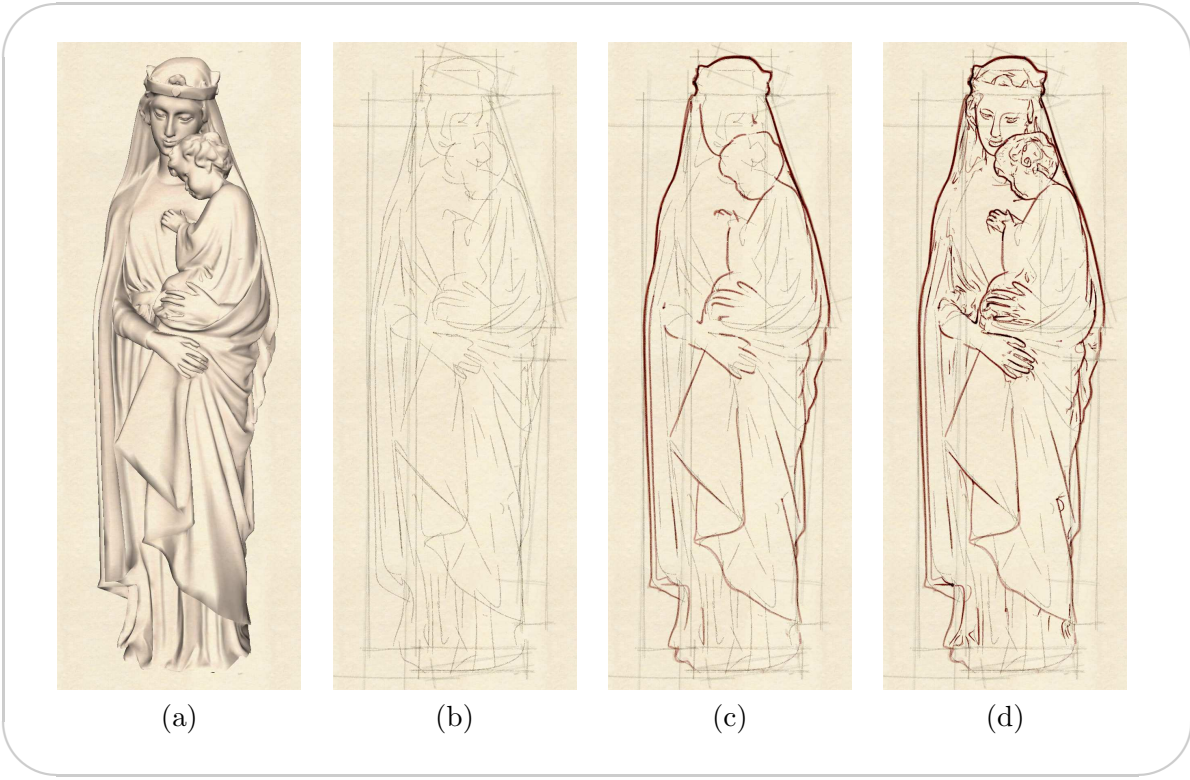


FIG. 7.7 – Rendu de la vierge dans un style "Renaissance"

Un style complexe composé de plusieurs modules de style. (a) : le modèle 3D de la statue de la vierge auquel le style est appliqué. Les modules de style ont été séparés en trois groupes. Les différentes illustrations montrent l'ajout progressif de ces modules de style. (b) : le premier groupe trace un patron pour le dessin. (c) : Le deuxième affirme la forme définitive en traçant les lignes les plus longues. (d) : Le troisième groupe ajoute des détails dans les zones de forte densité.

L'illustration de la figure 7.8 utilise le même type d'attributs que le style de la figure 7.7. En revanche, la feuille de style est ici beaucoup plus simple : un module de style trace les lignes cachées par une seule surface sous forme de traits de construction (leur géométrie est étirée) afin de conférer au résultat final un aspect "technique", tandis qu'un autre dessine les lignes visibles avec des attributs visuels évoquant l'utilisation d'une plume et d'encre. L'image ainsi obtenue rappelle les illustrations techniques de la même période de la "Renaissance".

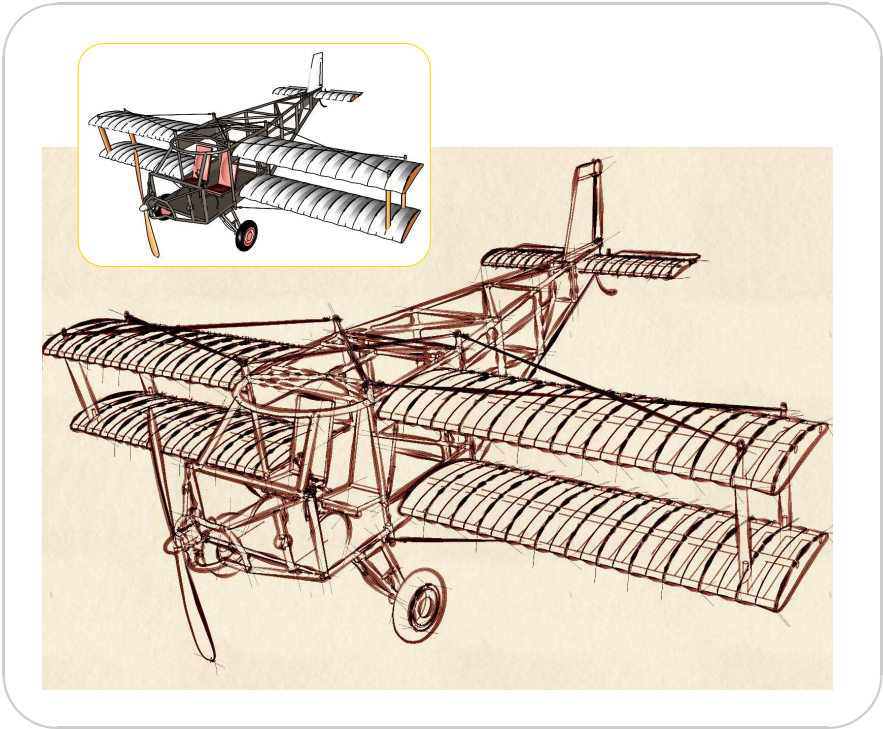


FIG. 7.8 – Style illustration technique de la “Renaissance”

Grâce à un tracé ténu des lignes cachées, déformées pour simuler des traits de construction, et à un choix judicieux des attributs bas niveau de style, cette illustration évoque les dessins techniques d'époque.

Style illustration technique

L'illustration technique est l'une des applications de prédilection de notre approche. En effet, l'une des conséquences de l'avènement de la CAO est que les industriels disposent des modèles 3D des objets qu'ils conçoivent et manufacturent. Dès lors, il semble intéressant d'utiliser ces modèles pour générer les divers types d'illustrations requis (*e.g.* manuels utilisateur, illustrations techniques). L'emploi d'un système tel que le nôtre dans ce contexte permettrait, d'une part, une automatisation plus importante de ces tâches et, d'autre part, un accès plus aisé aux visualisations complexes utiles en illustration technique.

La figure 7.9 illustre, par exemple, ce type de contraintes sur un modèle de locomotive d'époque pour lequel on veut visualiser précisément un sous-ensemble d'éléments, tout en le gardant dans le contexte de l'assemblage complet. C'est en particulier l'information des objets occultants qui est exploitée dans cette feuille de style. Les modules de style utilisés peuvent être réunis en trois groupes : le premier trace les lignes n'appartenant pas au sous-ensemble à mettre en valeur. On choisit donc un style imprécis et discret pour tracer ces traits. Le deuxième trace les lignes de silhouette des objets importants : les segments de ces traits, qui sont cachés par d'autres objets que ceux appartenant au sous-ensemble d'intérêt, sont tracés en pointillé, tandis que les segments visibles sont tracés en trait plein. Le troisième groupe applique un traitement similaire aux lignes d'arêtes vives et les trace d'une couleur différente de celle employée pour les silhouettes, de manière à respecter les conventions du

dessin technique. Le style résultant permet de visualiser clairement le sous-ensemble des objets intéressants, tout en les conservant dans le contexte de l'assemblage complet.

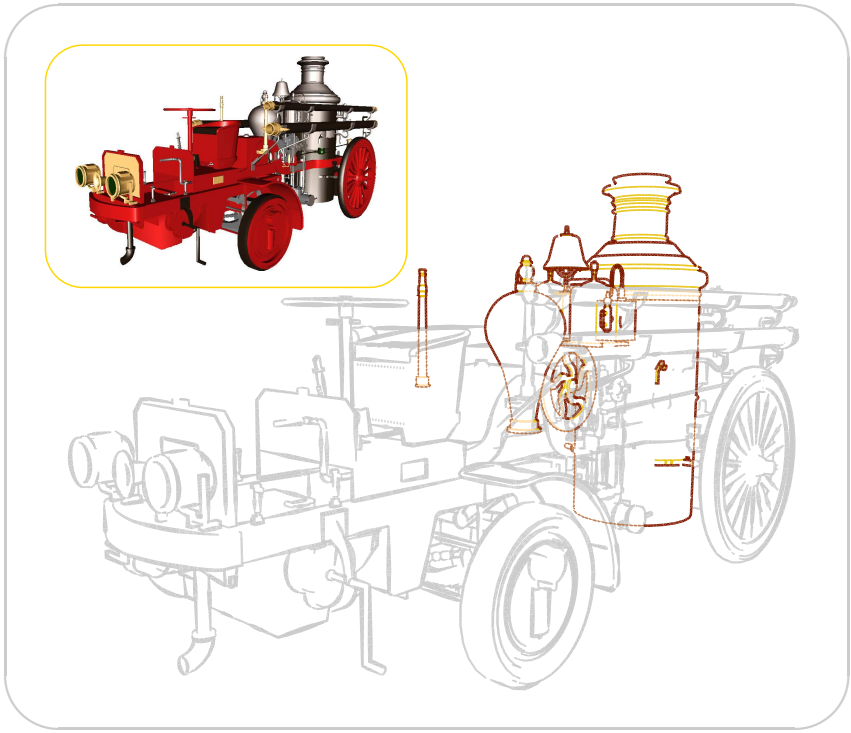


FIG. 7.9 – Style illustration technique

On utilise dans cette illustration l'information des objets occultants pour mettre en valeur un sous-ensemble d'objets d'une locomotive d'époque (le modèle 3D est montré en haut à gauche). Les lignes n'appartenant pas à ce sous-ensemble sont tracées en gris clair et de manière approximative afin de ne pas focaliser l'attention. Pour le sous-ensemble des objets importants, les lignes sont entièrement dessinées, y compris les parties cachées par des objets n'appartenant pas à ce sous-ensemble, qui sont tracées en pointillé.

Style incluant de la simplification

La simplification et l'abstraction restent, à ce jour, l'apanage de l'humain et contribuent à donner à une image l'apparence d'un dessin fait à la main.

Ainsi, dans la représentation d'un objet complexe, tel que celui montré sur la figure 7.10 (a), il est probable que beaucoup d'artistes omettent un certain nombre de lignes pour améliorer la lisibilité du résultat. De manière similaire à la figure 6.11 page 170, on utilise donc la densité causale pour simplifier la grille qui masque le compresseur : chaque barreau est tracé avec un seul trait, et la moitié des barreaux est uniformément supprimée. La figure 7.10 (b) montre le résultat d'une telle opération. On voit alors le problème entraîné par cette simplification : il y a une incohérence entre la visibilité induite par la grille initiale et celle que l'on devrait avoir avec la grille simplifiée. Pour corriger ce problème, on programme un itérateur de chaînage avancé qui permet de boucher les petites occultations : lorsqu'une arête cachée est rencontrée au cours du chaînage, un parcours prédictif permet de connaître la taille relative

de la région cachée, entamée par cette arête, par rapport à la taille de la chaîne complète lorsque les occultations ne sont pas prises en compte. Si cette taille est suffisamment petite, l'arête est sélectionnée. Un tel chaînage permet de construire des traits continus au travers des petites occultations, comme on peut le voir sur la figure 7.10 (c). Ce type de combinaison simplification/chaînage est parfaitement adaptée à une mise en style de type "esquisse" (*cf.* figure 7.10 (d)) qui tolère les quelques imprécisions résultant d'un tel processus.

Style "esquissé"

De nombreux styles sont regroupés sous cette appellation. Tous les styles donnant la claire impression visuelle de dessins faits à la main tombent dans cette catégorie. On distingue différentes techniques qui participent à donner cette impression visuelle.

Lignes de construction L'un des effets qui permettent de donner un aspect "esquisse" à un dessin est l'ajout de lignes de construction. En associant un module de style produisant de tels traits à un module créant des lignes plus définitives (au moins pour une partie de la scène), on donne au dessin une apparence inachevée. Le module de style chargé des lignes de construction consiste principalement en un découpage récursif aux points de courbure importante suivi d'une déformation géométrique permettant d'approximer chaque trait par un segment rectiligne. Sur la figure 7.11, nous avons combiné ce module de style avec un module de style produisant des traits définitifs pour les lignes les plus proches du point de vue. Le code correspondant au module de style chargé des traits de construction se trouve section 7.2.1.

"Gribouillage" Le terme "gribouillage" désigne ici l'impression causée par le fait de dessiner une ligne (de la vue) à l'aide de plusieurs traits se recouvrant au moins partiellement. Cette technique est très fréquemment utilisée durant les phases de recherche de la réalisation d'une œuvre picturale. Il existe en particulier deux moyens de créer de tels traits dans notre système, le chaînage multiple et le découpage séquentiel.

Chaînage multiple Le principe du chaînage multiple est de permettre le chaînage d'une même arête plusieurs fois (*cf.* section 5.4.2). Les traits construits de cette manière peuvent ensuite être spatialement bruités de manière à distinguer les différents passages de la chaîne par une même arête. Il résulte de cette construction une impression de "gribouillage" particulièrement naturelle, comme on peut le voir sur la figure 7.12.

Découpage séquentiel Comme nous l'avons expliqué à la section 5.2.4, il est possible d'avoir des recouvrements entre les traits produits par un découpage séquentiel. En choisissant les prédicats de coupure pour les points de début et de fin de chaîne de manière à ce que l'on ait beaucoup plus de points de début que de fin (ou inversement), on obtient des traits tels que ceux que pourrait tracer un artiste s'y prenant à plusieurs reprises pour tracer le même trait, mais en le débutant à un endroit différent à chaque fois. L'illustration de la figure 7.13 a été créée en utilisant cette technique.

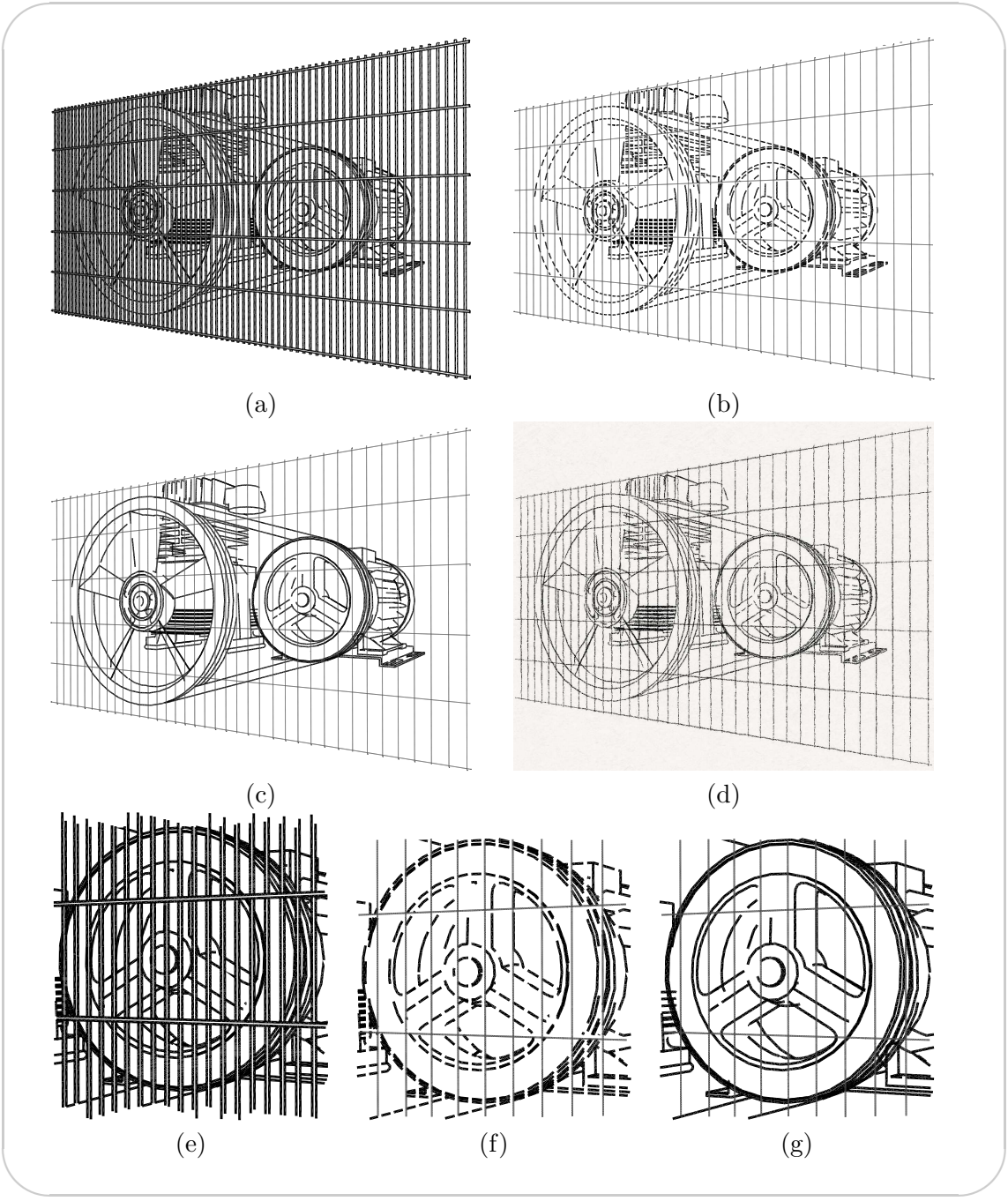
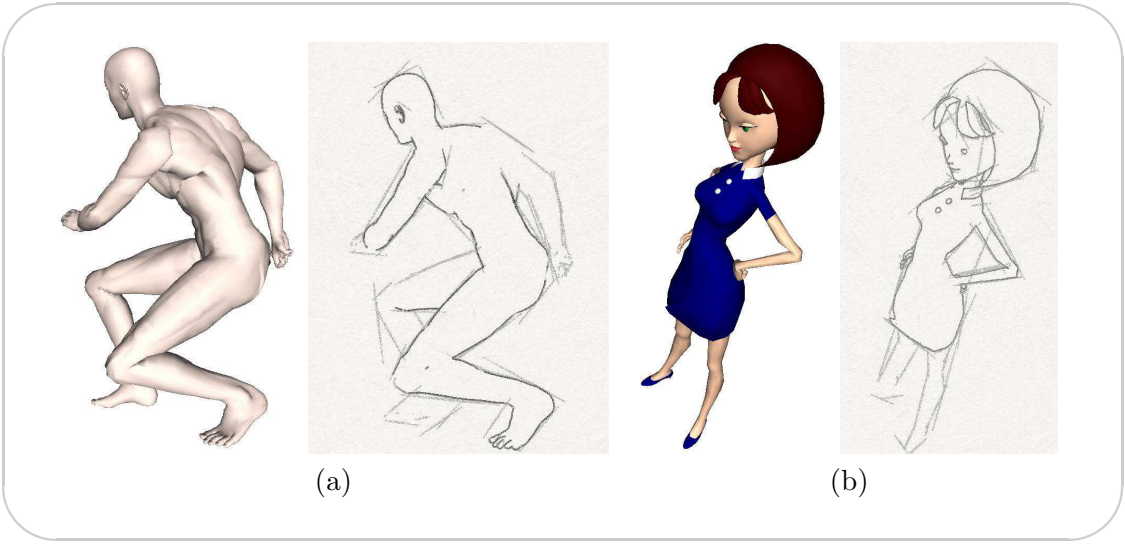


FIG. 7.10 – Simplication basée sur la densité et chaînage au travers de petites occultations

(a) : l'ensemble initial des lignes d'un objet complexe. (b) : La grille a été simplifiée de manière uniforme (cf. figure 6.11) afin d'améliorer la lisibilité du dessin. Si aucun traitement n'est fait, ceci résulte en un effet pointillé pour les lignes du compresseur dont l'information de visibilité n'est plus à jour. (c) : On utilise un chaînage avancé qui permet de chaîner au travers des occultations suffisamment petites pour boucher les trous dus à l'omission de lignes de la grille. (d) : Le résultat final ; une telle simplification est bien adaptée à l'affectation d'attributs de type "esquisse". (e) : Un gros plan sur la petite roue du compresseur, tracée avec l'ensemble des lignes initiales, (f) : après simplification de la grille (on voit clairement l'effet pointillé sur les traits du compresseur), (g) : après utilisation du chaînage au travers des petites occultations (les traits ont été construits au travers de ces petites occultations).



(a) (b)

FIG. 7.11 – Style sketchy : lignes de construction

La combinaison d'un module de style pour les lignes de construction avec un module de style dessinant des traits définitifs pour les lignes suffisamment proches permet d'obtenir un style d'esquisse inachevée. En haut, les modèles 3D, en bas, les dessins produits par notre système.

Style oriental

Nous avons ici tenté d'imiter un style oriental à l'aide de deux modules de style simulant deux tailles de pinceaux différentes. Pour ces deux modules, le travail concerne principalement les attributs bas niveau de style : les traits sont raccourcis aux extrémités et d'importantes variations d'épaisseur leur sont affectées. Le premier module simule ainsi un pinceau épais pour tracer les lignes les plus longues. Il utilise notamment l'information de densité pour éviter l'encombrement de gros traits et garder un aspect épuré. Le deuxième simule un pinceau plus fin, et est utilisé pour tracer les autres lignes, plus petites. Les illustrations de la figure 7.14 ont été réalisées avec ces modules de style.

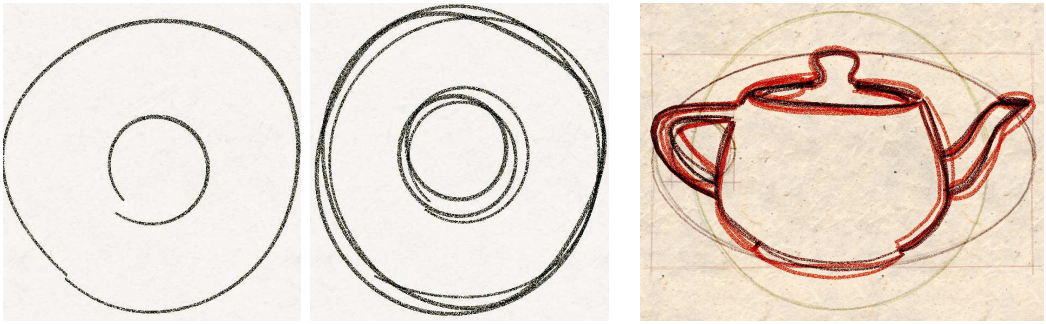


FIG. 7.12 – Style sketchy : chaînage multiple

(a) : Utilisation d'un shader *de bruitage spatial sur des traits construits par chaînage simple (à gauche) et par chaînage multiple (à droite). (b) : Combinaison d'un module de style utilisant un chaînage multiple avec des modules de style créant le patron du dessin pour renforcer l'effet "esquisse".*

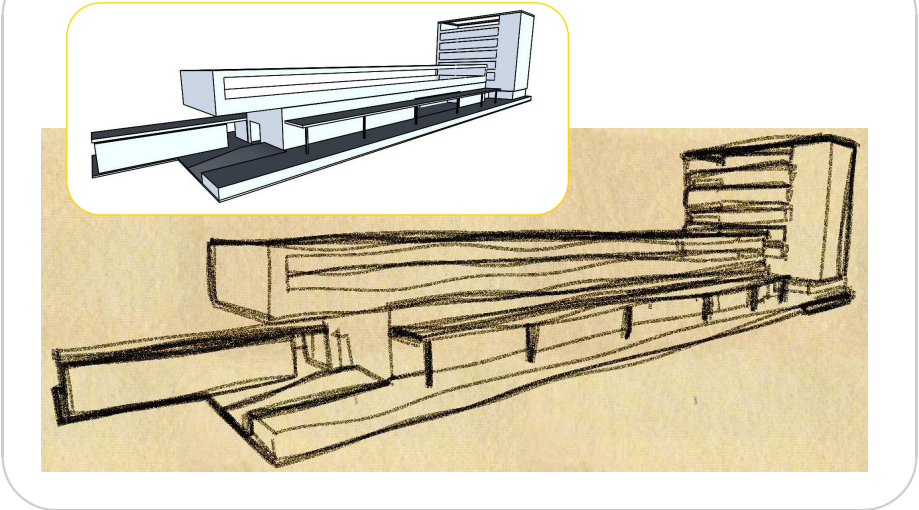


FIG. 7.13 – Style sketchy : découpage séquentiel

L'effet "gribouillage" est ici produit en utilisant un découpage séquentiel pour lequel les prédicats de coupure de début et de fin de chaînes sont découplés et en appliquant aux traits ainsi construits un shader *de bruitage spatial.*

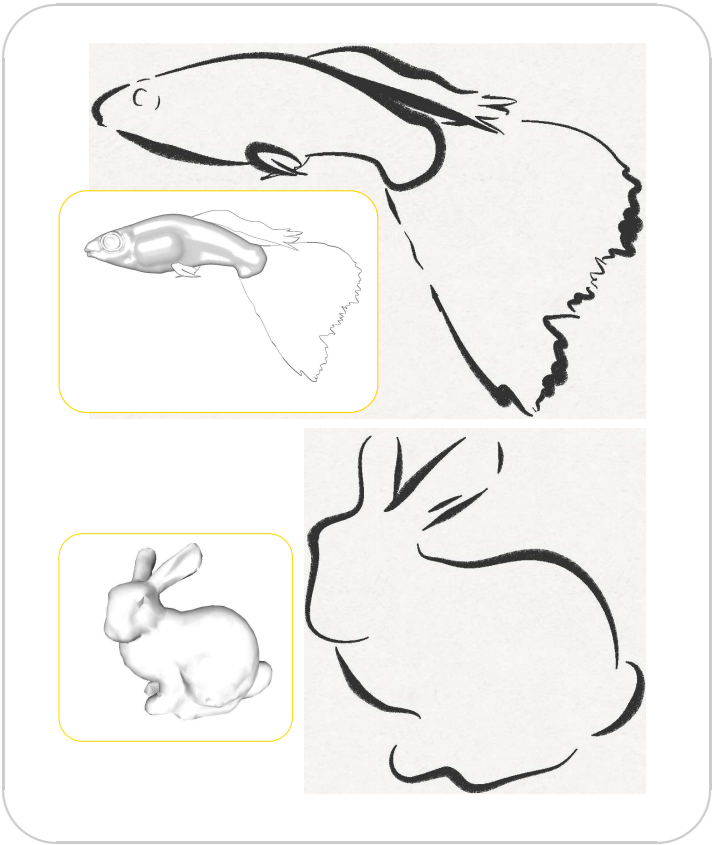


FIG. 7.14 – Style oriental

Cette imitation de style oriental a été réalisée à l'aide de deux modules de style simulant respectivement un gros pinceau et un pinceau de taille moyenne. Ces deux modules de style utilisent notamment des shaders *de raccourcissement des traits et de réduction progressive de l'épaisseur aux extrémités.*

chapitre 8

Conclusion

Ce mémoire présente une nouvelle formulation du processus de création d'image pour la génération de dessins au trait à partir de modèles 3D. Dans ce chapitre, nous commençons par faire un résumé des contributions et ensuite nous citons quelques pistes qui nous semblent pertinentes pour des travaux futurs.

8.1 Résumé des contributions

Cette thèse s'appuie sur le postulat que le style dans le dessin de lignes peut être exprimé comme la mise en relation d'attributs de style avec des informations génériques extraites de la scène représentée. L'une de nos contributions est d'avoir validé cette supposition au travers de nos expérimentations.

Toutefois, notre principale contribution est l'approche unifiée et flexible pour le rendu stylisé de dessins au trait qui découle de la supposition mentionnée précédemment et qui permet d'exprimer un style de façon générique (*i.e.* indépendante d'une scène donnée). En particulier, à un niveau de détails plus fin, nous avons ainsi contribué à :

- L'identification d'un ensemble d'attributs de style plus complet que ceux qui avaient pu être proposés avant, incluant notamment des attributs de haut niveau tels que la sélection ou la topologie des traits.
- L'identification d'un ensemble d'informations pertinentes dans le cadre de la modélisation de style.
- La spécification d'un ensemble de structures utiles à la modélisation de style à partir d'une scène 3D et en particulier du *graphe de vue* qui est adapté à une mise en style basée sur l'information.
- Une décomposition du processus de dessin sous forme d'un ensemble d'opérateurs programmables organisés en pipeline et l'identification des éléments névralgiques de ce pipeline dont le contrôle doit nécessairement être laissé à l'utilisateur.
- Une exploitation de l'information de densité comme contrôleur de stratégies de simplification de dessins au trait.

L'introduction de *shaders* programmables pour le rendu non-photoréaliste ouvre de nombreuses voies intéressantes pour le design graphique et stylisé, d'autant plus que, par définition, les styles non-photoréalistes autorisent la plus grande liberté quant aux modifications visuelles et géométriques du modèle sous-jacent.

Dans notre expérience, le développement d'une feuille de style est comparable à celui d'un *shader* procédural en rendu traditionnel. L'étape la plus laborieuse est souvent la formulation du but en des termes qui puissent être traduits sous forme de programmes. Cette étape passée, l'expérimentation et le raffinement se déroulent de façon naturelle et aisée.

Il est finalement important de noter que notre principale contribution n'est pas dans les nouveaux effets visuels que nous avons montrés, mais bien dans l'approche unifiée et flexible pour le rendu stylisé de dessins au trait que nous venons de détailler.

8.2 Discussion

Comme pour beaucoup de travaux de recherche touchant au NPR, il est difficile de valider notre approche.

Dans cette section, nous tâchons, tout au moins, de poser les questions relatives à une évaluation de la thèse présentée ici. On commence par discuter les choix faits à un bas niveau avant de s'interroger sur les choix plus généraux relatifs à l'approche elle-même.

8.2.1 Évaluation des décisions de bas niveau

Si l'on admet la pertinence d'une approche dans laquelle le style est décrit comme une procédure mettant en relation des attributs et des informations, alors les décisions de bas niveau portent sur le choix des attributs de style, des informations utiles, des opérateurs et de leur organisation en pipeline ainsi que sur l'identification des éléments dont le contrôle doit être fourni à l'utilisateur.

Choix des attributs

La question est d'une part de savoir si les attributs considérés sont nécessaires et suffisants et d'autre part si l'approche permet à l'utilisateur d'étendre facilement les attributs de base.

Il est important de noter que les deux groupes d'attributs de style que nous avons identifiés, respectivement de bas et de haut niveau, nous ont permis de réaliser tous les objectifs de style que nous nous étions fixés. On ne peut pour autant pas affirmer que ces attributs sont absolument suffisants pour représenter tous les effets visuels possibles. Les conséquences du manque d'un attribut ne sont pas les mêmes selon que celui-ci fasse défaut aux attributs de bas ou de haut niveau. Le premier cas est moins probable dans la mesure où, d'une part, les attributs de bas niveau sont par nature facilement observables et, d'autre part, ces attributs sont utilisés depuis longtemps en NPR. En outre, l'ajout d'attributs de bas niveau est relativement facile à intégrer en autorisant l'ajout d'attributs de type réel ou vecteur de réels de dimension deux ou trois (comme c'est le cas dans notre système). Il faut cependant, bien évidemment, disposer du moteur de rendu adéquat, prenant les attributs définis par l'utilisateur en compte. On pourrait par exemple intégrer un attribut "orientation de la plume" qui permettrait au moteur de rendu d'automatiquement faire varier l'épaisseur du trait en conséquence, de manière à imiter un trait calligraphique. Le cas du manque d'attributs de haut niveau est plus aisément envisageable et a des répercussions plus profondes sur l'approche. Plus aisément envisageable parce que la perception des attributs de haut niveau requiert un œil exercé et qu'il serait prétentieux d'affirmer que le notre l'est absolument. Les répercussions d'un tel manque pourraient être plus profondes dans la mesure où elles impliqueraient potentiellement un manque dans la base d'opérateurs elle-même.

Nos expériences et observations n'ont en tout cas jamais remis en cause cette liste d'attributs.

Choix des lignes caractéristiques

Comme nous l'avions mentionné dans le chapitre 2, la définition géométrique des lignes caractéristiques reste à ce jour un sujet de recherche très actif. Pour cette raison, les familles de lignes caractéristiques évoluent rapidement. Notre approche ne fait aucune supposition sur la nature des familles de lignes caractéristiques détectées en entrée ni sur leur nombre. Elle est donc parfaitement extensible à de nouvelles familles de lignes.

En terme d'implémentation, il serait bien sûr nécessaire de modifier la base de code du système, mais l'architecture de ce dernier a été pensée de manière à permettre facilement une telle intégration. En particulier, le code de construction du graphe de vue, qui est le plus sensible à l'ajout d'une famille de lignes caractéristiques, est prévu pour gérer automatiquement toute ligne caractéristique passant par les arêtes du maillage ou en traversant les faces.

> **Remarque :** L'ajout d'une nouvelle famille de lignes caractéristiques ne constitue pas un ajout d'information mais un ajout d'une valeur à une information discrète existante (l'information de nature des lignes). Comme nous allons le voir dans la prochaine section, l'ajout d'une information, peut en revanche remettre en question des éléments de l'approche elle-même.

Choix des informations

De même que pour les attributs de style, on peut se poser la question de savoir si, d'une part, l'ensemble des informations considéré est nécessaire et suffisant et, d'autre part, s'il est aisément extensible par l'utilisateur.

Nous avons prouvé tout au long de cette thèse que ces informations étaient nécessaires. Sont-elles suffisantes ? La collecte de ces informations s'est faite au fur et à mesure de nos expérimentations jusqu'à ce que l'ensemble formé se stabilise. Il est toutefois impossible de dire qu'il n'existe pas de style nécessitant une information supplémentaire. Les informations de base communément prises en compte par les modeleurs 3D, telles que la géométrie ou les matériaux, sont en tout cas toutes disponibles dans notre système. Il existe au moins un type d'information qui n'est pas intégré à notre liste, il s'agit des information sémantiques. Ces dernières jouent souvent un rôle dans la stylisation d'un dessin et ne sont pas toujours exprimables commme une combinaison des informations classiques. Notre méthode échouera à modéliser un tel style.

Concernant l'extensibilité du système d'informations par l'utilisateur, elle est à ce jour impossible. Si ce dernier peut définir ses propres fonctions de requêtes d'information à partir des informations disponibles, l'accès au modèle 3D et donc l'ajout d'informations ne sont pas prévus. L'ajout d'informations d'ordre sémantique est par conséquent impossible. Conceptuellement, et dans le cas général, la difficulté principale liée à cette extensibilité est que tout ajout d'information implique potentiellement la redéfinition du graphe de vue. Cette dernière opération est compliquée, d'une part, et se situe largement en amont de la zone de contrôle offerte à l'utilisateur (constituée par le module de style). Seul l'ajout d'une information continue en 2D n'entraînerait pas de modifications du graphe de vue. On pourrait donc envisager la possibilité de permettre à l'utilisateur de définir des cartes 2D d'informations (définies dans

les principaux espaces de base, $\mathbb{R}$, $\mathbb{R}^2$, ...) sous la forme d'images correspondant pixel à pixel au au dessin. La mise en œuvre d'un tel mécanisme fait partie des travaux futurs.

Structuration en graphe de vue

La structuration des lignes caractéristiques sous la forme d'un graphe de vue comporte deux aspects : le découpage des lignes en arêtes en suivant la continuité par morceaux de la fonction d'information d'un côté, et la structuration de ces arêtes en graphe planaire de l'autre côté.

Le choix du découpage des lignes s'appuyant sur la fonction d'informations découle naturellement de notre approche qui place les informations au cœur de la plupart des décisions. En effet, dans ce contexte, il semble raisonnable de souhaiter manipuler des lignes sur lesquelles on n'observe pas de changement radical de l'information (dans le cas contraire, une ligne contenant à la fois une partie cachée et une partie visible, devrait-elle être considérée visible ou invisible ?).

La structuration de ces lignes en graphe découle naturellement de la nature 2D d'un dessin qui offre à l'artiste la possibilité de s'affranchir de la topologie 3D de la scène en exploitant les relations entre lignes voisines. Il est donc essentiel de disposer d'un tel arrangement pour offrir à l'utilisateur un contrôle flexible sur la topologie des traits.

> **Remarque :** Au-delà de la question structurelle, le choix de travailler avec le graphe de vue signifie également regarder le processus de dessin comme une projection de la 3D vers la 2D. Est-ce une limitation et existe-t-il une alternative ? Les conséquences directes de ce point de vue sont que seules les lignes ayant été détectées en 3D pourront apparaître dans le dessin et, les traits ne pourront pas s'écarter de la topologie 2D de la scène. Cela peut constituer une limitation sévère, en particulier dans le cas de la simplification de dessins au trait : par exemple, un artiste dessinant de manière simplifiée une structure complexe telle qu'une grille ou un toit modifiera souvent l'échelle ou la position des motifs dessinés. Notre approche ne permet pas ce type de simplifications.

Choix et organisation des opérateurs

La décomposition du processus de dessin en opérateurs, telle que nous la proposons, découle à la fois des éléments manipulés (éléments du graphe de vue et traits) et des attributs de style.

Les éléments manipulés proviennent de la modélisation classique du processus du dessin comme un pipeline travaillant sur des arêtes caractéristiques et produisant des traits stylisés. Notre modélisation n'est pas différente si ce n'est que nous offrons plus de liberté dans les chemins suivis par les traits et qu'un premier chaînage a été effectué pour transformer les arêtes (du maillage) initialement détectées comme caractéristiques, en arêtes du graphe de vue (chaînage justifié précédemment).

La base d'opérateurs que nous avons choisie s'articule donc autour de la construction et de la mise en style de traits à partir des arêtes du graphe de vue et a pour objectif de permettre la modélisation de tous les attributs de style.

Cet ensemble d'opérateurs est-il nécessaire ? Nous avons démontré dans ce document l'intérêt de chaque opérateur, en particulier en montrant des styles irréalisables en l'absence de l'un d'eux ; ils sont donc nécessaires.

Sont-ils suffisants ? Démontrer cette propriété reviendrait à prouver que n'importe quel style peut être modéliser à l'aide de cette base, ce qui est impossible. Sans pouvoir le prouver davantage, nous espérons tout au moins que tout style s'appuyant sur l'ensemble des informations considéré peut être décrit avec ces opérateurs.

Finalement, une certitude est qu'il est possible de générer une grande variété de styles avec cette approche.

Concernant l'organisation en pipeline (et les modes de synchronisation), nous avons vu à la section 5.4.4 qu'une organisation plus flexible de la séquence d'opérateurs pouvait être imaginée.

Choix des éléments programmables

Enfin, la dernière décision de bas niveau concerne le choix des éléments du pipeline dont la programmation doit être confiée à l'utilisateur.

De ce choix découlent la flexibilité du système ainsi que l'apport qu'il représente dans le travail de modélisation de style. En effet, en n'exposant que peu d'éléments à la programmation, la charge de demandée à l'utilisateur est plus faible mais le système en devient également moins flexible. Par contraste, si trop d'éléments sont laissés à la charge de l'utilisateur, la facilité de développement de style s'en trouve réduite et l'intérêt du système en est amoindri.

À chaque niveau du processus de dessin nous avons donc tâché d'isoler les éléments pour lesquels seul l'utilisateur pouvait prendre une décision. En pratique, on trouve des éléments programmables dans la requête d'informations, et dans les opérateurs. Concernant la requête d'informations, le système fournit des fonctions d'accès pour l'ensemble des informations et adopte un comportement par défaut dans le cas de requêtes ambiguës (aux points de discontinuité de l'information). L'utilisateur a la possibilité de surcharger ce comportement en ces points de discontinuités, ce qui semble être la contribution à la fois nécessaire et minimale qui puisse lui être demandée.

Pour chaque opérateur, nous avons séparé la structure algorithmique commune à toute utilisation des parties variables qui en déterminent le comportement. Ainsi, le travail de l'utilisateur consiste uniquement, dans la majorité des cas, à définir des objets fonctions (*functors*) et des fonctions d'itérations, ce qui, ici encore, semble le minimum essentiel à exposer et représente également une façon intéressante de factoriser le travail de code fourni par l'utilisateur. Seul l'opérateur de *shading* ne présente pas de structure algorithmique standard et son implémentation est entièrement laissée à l'utilisateur.

8.2.2 Évaluation du succès de l'approche

En prenant un peu de recul par rapport au travail effectué, on peut maintenant se demander si notre approche atteint les objectifs fixés. Cette section est organisée sous la forme d'une suite de questions, accompagnées, le plus souvent possible, de réponses.

Quelle est la diversité des styles accessibles ?

Nous avons montré au travers des différents exemples illustrant la thèse une grande variété dans les styles générés. Sans pouvoir prétendre générer n'importe quel style, nous pouvons affirmer que de nombreux styles très différents peuvent être modélisés en suivant notre approche.

Par ailleurs, en nous inspirant de styles observés dans des illustrations réelles pour créer nos exemples, nous avons en quelque sorte soumis notre système au test de la reproduction de styles, et ceci toujours avec succès. Bien que ce processus soit hautement bruité, notamment parce que les démarches d'analyse et de compréhension du style réel sont dépendantes de l'utilisateur, il nous a conforté dans nos choix d'attributs, d'opérateurs et d'informations.

Ce succès est dû à deux facteurs principaux. Premièrement, notre approche est entièrement focalisée sur une flexibilité maximale dans le contrôle du style. Ainsi, le choix d'une approche programmable laisse à l'utilisateur une grande liberté d'expression dans ses descriptions. En particulier, des styles très complexes s'appuyant sur un grand nombre d'informations, et donc impossibles à décrire à l'aide d'une approche interactive, sont accessibles. Ensuite, nous avons pris soin de définir le style de la manière la plus fine possible, notamment en introduisant des attributs de style de haut niveau, tels que la topologie des traits, souvent négligés par les travaux précédents. C'est la combinaison de ces deux aspects qui offre une infinité de variations sur nos styles.

Est-il facile d'explorer des styles ?

Premièrement, l'exploration de nouveaux styles n'est possible que grâce à la grande flexibilité fournie par notre approche. Ensuite, bien que la programmation ne constitue peut-être pas la plus intuitive des techniques d'interaction, elle reste probablement la plus adaptée à l'exploration de styles. En effet, cette approche a l'avantage de naturellement séparer les différentes étapes du processus de dessin (contrôle de la topologie, affectation des attributs) et de privilégier une modélisation de style par niveaux de détails : on commencera par modéliser les grandes lignes du style afin de raffiner chaque composante à sa guise. Cette approche favorise également la réutilisation de styles existants pour l'exploration : on pourra par exemple éditer un style proche du style souhaité et le modifier par petites étapes jusqu'à atteindre l'objectif fixé. On notera qu'il est alors essentiel que la visualisation des modifications apportées à un style soit interactive, ce qui est aujourd'hui possible avec un langage de description de style interprété ou compilé à la volée. En outre, l'utilisateur peut instantanément vérifier le "passage à l'échelle" du style obtenu, en l'appliquant au rendu d'une variété de modèles 3D. Par manque de temps, nous n'avons malheureusement pas pu mener les études utilisateur nécessaires à confirmer ces impressions.

Quel apport représente ce système ?

Il n'est pas toujours évident dans le cas d'approches programmables, d'évaluer l'apport qu'elles représentent. En particulier, on peut se demander s'il est vraiment plus simple de programmer un style dans notre système que directement en C++, comme un moteur de rendu spécifique. Dans cette section, nous passons en revue les différents éléments d'un tel système qui sont indispensables et fournis, et qui représentent ainsi un gain de temps considérable dans la définition d'un style. (Nous ne citons pas ici les éléments basiques communs à la plupart des applications graphiques tels que la lecture de modèle 3D où la visualisation.)

La supposition principale de notre approche est qu'un style est défini en fonction d'informations génériques de la scène. La première économie considérable réalisée par l'utilisateur est de ne pas avoir à programmer le calcul, ni surtout l'accès, à ces informations.

L'ensemble des structures définies dans cette thèse, et en particulier le graphe de vue, représente une organisation essentielle des données. La construction de ces structures requiert un certain soin dont l'utilisateur n'a pas à se soucier.

Ensuite, bien qu'un travail de programmation lui soit demandé, l'utilisateur s'appuie sur une méthodologie lorsqu'il programme un module de style à l'aide de nos opérateurs. Disposer d'une telle méthodologie, si elle est bonne, représente également un gain de temps considérable. En outre, la segmentation des parties fixes et contrôlables des opérateurs via les règles de style évite à l'utilisateur un gros travail de code. Il est ici encore difficile d'évaluer la qualité de celle-ci. Elle réside en partie sur le choix des opérateurs et de leur granularité.

Au cours de nos expérimentations, nous avons identifié les règles les plus communément rencontrées dans la modélisation de style et les fournissons de manière standard dans le système. La programmation d'un style se restreindra ainsi souvent à une utilisation de combinaisons de composants prédéfinis.

Finalement, une mesure pertinente de la difficulté de produire un style avec notre approche peut être le nombre de lignes de code qu'il a été nécessaire d'écrire. La figure 8.1 donne une idée de la taille des modules de style écrits pour générer différents styles.

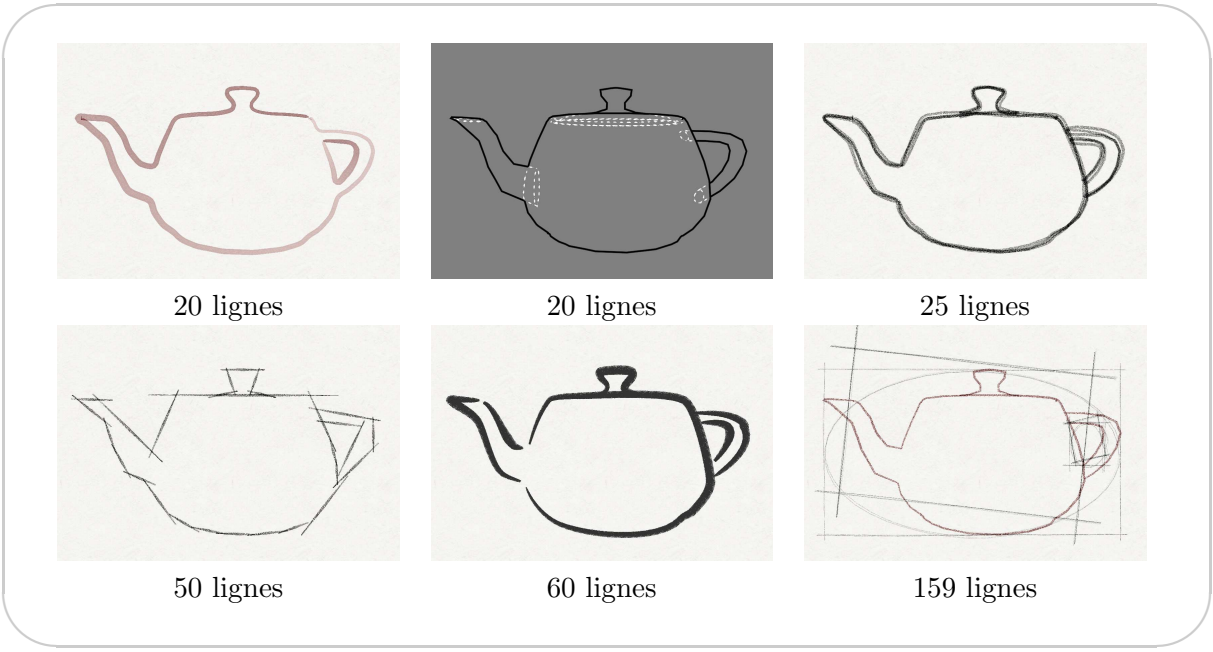


Fig. 8.1 – Nombre de lignes de code nécessaires à l'écriture de différents styles

Ces chiffres indiquent une relative facilité dans la programmation d'un style.

Ici encore, nous aurions souhaité mener une étude utilisateurs pour évaluer ce gain de temps.

Est-il facile de penser un style en terme de programmation ?

A priori, la structure algorithmique d'un programme ne semble pas forcément la plus adaptée à décrire un style visuel. Cette idée peut sembler d'autant plus étrange que les artistes ou les historiens de l'art n'étant généralement pas familiers des langages de programmation, elle a été rarement envisagée.

Pourtant, deux raisons semblent montrer qu'il est somme toute relativement naturel de penser un style comme un programme. Premièrement, la représentation picturale est aisément modélisable sous la forme d'un processus, et un langage de programmation est tout désigné pour décrire un processus. Deuxièmement, comme on l'a vu au chapitre 1, lorsqu'on décrit un style de manière générique, c'est-à-dire en dépassant ses effets pour analyser sa structure, c'est encore en des termes algorithmiques. Et dans ce cas, il est également facile de transposer cette description en un programme.

Notre expérience de ce schéma d'analyse de style nous a montré qu'il était en réalité simple et agréable d'envisager un style comme un programme. Il aurait fallu récolter les avis d'utilisateurs non initiés à la programmation pour valider cette affirmation, nous n'avons ici encore malheureusement pas pu mener cette expérience.

Est-il facile de reproduire un style ?

De manière générale, l'étape la plus délicate dans le processus de reproduction d'un style est la phase d'analyse menée par l'utilisateur à l'issue de laquelle le style aura été traduit en des termes algorithmiques. La décomposition en modules de style, l'identification des relations entre attributs et informations qui en résultent peuvent être plus ou moins bonnes en fonction de l'utilisateur et de la complexité du style observé. Dans le cas où le style trouve une formulation naturelle, sa transcription sous forme de modules de style sera aisée ; elle sera laborieuse dans le cas contraire.

Certaines des illustrations produites par notre système et peuplant cette thèse s'inspirent plus ou moins précisément de styles observés sur des illustrations réelles. Nous n'avons pas eu le sentiment, lors de tels exercices, d'être "bloqués", de ne pas pouvoir modéliser un style. Il serait intéressant de soumettre le système à une plus grande variété de styles encore pour mieux en évaluer les limites.

Cette approche peut-elle être étendue à l'animation ?

Telle qu'elle a été présentée, notre approche ne permet pas d'assurer la cohérence temporelle d'une animation. Ainsi, chaque image sera générée indépendamment des précédentes et des suivantes. Bien qu'il existe de nombreuses applications à la génération d'illustrations fixes, la production d'une animation reste l'une des possibilités les plus excitantes de la synthèse d'images et cette absence de gestion de la cohérence temporelle constitue en soit une limitation importante dont le reproche nous a souvent été fait. Comme nous l'avons expliqué au chapitre 1, il existe un conflit entre le maintien de la cohérence temporelle et le respect du style tel qu'il a été spécifié. Pour cette raison, une approche telle que celle introduite par Kalnins *et al.* [KDMF03], qui consiste à automatiquement modifier les attributs et la topologie des traits de manière à minimiser les artefacts visuels, n'est pas "philosophiquement" compatible avec notre approche qui offre à l'utilisateur un contrôle fin sur ces mêmes attributs. Selon la logique de cette approche, seul l'utilisateur est en mesure de spécifier le comportement a adopter dans tel ou tel cas pour maintenir la cohérence temporelle, et il faudrait intégrer cette

dernière au système programmable. Les moyens permettant d'arriver à un tel résultat font partie des travaux futurs.

8.3 Travaux futurs

Mise en volume

Notre système est actuellement limité aux dessins au trait composés des ensembles d'arêtes de notre graphe de vue. Une extension naturelle pourrait consister à inclure un traitement cohérent des lignes de tons et de hachures. De même, des travaux futurs pourraient inclure des traitements procéduraux similaires pour la mise en volume (*shading*), le système de marques et tous les autres composants NPR.

Cohérence temporelle

Notre thèse n'aborde pas le problème de la cohérence temporelle. En réalité, l'extrême flexibilité du système pourrait, dans certains cas, accentuer les problèmes de cohérence temporelle par rapport aux systèmes de rendu précédents. En particulier, il serait parfois impossible de respecter pour chaque image d'une animation à la fois les contraintes fixées par le module de style et celles induites par une gestion de la cohérence temporelle, telles qu'une paramétrisation cohérente des traits [KDMF03].

Cependant, des travaux récents en NPR pour l'animation [KDMF03, KSC+01, CTP+03], renforce notre sentiment que la question de la cohérence temporelle pourrait grandement bénéficier d'une approche programmable. Une plus grande variété de compromis et de choix stylistiques liés à la description temporelle serait par exemple ainsi accessible.

Dans le cas d'une animation prédéfinie, on pourrait imaginer inclure la dimension temporelle dans les paramètres des modules de style, de manière à intégrer la gestion des évènements de visibilité (*e.g.* fusion de deux traits, apparition d'une silhouette) à la description de style. De cette manière, même les opérations topologiques (*e.g.* découpage de traits), qui sont naturellement à l'origine de discontinuités temporelles, pourraient anticiper sur l'apparition de telles discontinuités.

Généralisation de l'approche programmable

Ce projet de recherche nous a conforté dans l'idée que les approches programmables sont particulièrement bien adaptées au domaine du NPR. En effet, comme nous l'avons mentionné à plusieurs reprises, du point de vue de l'utilisateur, le NPR représente, par rapport au rendu photoréaliste, un besoin accru de contrôle. Or, l'approche programmable est la seule à concilier contrôle flexible et automatisation. Pour cette raison, il serait intéressant de généraliser l'utilisation d'une telle approche à d'autres composants du NPR.

Rendu de marques programmable

Le système de rendu de marques utilisé à ce jour dans notre système s'appuie sur l'utilisation de textures. Il serait intéressant d'envisager d'élaborer un système programmable qui permettrait à l'utilisateur de spécifier les formes et les positions des marques sur le papier, les interactions entre le medium et le support, *etc*...

Détection programmable des lignes caractéristiques

Pour l'instant, le choix des familles de lignes caractéristiques est intégré "en dur" dans le système et se limite aux familles identifiées dans la littérature. Ici aussi, l'accès à un système programmable permettant à l'utilisateur de spécifier les caractéristiques des lignes qu'il juge d'intérêt pourrait représenter un outil de recherche performant.

Simplification de dessins au trait

Le travail sur l'information de densité présenté au chapitre 6 ouvre plusieurs pistes de pour de futures recherches. Les stratégies que nous avons proposées sont seulement des illustrations du pouvoir de l'information de densité. De nouveaux styles et de nouvelles approches pour analyser et contrôler l'encombrement pourraient faire l'objet de travaux futurs. De plus, la densité causale soulève de manière cruciale la question de l'ordonnancement des traits : quelles sont lignes caractéristiques les plus importantes pour la représentation fidèle d'un objet ? Nous pensons que la géométrie, la perception et l'analyse d'image peuvent être utilisées pour produire des estimations pertinentes de l'importance d'un trait. Finalement, l'information de densité pourrait être combinée avec de l'information de nature sémantique ou cognitive (tels que des courbes issues de systèmes de suivi de regard (*eye-tracking systems*) [DS02b]).

Table des figures

Liste des tableaux

Algorithmes

Bibliographie

Références scientifiques

[ACSD+03] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3) :485–493, 2003.

[AG99] A. Apodaca and L. Gritz, editors. *Advanced Renderman : Creating CGI for Motion Pictures*. Morgan Kaufmann, 1999.

[Ale01] Andrei Alexandrescu. *Modern C++ Design*. Addison-Wesley C++ In-Depth. Addison-Wesley Publishing Company, New York, NY, 2001.

[Apo99] A. Apodaca. Photosurrealism. In A. Apodaca and L. Gritz, editors, *Advanced Renderman : Creating CGI for Motion Pictures*. Morgan Kaufmann, 1999.

[App67a] Arthur Appel. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 1967 22nd national conference*, pages 387–393. ACM Press, 1967.

[Ari02] Michael Arias. Softimage—xsi - toon shader reference. Technical report, Avid, 2002.

[BA83] Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Trans. on Communications*, COM-31,4 :532–540, 1983.

[BDG+99] Gill Barequet, Christian A. Duncan, Michael T. Goodrich, S. Kumar, and M. Pop. Efficient perspective-accurate silhouette computation. In *Symposium on Computational Geometry*, pages 417–418, 1999.

[BE99] F. Benichou and G. Elber. Output sensitive extraction of silhouettes from polygonal geometry. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, page 60. IEEE Computer Society, 1999.

[Bea96] D. M. Beazley. SWIG : an easy to use tool for integrating scripting languages with C and C++. In USENIX [USE96], pages 129–139.

[BH98] David J. Bremer and John F. Hughes. Rapid approximate silhouette rendering of implicit surfaces. *Proceedings of Implicit Surfaces '98*, pages 155–164, June 1998.

[BKOS00] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry*. Springer-Verlag, 2000.

[Bou98] Lubomir Bourdev. Rendering nonphotorealistic strokes with temporal and arc-length coherence. Master's thesis, Brown University, 1998.

[BS00] J. Buchanan and M. Sousa. The edge buffer : A data structure for easy silhouette rendering. In *Non-Photorealistic Animation and Rendering*, 2000.

[BWL04] William V. Baxter, Jeremy Wendt, and Ming C. Lin. IMPaSTo : A realistic model for paint. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 45–56, June 2004.

[CAS+97] C. Curtis, S. Anderson, J. Seims, K. Fleischer, and D. Salesin. Computer-generated watercolor. *Proc. SIGGRAPH*, 1997.

[CHZ00] J. Cohen, J. Hughes, and R. Zeleznik. Harold : A world made of drawings. In *Non-Photorealistic Animation and Rendering*, 2000.

[CJTF98] W. Toledo Corrêa, R. Jensen, C. Thayer, and A. Finkelstein. Texture mapping for cel animation. *Proc. SIGGRAPH*, 1998.

[Coo84] Robert L. Cook. Shade trees. In *Proc. SIGGRAPH*, 1984.

[CS04] Patrick Coleman and Karan Singh. Ryan : rendering your animation nonlinearly projected. In *NPAR '04 : Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 129–156. ACM Press, 2004.

[CTP+03] Cunzi, Thollot, Paris, Debunne, Gascuel, and Durand. Dynamic canvas for non-photorealistic walkthroughs. In *Proc. Graphics Interface*, 2003.

[Cur98] C. Curtis. Loose and Sketchy Animation. In *SIGGRAPH : Conference Abstracts and Applications*, 1998.

[dC76] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Dec96a] Philippe Decaudin. Cartoon looking rendering of 3D scenes. Research Report 2919, INRIA, June 1996.

[Dec96b] Phillippe Decaudin. Cartoon-looking rendering of 3d-scenes. Technical Report INRIA 2919, Universite de Technologie de Compiegne, France, june 1996.

[DFRS03] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. on Graphics*, 22(3), 2003.

[DOM+01] F. Durand, V. Ostromoukhov, M. Miller, F. Duranleau, and J. Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Eurographics Workshop on Rendering*, 2001.

[DS00] O. Deussen and T. Strothotte. Pen-and-ink illustration of trees. *Proc. SIGGRAPH*, 2000.

[DS02a] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. *ACM Trans. on Graphics*, 21(3), 2002. (Proc. SIGGRAPH).

[DS02b] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. In *Proc. SIGGRAPH*, 2002.

[Dur02a] Durand. An invitation to discuss computer depiction. In *Proc. NPAR*, 2002.

[Elb98] Elber. Line art illustrations of parametric and implicit forms. *TVCG*, 4(1), 1998.

[EMP+94] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Worley. *Texturing and Modeling : A Procedural Approach*. Academic Press, October 1994. ISBN 0-12-228760-6.

[FA91] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. PAMI*, 13(9), 1991.

[FTP99] W. Freeman, J. Tenenbaum, and E. Pasztor. An example-based approach to style translation for line drawings. Technical Report 99-11, MERL, 1999.

[GG01] Gooch and Gooch. *Non-Photorealistic Rendering*. AK-Peters, 2001.

[GGSC98] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. *Proc. SIGGRAPH*, 1998.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements od Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, New York, NY, 1995.

[Gla91] Andrew S. Glassner. Maintaining winged-edge models. In *Graphics Gems II*, pages 191–201. Morgan Kaufman Publishers Inc., 1991.

[Gom95b] Ernst Hans Gombrich. *The Story of Art (L'histoire de l'art)*. Phaidon Press, 1995. 16th edition.

[GSG+99] B. Gooch, P. Sloan, A. Gooch, P. Shirley, and R. Riesenfeld. Interactive Technical Illustration. *ACM Symp. on Interactive 3D Graphics*, 1999.

[Hae90] P. Haeberli. Paint by numbers : Abstract image representations. *Proce. SIGGRAPH*, 1990.

[Her98] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. *Proc. SIGGRAPH*, 1998.

[Her01a] Hertzmann. Paint by relaxation. In *CGI*, pages 47–54, 2001.

[Her03] Aaron Hertzmann. Machine learning for computer graphics : A manifesto and a tutorial. In *Invited paper, Pacific Graphics*, 2003.

[HIR+03] Nick Halper, Tobias Isenberg, Felix Ritter, Bert Freudenberg, Oscar Meruvia, Stefan Schlechtweg, and Thomas Strothotte. Opennpar : A system for developing, programming, and designing non-photorealistic animation and rendering. In *Pacific Conference on Computer Graphics and Applications*, pages 424–428, 2003.

[HJO+01] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. Image analogies. *Proc. SIGGRAPH*, 2001.

[HL90] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. In *Proc. SIGGRAPH)*, 1990.

[HL94] Siu Chi Hsu and Irene H. H. Lee. Drawing and animation using skeletal strokes. *Proc. SIGGRAPH 94*, 1994.

[HM92] Chet S. Haase and Gary W. Meyer. Modeling pigmented materials for realistic image synthesis. *ACM Trans. Graph.*, 11(4) :305–335, 1992.

[HMK02] D. Hekmatzada, J. Meseth, and R. Klein. Non-photorealistic rendering of complex 3d models on mobile devices. In *8th Annual Conference of the International Association for Mathematical Geology*, volume 2, pages 93–98. Alfred-Wegener-Stiftung, September 2002.

[HOCS02] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In *EGRW '02 : Proceedings of the 13th Eurographics workshop on Rendering*, pages 233–246. Eurographics Association, 2002.

[HS99b] J. Hamel and T. Strothotte. Capturing and re-using rendition styles for non-photorealistic rendering. *Computer Graphics Forum*, 18(3) :173–182, September 1999.

[HSS02] Halper, Schlechtweg, and Strothotte. Creating non-photorealistic images the designer's way. In *Proc. NPAR*, 2002.

[HZ00] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. *Proc. SIGGRAPH*, 2000.

[IHS02] Tobias Isenberg, Nick Halper, and Thomas Strothotte. Stylizing Silhouettes at Interactive Rates : From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum (Proceedings of Eurographics)*, 21(3) :249–258, September 2002.

[IMT99] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy : A sketching interface for 3d freeform design. *Proc. SIGGRAPH*, 1999.

[JEGPO02] Pierre-Marc Jodoin, Emric Epstein, Martin Granger-Piché, and Victor Ostromoukhov. Hatching by Example : a Statistical Approach. In *Proc. NPAR*, 2002.

[J.M89] J.Martin. *Technical Illustration : Materials, Methods and Techniques*. MacDonald and Co Publishers, 1989.

[JWY02] Ming Ouhyoung Jun-Wei Yeh. Non-photorealistic rendering in chinese painting of animals. *Proc. ChinaGraph2002*, 2002.

[KDMF03] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. *ACM Trans. on Graphics*, 22(3), 2003.

[KKL+00] A. Klein, M. Kazhdan, W. Li, W. Toledo Correa, A. Finkelstein, and T.Funkhouser. Non-photorealistic virtual environments. *Proc. SIGGRAPH*, 2000.

[KMM+02] Kalnins, Markosian, Meier, Kowalski, Lee, Davidson, Webb, Hughes, and Finkelstein. Wysiwyg npr : Drawing strokes directly on 3d models. *ACM ToG*, 21(3), 2002. (Proc. SIGGRAPH).

[KMN+99] M. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. Holden, and J. Hughes. Art-based rendering of fur, grass, and trees. *Proc. SIGGRAPH*, 1999.

[Kø84] J. J. Kœnderink. What does the occluding contour tell us about solid shape ? *Perception*, 13 :321–330, 1984.

[KSC+01] Allison W. Klein, Peter-Pike J. Sloan, Alex Colburn, Adam Finkelstein, and Michael F Cohen. Video cubism. Technical Report MSR-TR-2001-45, Microsoft Research, 2001.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes : A high resolution 3d surface construction algorithm. In *SIGGRAPH '87 : Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[Lee99] Jintae Lee. Simulating oriental black-ink painting. *IEEE Comput. Graph. Appl.*, 19(3) :74–81, 1999.

[LMHB00] A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Non-Photorealistic Animation and Rendering*, 2000.

[MAZD85] E. H. Wold M. A. Z. Dippe. Antialiasing through stochastic sampling. *Proc. SIGGRAPH*, 1985.

[Mei96] B. Meier. Painterly rendering for animation. *Proc. SIGGRAPH*, 1996.

[MH04] Morgan McGuire and John F. Hughes. Hardware-determined feature edges. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 35–147. ACM Press, 2004.

[MKT+97] L. Markosian, M. Kowalski, S. Trychin, L. Bourdev, D. Goldstein, and J. Hughes. Real-time nonphotorealistic rendering. *Proc. SIGGRAPH*, 1997.

[MMK+00] L. Markosian, B. Meier, M. Kowalski, L. Holden, J.D. Northrup, and J. Hughes. Art-based rendering with continuous levels of detail. In *Non-Photorealistic Animation and Rendering*, 2000.

[Mus97] David R. Musser. Introspective sorting and selection algorithms. *Software - Practice and Experience*, 27(8) :983–993, 1997.

[NH01] Masaru Nomura and Nozomu Hamada. Feature edge extraction from 3d triangular meshes using a thining algorithm. *Proc. of SPIE*, 2001.

[NM00] J.D. Northrup and L. Markosian. Artistic silhouettes : A hybrid approach. In *Non-Photorealistic Animation and Rendering*, 2000.

[OBS04] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. on Graphics*, 2004. (Proc. SIGGRAPH).

[Ope02] Open NPAR. http ://www.opennpar.org/, 2002.

[Ost99] V. Ostromoukhov. Digital facial engraving. *Proc. SIGGRAPH*, 1999.

[PD90b] Harry Plantinga and Charles R. Dyer. Visibility, occlusion, and the aspect graph. *Int. J. Comput. Vision*, 5(2) :137–160, 1990.

[Per85] Ken Perlin. An image synthesizer. In *SIGGRAPH '85 : Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296. ACM Press, 1985.

[PHMF01] E. Praun, H. Hoppe, M., and A. Finkelstein. Real-time hatching. *Proc. SIGGRAPH*, 2001.

[PS95] Bernhard Preim and Thomas Strothotte. Tuning rendered line drawings. *Proceedings of WSCG'95*, pages 228–238, 1995.

[Ras01] Ramesh Raskar. Hardware support for non-photorealistic rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 41–47. ACM Press, 2001.

[RC99] R. Raskar and M. Cohen. Image Precision Silhouette Edges. In *Proc. 1999 ACM Symp. on Interactive 3D Graphics*, 1999.

[RKS00] Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Extraction of feature lines on triangulated surfaces using morphological operators. In Andreas Butz, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics (AAAI Spring Symposium-00)*, volume 00-04 of *Technical Report / SS / American Association for Artificial Intelligence*, pages 71–75, Stanford, USA, 2000. American Association for Artificial Intelligence, AAAI Press.

[RvE92] J. Rossignac and M. van Emmerik. Hidden contours on a frame-buffer. In *Proceedings of the 7th Eurographics Workshop on Computer Graphics Hardware*, pages 188–204, Cambridge, UK, September 1992.

[SABS94] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen-and-ink illustration. *Proceedings of SIGGRAPH 94*, 1994.

[SALS96] M. Salisbury, C. Anderson, D. Lischinski, and D. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. In *Proc. SIGGRAPH*, 1996.

[SB99b] Mario Costa Sousa and John W. Buchanan. Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics Forum*, 18(3), 1999.

[SC92] Paul S. Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. In *Computer Graphics (Proc. of SIGGRAPH 92)*, volume 26, pages 341–349, July 1992.

[SGG+00] P. Sander, X. Gu, S. Gortler, H. Hoppe, and J. Snyder. Silhouette clipping. *Proc. SIGGRAPH*, 2000.

[SGS04] Karan Singh, Cindy Grimm, and Nisha Sudarsanam. The ibar : a perspective-based camera widget. In *UIST '04 : Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 95–98. ACM Press, 2004.

[Sin02] Karan Singh. A Fresh Perspective. In *Proc. Graphics Interface*, pages 17–24, May 2002.

[SLM00] Alexander Stepanov, Meng Lee, and David Musser. *The C++ Standard Template Library*. Prentice Hall, 2000.

[SOD04] Yann Semet, Una-May O'Reilly, and Frédo Durand. An interactive artificial ant approach to non-photorealistic rendering. In *Genetic and Evolutionary Computation COnference*, 2004.

[SP03a] M. C. Sousa and P. Prusinkiewicz. A few good lines : Suggestive drawing of 3d models. *Proc. Eurographics*, 2003.

[SP03b] M.C. Sousa and P. Prusinkiewicz. A few good lines : Suggestive drawing of 3d models. *Computer Graphics Forum (Proc. of EuroGraphics '03)*, 22(3), 2003.

[SS02] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics. Modeling, Rendering, and Animation*. Morgan Kaufmann, 2002.

[ST90] T. Saito and T. Takahashi. Comprehensible Rendering of 3-D Shapes. In *Computer Graphics (Proc. SIGGRAPH)*, 1990.

[Str86] Steve Strassmann. Hairy brushes. *Siggraph*, 20(4) :225–232, August 1986.

[SWHS97] M. Salisbury, M. Wong, J. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. *Proc. SIGGRAPH*, 1997.

[Tay03] Conrad Taylor. Line thickness, a means of expression, 2003.

[Tee03] Daniel Teece. Ink line rendering for film production. In Mario Cost Sousa, editor, *Theory and practice of Non-Photorealistic Graphics : Algorithms, Methods and Production Systems, SIGGRAPH Course Notes*. SIGGRAPH, 2003.

[TF97] J. Tenenbaum and W. Freeman. Separating style and content. In *Advances in Neural Information Processing Systems*, volume 9, page 662, 1997.

[Ups89] S. Upstill. *The Renderman Companion*. Addison-Wesley, Reading, MA, 1989.

[USE96] USENIX, editor. *4th Annual Tcl/Tk Workshop*, July 1996.

[W+99] Mason Woo et al. *OpenGL programming guide : the official guide to learning OpenGL, ver. 1.2*. Addison-Wesley, Reading, MA, USA, third edition, 1999.

[WD05] John Willats and Frédo Durand. Defining pictorial style : Lessons from linguistics and computer graphics. *Axiomathes*, 15(2), 2005.

[WHS03] Der-Lor Way, S.-W. Huang, and Zen-Chung Shih. Physical-based model of ink diffusion in chinese paintings. In *WSCG*, 2003.

[Wil97a] J. Willats. *Art and Representation*. Princeton U. Pr., 1997.

[WLS02] Der-Lor Way, Yu-Ru Lin, and Zen-Chung Shih. The synthesis of trees in chinese landscape painting using silhouette and texture strokes. *Journal of WSCG Volume 10*, 2002.

[WM04] Brett Wilson and Kwan-Liu Ma. Rendering complexity in computer-generated pen-and-ink illustrations. In *NPAR*, 2004.

[WPFH02] Matthew Webb, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Fine tone control in hardware hatching. In *NPAR 2002 : Second International Symposium on Non Photorealistic Rendering*, pages 53–58. ACM SIGGRAPH / Eurographics, June 2002. ISBN 1-58113-494-0.

[WS82b] G. Wyszecki and W. S. Stiles. *Color Science : Concepts and Methods, Quantitative Data and Formulae*. John Wiley and sons, second edition, 1982.

[WS94] G. Winkenbach and D. Salesin. Computer-generated pen-and-ink illustration. *Proc. SIGGRAPH*, 1994.

[XC04] Hui Xu and Baoquan Chen. Stylized rendering of 3d scanned real world environments. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 25–34. ACM Press, 2004.

[YLLC03] Young Jung Yu, Do Hoon Lee, Young Bock Lee, and Hwan Gue Cho. Interactive rendering technique for realistic oriental painting. *Journal of WSCG*, 11(1), 2003.

[YM04] Jingyi Yu and Leonard McMillan. A framework for multiperspective rendering. In *Eurographics Symposium on Rendering*, Norrkoping, Sweden, 2004.

Arts plastiques

[AC03a] Stéphane Amache and Jérôme Combe. Deux pieds (thomas fersen). Amazing Studios, 2003.

[AC03b] Stéphane Amache and Jérôme Combe. Respire (mickey 3d). Amazing Studios, 2003.

[AC03c] Stéphane Amache and Jérôme Combe. Yalil (mickey 3d). Amazing Studios, 2003.

[Bau96] Baudoin. *Le voyage*. L'Association, 1996.

[Ben96] Ted Benoit. Blake et mortimer. Blake et Mortimer, 1996.

[Bil97] Bilal. *Visioni di fine millennio*. Hazard Edizioni, 1997.

[Cha60] Marc Chagall. Ahasuerus sends vashti away. Drawing for the bible, 1960.

[Cho03] Sylvain Chomet. Les triplettes de belleville. 2d.3D, 2003.

[CT98] Cailleteau and Tota. *Projet Atlanta*. Aquablue. Delcourt, 1998.

[CV89a] Cailleteau and Vatine. *Nao*. Aquablue. Delcourt, 1989.

[Dis37] Disney. Snow white and the seven dwarfs, 1937.

[Dis40] Disney. Pinocchio, 1940.

[Fra52] Franquin. Spirou et fantasio. Spirou et les héritiés, 1952.

[Her50] William Herdman. Interior of rosslyn chapel, 1850.

[Jac77] Edgar P. Jacobs. Blake et mortimer. Blake et Mortimer, 1947-1977.

[Lin01] Richard Linklater. Waking life, 2001.

[Mat54] Henri Matisse. Loulou ?, 1869-1954.

[Oto96] Katsuhiro Otomo. *Retour sur terre*. Mother Sarah. Delcourt, 1996.

[Pel01] Patrice Pellerin. L'épervier. Dupuis, 2001.

[Pic22] Pablo Picasso. Mère et enfant. Baltimore Museum of Art, 1922.

[SA04] Masamune Shirow and Shinji Aramaki. Appleseed. Digital Frontier Inc., 2004.

[Sat03] Marjane Satrapi. *Persepolis T4*. Persepolis. L'Association, 2003.

[SS01] Hironobu Sakaguchi and Moto Sakakibara. Final fantasy : The spirits within. Square Pictures, 2001.

[SWHW95] Ray Smith, Ray Wright, James Horton, and Michael Wright. *An Introduction to Art Techniques*. Dorling Kindersley Limited, 1995.

[Tan02] Jiro Taniguchi. *Tome 1*. Quartier lointain. Casterman, 2002.

[Tar90] Tardi. *Une gueule de bois en plomb*. Nestor Burma. Casterman, 1990.

[TJ94] Tome and Janry. *Le petit spirou*. Dupuis, 1994.

[TJ98] Tome and Janry. *Machine qui rêve*. Spirou et Fantasio. Dupuis, 1998.

[VH93] Valles and Van Hamme. *Margrit 1886*. Les Maîtres de l'Orge. Glénat, 1993.

[VH95] Valles and Van Hamme. *Noel 1932*. Les Maîtres de l'Orge. Glénat, 1995.

Cette thèse s'intéresse à la génération d'illustrations "non-photoréalistes" (imitant par exemple des dessins ou des peintures) à partir de scènes 3D. L'abstraction ou la stylisation apportent à ce type de rendu des qualités communicatives, esthétiques et expressives qui le distinguent de la synthèse d'images classique.

Les objectifs sont, d'une part, de fournir à l'utilisateur un contrôle flexible sur le style du rendu non-photoréaliste et, d'autre part, de proposer une formulation du style qui en permette la réutilisation pour le rendu de différentes scènes 3D.

Nous avons choisi d'adopter une approche programmable qui s'appuie sur le postulat selon lequel les attributs de style (couleur, épaisseur, ...) sont choisis en fonction d'informations génériques de la scène (nature des lignes, discontinuité en profondeur, ...), et s'inspire des approches procédurales (telles que Pixar Renderman). L'idée consiste à exprimer un style comme un ensemble de procédures spécifiant les relations attributs/informations : l'utilisateur "programme" une feuille de style qui peut ensuite être utilisée pour le rendu de plusieurs scènes 3D différentes ou de plusieurs images d'une séquence animée. Cette approche est la première à offrir à la fois un contrôle flexible sur le style du rendu et une formulation réutilisable de ce style.

This dissertation deals with the rendering of "non-photorealistic" illustrations (imitating drawings or paintings for instance) from 3D scenes. Abstraction or stylization enhance this type of rendering with communicative, aesthetic and expressive qualities that distinguish it from usual image synthesis.

Our goals is to provide the user with a flexible control over the style of the non-photorealistic rendering and, also, to propose a style formulation that allows the reuse of the same style to render different 3D scenes.

We chose a programmable approach that relies on the assumption that style attributes (color, thickness, ...) are chosen depending on generic information from the scene (line charasteristics, depth discontinuity, ...), and takes inspiration from procedural approaches (such as Pixar Renderman). The idea consists in expressing a style as a set of procedures specifying the relations between attributes and information : The user "programs" a style sheet that can be used to render several different 3D scenes or several images from an animated sequence. This approach is the first one to offer both flexible control over the style of the rendering and a reusable formulation of that style.