HAL
open science

Pré-traitement de grosses bases de données pour la visualisation interactive

Xavier Décoret

To cite this version:

Xavier Décoret. Pré-traitement de grosses bases de données pour la visualisation interactive. Interface homme-machine [cs.HC]. Université Joseph-Fourier - Grenoble I, 2002. Français. NNT : . tel-00528890

HAL Id: tel-00528890

https://theses.hal.science/tel-00528890

Submitted on 22 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Joseph Fourier de Grenoble (UJF)

Pré-traitement de grosses bases de données pour la visualisation interactive

Xavier Décoret

Thèse présentée 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
*i*MAGIS-GRAVIR/IMAG-INRIA. UMR CNRS C5527.

Composition du jury :

François	Sillion	Directeur de thèse
Fabrice	Neyret	Président du Jury
Daniel	Cohen-Or	Rapporteur
Markus	Gross	Rapporteur
Hugues	Hoppe	Rapporteur

It takes five hundred small details to make one favorable impression.

Cary Grant

Introduction

Notre époque est caractérisée par l'explosion de la quantité d'information disponible. À l'origine de ce développement, les ordinateurs ne peuvent cependant pas résoudre tous les problèmes que pose la *représentation*, le *stockage* et le *traitement* de cette information. Les ordinateurs d'aujourd'hui montrent assez rapidement leurs limites, et ceux toujours plus puissants qu'on est en droit d'attendre pour demain n'y changeront rien. Car les possibilités créent les besoins et ces besoins excéderont toujours les capacités des machines les plus modernes.

Un domaine dans lequel ceci est particulièrement sensible est celui des *environnements virtuels*. Popularisés par les jeux vidéos et le cinéma, ces univers sont des maquettes d'environnements réels ou imaginaires dans lesquels les utilisateurs peuvent se promener et interagir comme s'il s'y trouvaient. Mais ici, tout est virtuel : la maquette n'existe que dans l'ordinateur sous forme d'information numérique. L'ordinateur a la charge de générer, à partir de cette maquette ou *modèle*, les images, les sons et éventuellement d'autres stimuli qui donneront à l'utilisateur l'impression d'être plongé dans cet univers.

Malheureusement, le sentiment d'immersion a un coût. Pour satisfaire l'utilisateur, il faut des environnements de plus en plus grands et de plus en plus richement détaillés. Si les modèles de petite taille (intérieur d'un bâtiment) et pauvre en détails visuels (sols et murs plats, peu d'objets) ont fait les succès des premiers environnements virtuels, ils sont aujourd'hui de moins en moins satisfaisants aux yeux d'un public qui s'est familiarisé avec cette technologie. Il en va de même pour la qualité des images produites. La quête du réalisme nécessite des images de haute résolution et la prise en compte, lors de la *synthèse* de ces images à partir du modèle numérique, d'effets visuels complexes comme les effets d'éclairage.

Or si la quantité d'information à traiter - complexité des modèles - et le temps de traitement - complexité de la simulation - augmentent, le temps disponible pour traiter cette information reste le même. En effet, l'ordinateur doit générer des images à une fréquence minimale de 25 Hz (c'est-à-dire la fréquence d'affichage d'un film au cinéma), l'idéal étant des fréquences plus élevées de 60 à 100 Hz. Il faut donc arriver à produire le plus rapidement possible des images de la meilleure qualité possible, à partir de grosses bases de données représentant des univers virtuels étendus et richement détaillés.

Pour cela, nous proposons de pré-traiter ces bases de données afin d'optimiser l'information qui y est représentée, et l'usage que l'on en fait.

Contexte

La vie d'un modèle virtuel comporte trois étapes : la production, le stockage et la transmission, et enfin le traitement. À chaque étape, les contraintes et les objectifs différents ; l'information que contient le modèle, et l'usage que l'on en fait aussi.

Production

Les modèles tridimensionnels d'environnements virtuels sont produits de deux manières. Soit « à la main » en utilisant des logiciels de modélisation. Soit de manière semi-automatique en utilisant des appareils d'acquisition tels que les scanners 3D, ou des algorithmes de reconstruction, à partir de photographies par exemple.

La première manière est la source de la plupart des modèles disponibles actuellement. Pour de tels modèles, on dispose généralement de beaucoup d'information : la forme du modèle et son apparence, mais aussi l'organisation spatiale du modèle, la hiérarchisation des éléments qui le composent et même éventuellement l'historique de sa construction. Ces informations sont en général saisies par le modeleur car elles lui facilitent la tâche. À l'inverse, les modèles obtenus automatiquement sont en général beaucoup plus pauvres en informations de haut niveau, même si ils sont souvent beaucoup plus détaillés géométriquement. Un scanner 3D fournit par exemple une collection de milliers -voire de millions- de facettes localisées dans l'espace, et sur lesquelles est éventuellement plaquée une photographie.

Stockage et transmission

Ces données doivent ensuite être stockées. Cela peut parfois représenter un véritable défi comme en témoigne le *Digital Michaelangelo Project* [LPC+00a] ou les travaux de modélisation de Los Angeles de l'*Urban Simulation Team* [UST]. Cependant, la taille des données n'est pas le problème *en soi*. Il est en effet encore possible de créer des espaces de stockage suffisants pour contenir de tels modèles. Par contre, l'utilisateur moyen ne disposant pas de tels espaces sur son ordinateur, il lui est impossible de disposer d'une copie des données. Il faut donc utiliser des systèmes clients/serveurs pour accéder à l'information. Se pose alors le problème de la transmission des données : on doit d'une part ne transmettre que ce qui est utile au client, et d'autre part réduire le temps d'initialisation. Le client doit pouvoir commencer à naviguer dès sa connexion au serveur, sans attendre un chargement initial de plusieurs minutes.

Traitement

Les traitements peuvent être divers, de l'affichage simple, à la simulation d'évènements complexes en réponse aux actions de l'utilisateur : déplacement ou déformation d'objets, modification des conditions de navigation (éclairage, conditions climatiques, *etc*...).

Selon le traitement, différents *niveaux* d'informations seront utilisés. On utilisera d'une part différents *types* d'informations : forme et couleur pour l'affichage, propriétés physiques pour une simulation, direction et intensité d'une force pour des systèmes haptiques à retour d'effort. On travaillera d'autre part avec différentes *granularités* d'informations. Certaines parties importantes d'un modèle seront ainsi finement représentées, et d'autres plus grossièrement.

En outre, on peut distinguer deux types de traitements : les traitements qui produisent un résultat à partir du modèle, et les traitements qui modifient le modèle. Dans la première catégorie, on trouve par exemple l'affichage, qui génère une image à partir du modèle tri-dimensionnel, ou encore certains calculs d'éclairage (lorsque la source lumineuse est fixe et que donc les ombres ne changent pas). Dans la deuxième catégorie, on trouve notamment tous les calculs d'animations, qui modifient le modèle (position des objets, position et caractéristiques des sources lumineuses) au cours du temps. On trouve aussi par exemple les calculs de radiosité qui subdivisent le modèle initial pour représenter les limites d'ombres. Selon la forme sous laquelle est représenté le modèle, ces traitements sont plus ou moins faciles.

Cadre de travail

On considère un modèle 3D d'un environnement virtuel. Ce modèle est donné sous forme d'un ensemble de polygones et de textures (sortes de papiers peints qui tapissent les polygones). Pour se placer dans un cas général, on considère que cet ensemble est non organisé : aucune information hiérarchique ou sémantique n'est disponible. En effet, même lorsque cette information est présente lors de la fabrication du modèle, elle est souvent perdue en cours de route à cause notamment des problèmes de format de fichiers et des pertes dues aux multiples conversions nécessaires. En outre, dans le cas de modèles obtenus à partir de scanner ou d'algorithmes de reconstruction, cette information n'est pas disponible.

Le modèle considéré doit ensuite être affiché à une fréquence minimum de 25Hz. C'est-à-dire que, 25 fois par seconde, il faut générer une image en réponse aux actions de l'utilisateur (déplacement du point de vue, interactions avec le modèle, animation de certains objets). Actuellement, cela est effectué par le moteur graphique d'une carte spécialisée qui projette les polygones texturés sur l'écran de l'ordinateur, comme s'ils étaient vu par une caméra virtuelle représentant l'oeil de l'utilisateur. La capacité de ces cartes, bien qu'impressionnante et en croissance exponentielle, est cependant limitée et lorsque trop de polygones texturés doivent être projetés, il n'est plus possible de mettre à jour l'affichage à la fréquence souhaitée.

Pour garantir une mise à jour interactive, il y a plusieurs directions possibles. Tout d'abord il y a le développement des capacités de ces cartes accélératrices 3D et de leurs processeurs spécialisés, mais on ne peut pas y contribuer directement. Ensuite, il y a le développement d'algorithmes performants pour traiter les problèmes liés au passage de la 3D à la 2D, notamment l'élimination des parties cachées et les problèmes d'anti-aliassage. Enfin, on peut chercher à extraire de ces

calculs *dynamiques* une partie *statique* qu'on pourra donc pré-calculer. Ce sont ces deux dernières voies que nous avons choisi d'explorer.

Motivations

Nous avons été motivés par le fait qu'algorithmes et pré-traitement sont à notre avis liés au problème de la *représentation adaptée* de l'information. Autrement dit, comment *représenter* un modèle 3D afin de pouvoir l'afficher le plus rapidement possible ? La réponse à cette question dépend de plusieurs choses : le coût (stockage et traitement) des diverses représentations, les possibilités des machines et le processus de génération d'image (quelle information est utilisée et comment ?).

Comment représenter pose aussi la question de *quoi* représenter. Si la production d'image ne nécessite *a priori* que de l'information de forme et d'apparence, nous pensons qu'une information de plus haut niveau -sémantique notamment- peut conduire à des algorithmes et des traitements plus efficaces. Savoir que tels polygones appartiennent à un même objet et que cet objet est une maison, faisant elle-même partie d'un quartier dont on connaît l'étendue spatiale, donne par exemple une information hiérarchique qui peut-être exploitée avantageusement.

La question de la représentation est une question d'actualité car la représentation polygonale majoritairement utilisée est remise en cause par l'apparition de nouvelles méthodes de rendu à base d'image (*Image Based Rendering*) ou de rendu par points (*Point Based Rendering*). Des outils sont alors nécessaires pour comprendre, comparer et éventuellement combiner ces différentes méthodes.

Les enjeux sont énormes car les applications de la réalité virtuelle sont très nombreuses : simulateur d'entraînement, tourisme virtuel, planification d'urbanisation, jeux vidéos et tous les domaines pour lesquels l'immersion dans un univers existant, ayant existé ou complètement fantaisiste permet d'apprendre, de visiter, de se faire une idée ou de se distraire. Le marché est en plein essor et pour s'y retrouver dans la quantité de techniques, d'astuces, de variations et d'extensions qui apparaissent chaque jour, il faut prendre du recul et disposer d'une analyse synthétique du problème.

Organisation du manuscrit

Pour atteindre notre objectif, réduire la complexité des données à traiter, nous avons exploré trois voies. La première consiste en l'élimination des traitements inutiles ou redondants. Le plus important d'entre eux est lié au problème des parties cachées, qui est exposé au chapitre 1. Nous y présentons une analyse complète d'un algorithme publié auquel nous avons collaboré. La démonstration d'un résultat théorique est ensuite faite, qui constitue la principale contribution du chapitre. Ce résultat est exploité dans un nouvel algorithme de pré-calcul de la visibilité.

Une fois qu'a été détermine de manière efficace ce qui doit être affiché, il reste deux façons d'accélérer les traitements : simplifier les objets à traiter et utiliser

des représentations efficaces et adaptées aux différents traitements. Les voies que nous avons exploré dans ce sens sont la *simplification de maillage* et le *rendu à base d'image*. Le chapitre 2 présente la problématique de ces approches, et fait le tour des méthodes existantes. À partir de ces travaux, nous proposons dans le chapitre 3 une solution nouvelle à la croisée de la simplification de maillage et du rendu à base d'images. Les principales contributions de ce chapitre sont une nouvelle représentation, appellée *nuages de billboards*, et un algorithme automatique de fabrication à partir d'un modèle quelconque. Cette nouvelle représentation constitue le coeur de la thèse ; de nombreuses applications sont mises en avant et des pistes de recherches sont ouvertes.

Après avoir résumé les différentes contributions de la thèse, nous concluerons sur l'intégration de nos travaux dans un système complet de visualisation interactive de grosses bases de données, et dégagerons les perspectives sur lesquelles nous souhaitons continuer.

L'ouvrier qui veut bien faire son travail doit commencer par aiguiser ses instruments.

Confucius

Notations

Précisons les notations que nous utiliserons dans ce document. Les majuscules italiques désignent des points, les minuscules en gras désignent les vecteurs. Les coordonnées d'un point ou d'un vecteur sont notées x, y, z indicées par le nom du point où du vecteur. Quand il n'y a pas d'ambiguités, l'indice est omis. Les lettres cursives sont utilisées pour désigner des ensembles (un plan par exemple, est un ensemble de points). On désignera la transposée d'un vecteur par un T en exposant à droite. Le tableau ci-dessous résume les notations :

M	un point de l'espace
$\mathbf{u}$	un vecteur de l'espace
$\overrightarrow{MN}$	vecteur joignant deux point M et N
$\mathbf{u}.\mathbf{v}$	le produit scalaire de deux vecteurs
(x_M, y_M, z_M)	coordonnées cartésiennes du point M
$(x_\mathbf{u}, y_\mathbf{u}, z_\mathbf{u})$	coordonnées cartésiennes du vecteur $\mathbf{u}$
$\mathcal{P}$	un ensemble de points (par exemple un plan)
$\mathbf{n}^T$	transposée du vecteur $\mathbf{n}$
$\lvert\mathbf{n}\rvert$	norme du vecteur $\mathbf{n}$

Ce que l'on ne voit pas, on peut l'ignorer

Graham Greene

1 Calcul de visibilité

Dans ce chapitre, nous traitons du problème de l'élimination des parties cachées. Lorsqu'un modèle est affiché, certaines parties ne sont pas visibles. Les parties situées hors du champ de vue par exemple, ou encore les objets qui sont cachés par des objets opaques situés devant. L'objectif étant d'accélérer le rendu de modèle, il faut éliminer le plus tôt possible ces parties des algorithmes de traitement. Pour cela, nous commençons par présenter en détail le problème de l'élimination des parties cachées, de manière intuitive, puis de manière plus formelle. Nous nous attacherons à étudier en détail deux algorithmes précis qui ont guidé nos travaux. Fort de cette analyse, nous présentons ensuite les résultats auxquels nous sommes arrivés, et notre contribution à la résolution du problème de la visibilité.

1.1 La problématique

Le problème de la visibilité est bien exprimé par l'algorithme dit « du peintre ». Lorsqu'il réalise une peinture, l'artiste commence par peindre le fond de la toile aux couleurs du ciel et du sol. Il dessine ensuite successivement les objets, du plus lointain au plus proche, chaque nouvel objet recouvrant éventuellement les objets distants déjà peints. Pour peindre un mur blanc, un artiste rigoureux mais peu soucieux d'efficacité commencera donc par dessiner les collines et le village au lointain, puis la maison et le jardin qui se trouve derrière le mur et enfin, recouvrira tout cela par un grand rectangle blanc ! L'ordinateur est comme cet artiste, car il n'a aucune intuition du résultat et ce n'est qu'une fois le mur peint qu'on se rend compte que celui-ci cache tout ce qu'il a déjà dessiné.

Pour éviter le dessin inutile de ces parties « cachées », un pré-traitement peut

être appliqué au modèle pour déterminer ce qui sera au final visible dans une vue donnée du modèle.

Déterminer les parties d'un modèle qui sont visibles d'un point de vue donné, c'est-à-dire celle que l'on doit dessiner pour obtenir une image, est un problème inhérent à la synthèse d'image. Un certain nombre de méthodes ont été développées pour le résoudre. Les algorithmes de lancer de rayons et assimilé s'inspirent de la propagation des rayons optiques dans la nature en déterminant les objets qui sont rencontrés par des rayons lumineux issus du point de vue. Ces méthodes permettent en outre de calculer des phénomènes complexes contribuant fortement au réalisme d'une image virtuelle (ombre, réflection spéculaire, diffusion). Malheureusement elles sont assez coûteuses en calcul et ne sont actuellement pas assez rapides pour le temps réel. L'algorithme de *z-buffer*, bien que proposé très tôt [Cat74], n'a pu réellement être mis en œuvre que plus tard, dans les années 80 [1], quand les progrès matériels ont rendu disponibles les larges tampons mémoires qu'il nécessite. Aujourd'hui, il s'est imposé par sa simplicité de mise en œuvre et les possibilité d'accélération matérielle. Pour atteindre notre objectif de temps réel, nous cherchons à augmenter le rendement de cet élément.

En effet, l'algorithme du *z-buffer* est sous-optimal car il « peint » plusieurs fois certains pixels. L'optimalité est atteinte quand chaque pixel n'est peint qu'une fois, auquel cas on n'a dessiné que ce qui est visible. On cherche donc à minimiser le taux de recouvrement de pixels par des pixels situés devant. Pour cela on met au point des tests qui permettent d'éliminer rapidement des primitives graphiques avant que celles-ci ne soient transformées en un ensemble de pixels. Ces tests peuvent se faire au vol en fonction du point de vue courant, ou à l'avance en prévision des points de vue potentiels. Ces deux possibilités sont détaillées dans les sections suivantes.

1.1.1 Calcul dynamique

Les images d'un modèle sont obtenues en projetant sur l'écran de la caméra les primitives qui le composent [FvDFH90]. Les seules primitives qui peuvent contribuer à l'image sont alors celles situées dans la pyramide demi-tronquée [2] définie par le centre de projection et le plan de la caméra (voir Figure 1.1). On peut alors ignorer directement les primitives entièrement situées à l'extérieur de cette pyramide [Cla76]. Une hiérarchie de boîtes englobantes permet de rendre ce test très efficace [GBW90] et d'éliminer très rapidement beaucoup de parties cachées.

Cette approche est possible dynamiquement car les tests d'intersection avec la pyramide sont très rapides [AM00]. Des tests plus coûteux ne peuvent être effec-

[1] `http://www.siggraph.org/publications/newsletter/v32n1/contributions/baum.html`

[2] En réalité, à cause du nombre fini de bits et donc de la précision limitée du tampon stockant la profondeur associée à chaque pixel, on doit aussi limiter cette pyramide par un plan distant. Sans restriction, on considérera qu'aucune partie du modèle n'est située derrière ce plan, ce qui est toujours possible car le modèle est borné.

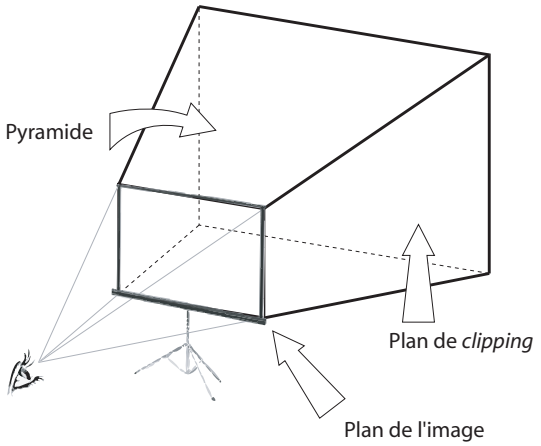


FIG. 1.1 – Pyramide de rendu

tués que s'ils restent pertinents suffisamment longtemps dans le temps pour pouvoir amortir leur coût sur la génération de plusieurs images. Pour cela, on peut par exemple calculer à un instant donné un ensemble de primitives qui sont certainement cachées pour tout point de vue à moins d'une certaine distance du point de vue courant. Comme le prochain point de vue sera par continuité parmi ceux-là, les résultats du calcul seront pertinents pendant un certain temps. C'est ce qu'exploite l'algorithme de « visibilité instantanée » de Wimmer et Wonka [WWS01]. Les auteurs proposent de calculer l'ensemble des objets potentiellement visibles (*Potentially Visible Set* ou PVS en abrégé dans la suite) pour une région donnée de l'espace, en utilisant une propriété de réduction des bloqueurs [WWS00]. Leur solution est valide pour des modèles 2D1/2 pour lesquels les calculs une fois amortis sur plusieurs images respectent le budget de calcul alloué par image.

Cette idée de calculer l'ensemble des objets potentiellement visibles d'une région donnée de l'espace est aussi proposée dans d'autres méthodes. Cependant, les calculs plus coûteux sur lesquels elles se fondent ne les rendent envisageables que dans le cas d'un pré-calcul, dont l'intérêt est exposé dans la section suivante.

1.1.2 Calcul statique

Nous disposons d'une méthode permettant de calculer la visibilité d'un point de vue donné. C'est le *z-buffer*. Malheureusement cette technique n'est utilisable que dynamiquement, car le nombre de point de vue potentiels est infini et il est donc impossible de pré-calculer ce qui est visible de chacun d'entre eux. Par contre, il est possible de recouvrir ces point de vues par un ensemble *fini* de régions de diamètres arbitrairement petits[3]. Ces régions sont appelées *cellules de visibilité*. Si

[3]On considère que l'on navigue dans un modèle borné et qu'en conséquence l'ensemble des points de vue possibles est borné, donc son adhérence est compacte, donc recouvrable par un en-

l'on a déterminé les PVS de ces cellules, on peut alors les utiliser dynamiquement en retrouvant simplement la cellule qui contient le point de vue courant. La propriété à garantir lorsque l'on calcule un PVS dynamiquement ou statiquement est la *conservativité*. Les objets du PVS ne sont en effet que « potentiellement » visibles ; certains peuvent être en réalité invisibles de la cellule. L'élimination des parties cachées étant, en aval du PVS, garantie par l'utilisation du *z-buffer*, cela n'aura pas d'impact sur l'image finale, mais seulement sur le coût du *z-buffer*. Par contre, il ne faut pas classifier comme caché un objet en réalité visible. En résumé, l'ensemble des objets *potentiellement* visibles d'une cellule doit contenir l'ensemble des objets *réellement* visibles de cette cellule.

Un certain nombre de méthodes plus ou moins efficaces ont été développées, mais nous ne les détaillerons pas ici. Pour une étude plus complète, le lecteur peut se référer à la thèse de Frédo Durand [Dur99]. Dans ce chapitre, nous proposons une méthode inspirée par notre réflexion sur la représentation de l'information. Cette méthode transforme le modèle en une structure de donnée adaptée à un calcul efficace de la visibilité. Elle s'appuie sur les travaux de Schaufler [SDDS00] auxquels nous avons collaboré, et étend les études sur les bloqueurs réduits de Wimmer et Wonka [WWS00] au cas 3D.

1.2 Analyse préliminaire

1.2.1 Analogie lumineuse

Le problème est le suivant. On a une scène composée d'un certain nombre d'objets. On considère une région de l'espace - la cellule de visibilité - et on cherche les objets visibles de cette région. Si l'on place dans la scène une source lumineuse omni-directionnelle occupant exactement le volume de la cellule, les objets recherchés sont ceux que la source éclaire directement. Déterminer ce qui est caché est équivalent à déterminer ce qui se trouve dans l'*ombre* (en négligeant les effets radiatifs). Calculs de visibilité et calculs d'ombre sont donc analogues, ce qui nous conduit à reformuler notre problème de la façon suivante :

> *Étant données deux régions, l'une appelée* source *et l'autre* récepteur, *si tout rayon lumineux joignant un point de la source et un point du récepteur intersecte un des objets de la scène, alors le récepteur n'est pas visible de la source. Un objet qui intersecte un tel rayon est appelé* bloqueur.

Dans la formulation initiale, la source est la cellule, et l'on considère successivement comme récepteurs les objets de la scène. La notion de récepteur est cependant plus large car on peut vouloir considérer un groupe d'objets, ou encore un volume englobant d'un tel groupe, et plus généralement une région quelconque de l'espace. Ce dernier cas est par exemple intéressant si l'on veut tester dynamiquement la visibilité d'objets en déplacement.

semble fini de compacts.

On notera que la source et le récepteur jouent des rôles symétriques. On dira ainsi que la source ne « voit » pas les parties du récepteur situées dans l'ombre, et réciproquement, que les points du récepteur situés dans l'ombre ne « voient » pas la source.

1.2.2 Fusion des pénombres

Comme il n'est pas possible algorithmiquement de considérer l'ensemble infini des rayons joignant la source et le récepteur, on doit considérer tous ces rayons dans leur ensemble et déterminer si ce *faisceau* est bloqué par les objets de la scène. Le problème est que le faisceau n'est pas nécessairement bloqué par un objet unique de la scène, mais généralement par un ensemble d'objets. C'est ce que l'on appelle la *fusion des pénombres*.

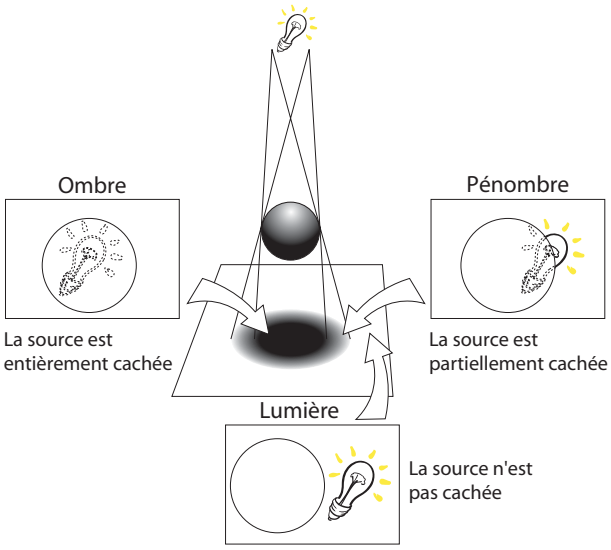


FIG. 1.2 – Ombre et pénombre

En chaque point du plan, la source est soit visible, soit entièrement cachée, soit partiellement cachée. Le point est alors en pleine lumière, dans l'ombre ou dans la pénombre. Dans ce dernier cas, le pourcentage de la source visible détermine l'intensité de la pénombre.

La pénombre d'un objet est l'ensemble des points pour lesquels la source n'est que partiellement cachée par l'objet. En ces points l'intensité lumineuse reçue n'est pas maximale, et l'on a donc, entre les zones qui ne voient pas du tout la source, et celles qui la voient entièrement, un « dégradé d'ombre » comme illustré sur la Figure 1.2. Considérons une scène avec un objet, et un point dans la pénombre de cet objet. En ce point P, une partie $\mathcal{S}_1$ de la source est cachée par l'objet (Figure 1.3(a)). Rajoutons maintenant un deuxième objet dans la scène. Pour notre point, il cache une partie $\mathcal{S}_2$ de la source, que l'on suppose non nulle, et non égale à la source (Figure 1.3(b)). Notre point est donc dans la pénombre de deux objets et

dans l'ombre d'aucun d'entre eux. Et pourtant comme $\mathcal{S}_1 \cup \mathcal{S}_2$ contient la source, le point P est dans l'ombre des deux objets (Figure 1.3(c)).

De manière génerale, si en un point, l'union des parties de la source cachées respectivement par chaque objet de la scène recouvre la source, alors le point est dans l'ombre. C'est cette action cumulée des bloqueurs qui est responsable de la plus grande partie des ombres dans une scène. Et tout particulièrement lorsque les objets sont petits en comparaison de la taille de la source et du récepteur.

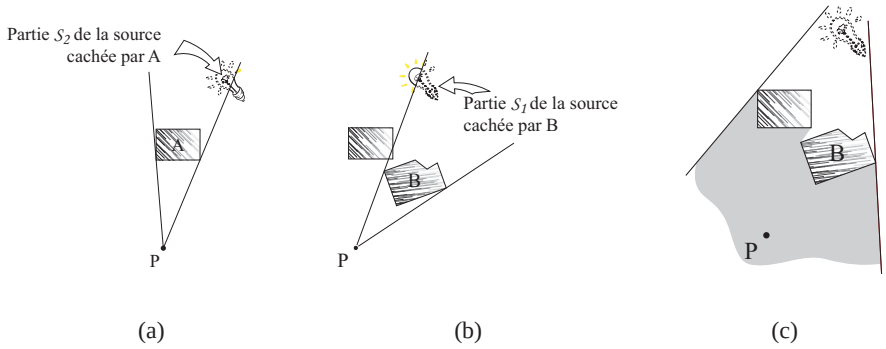


FIG. 1.3 – Fusion des pénombres

1.2.3 Contraintes algorithmiques

La seule chose sur laquelle on peut travailler algorithmiquement sont les objets de la scène car ils sont spécifiés par un nombre fini de coefficients (par exemple les coordonnées des sommets pour des objets polygonaux, ou les coefficients de la fonction potentiel pour des surfaces implicites). Pour déterminer si le faisceau est bloqué par un groupe d'objets, il faut alors nécesairement disposer d'une condition analytique, c'est-à-dire d'une fonction définie sur l'espace des coefficients, à valeur dans $\{0, 1\}$ et calculable en un nombre fini (et raisonnable) d'opérations.

Un algorithme de calcul de visibilité, pour un point où une région de l'espace, effectuera un certain nombre de fois les deux étapes suivantes :

une étape algorithmique qui sélectionne un groupe d'objets candidats pour bloquer le faisceau (ou décide d'arrêter le calcul) ;

une étape analytique qui détermine si le faisceau est bloqué.

La première étape est appelée *sélection des bloqueurs*. Si la scène comporte n objets, le nombre de groupes à considérer *a priori* est $\sum_{k=1}^{n} C_n^k = 2^n$ et on ne peut tous les traiter. La première difficulté est donc de définir une heuristique qui sélectionne des groupes. Tout la difficulté est de ne pas s'arrêter avant d'avoir considéré

le groupe qui bloque le faisceau (s'il existe), tout en considérant un nombre raisonnable de groupes. Par exemple on peut choisir une des deux heuristiques naïves suivantes : on effectue autant de fois les deux étapes qu'il y a d'objets, en ne considérant qu'un objet à la fois, ou bien on effectue une seule fois les deux étapes en considérant tout les objets à la fois.

Dans la deuxième étape, la difficulté vient du fait que la condition analytique n'est pas disponible dans le cas général. Dans la section suivante, nous exposerons un certain nombre de cas pour lesquels on dispose d'une condition conservative. Remarquons pour l'instant que l'expression analytique conditionne la sélection des bloqueurs. En effet, si l'on dispose d'une condition dans le cas général, on peut considérer directement l'ensemble des objets. Par contre, si la condition n'est valable que pour des cas particuliers, il faudra sélectionner des groupes qui satisfont les contraintes définissant ces cas.

Il est à noter que l'on peut contourner le problème de fusion des pénombres en divisant le faisceau en sous-faisceaux. À la limite de ce partitionnement, on retombe sur des rayons lumineux, pour chacun desquels il suffit de tester bloqueur par bloqueur s'il est intersecté. En pratique le niveau de subdivision requis pour approcher de cette limite est prohibitif [4]. Signalons cependant qu'il existe une partition $(\mathcal{S}_i)_{i=1\ldots m}$ de la source et une partition $(\mathcal{R}_j)_{j=1\ldots n}$ du récepteur telle que pour i, j donnés le sous-faisceau joignant $\mathcal{S}_i$ et $\mathcal{R}_j$ est soit entièrement non bloqué, soit entièrement bloqué par un même objet. Cette partition est donnée par le graphe d'aspect [PD90]. Malheureusement, l'algorithme permettant de le calculer a une complexité $O(n^{10})$ [GCS91].

Remarquons enfin qu'une façon de déterminer si le faisceau est bloqué est de calculer explicitement l'ombre de la source dans la scène. Dans le cas général, cette approche n'a que peu de chance d'être applicable. En effet, pour une source polyhédrique, l'ombre n'est pas délimitée seulement par des plans, mais aussi par des surfaces quadratiques [Tel92] qui sont difficiles à stocker et à intersecter. En outre, la fusion des pénombres nécessite de stocker les parties de la source cachées par différents objets de la scène, puis d'en réaliser l'union.

1.2.4 Conditions analytiques

Si l'on ne dispose pas dans le cas général d'une expression analytique permettant de déterminer si un faisceau est bloqué par un groupe d'objets donnés, un certains nombres de travaux proposent des expressions pour des cas particuliers. Dans ce qui suit, nous considérons que la source, le récepteur et les bloqueurs sont des polygones [5].

Si le bloqueur est convexe, le faisceau est bloqué si et seulement si tous les rayons particuliers joignant les sommets de la source et du récepteur intersectent

[4] Ce niveau de subdivision est donné par la résolution de l'image calculée. Cela revient à faire du lancer de rayon.

[5] Plus exactement, la source et le récepteur sont polyédriques, mais l'on peut se ramener à des polygones en considérant la section de la source qui voit le récepteur et réciproquement.

le bloqueur. Cohen-or *et al.* [CZ98] utilisent cette propriété. Leur heuristique de recherche de groupes de bloqueurs est simple : ils ne considèrent qu'un seul bloqueur, en commençant par les plus gros et les plus proches de la source, et en s'arrêtant lorsqu'un budget de calcul est dépassé. Remarquons que cette méthode revient à calculer le *cône d'ombre* du bloqueur convexe, c'est-à-dire l'ensemble des points d'où la source est entièrement cachée par le bloqueur. En décomposant un bloqueur quelconque en bloqueurs convexes, on réalise donc la fusion des ombres et non pas des pénombres. C'est ce qui explique que cette méthode ne donne pas de résultats significatifs en l'absence de gros bloqueurs convexes.

Dans le cas bidimensionnel [6], Bittner *et al* donnent l'expression du sous-faisceau bloqué par un triangle [BWW01]. Ils donnent une méthode pour calculer efficacement l'union des sous-faisceaux en utilisant le matériel graphique, même si le calcul pourrait se faire analytiquement. La Figure 1.4 donne brièvement le principe de leur méthode. L'extension au cas 3D pose des problèmes calculatoires car les sous-faisceaux à déterminer ne sont plus des polygones 2D mais des volumes d'un espace 4D [Dur99].

Dans le cas de scènes comportant des hublots, Teller propose une méthode permettant de réduire le faisceau à considérer entre la source et le récepteur en éliminant les parties bloquées par une séquence de hublots [TS91]. Citons enfin les travaux de Green *et al* [GKM93] qui proposent une version hiérarchique du *z-buffer*. Une pyramide est construite à partir de la carte de profondeur et utilisée pour rejeter rapidement des objets cachés sans avoir à les rasteriser dans le *z-buffer*. Zhang *et al* étendent cette idée avec les *Hierarchical Occlusion Maps* [ZMHKEH97].

Nous allons maintenant étudier en détail deux algorithmes dont nous sommes partis pour notre méthode de calcul de visibilité. Le premier est issu d'un travail auquel nous avons collaboré. C'est un algorithme très « pragmatique », relativement simple à comprendre si l'on souhaite l'implémenter. En complément de la publication, nous avons souhaité analyser plus en détail les résultats qu'il exploite, afin de comprendre quelles sont ses forces et ses faiblesses, et comprendre comment il réalise la fusion des pénombres. Le deuxième algorithme est basé sur un résultat théorique très intéressant, qui permet d'effectuer très efficacement la fusion des pénombres. En étendant ce résultat, et en s'inspirant des points qui font la force du premier algorithme, nous avons développé une méthode originale que nous présentons ici.

1.3 Visibilité volumétrique

Schaufler *et al* proposent une méthode de pré-calcul du PVS d'une cellule basée sur une discrétisation de l'espace et une représentation volumétrique des bloqueurs [SDDS00].

Le modèle polygonal est d'abord voxelisé à une résolution donnée. Les objets que l'on considère comme bloqueurs ne sont alors plus les polygones mais les

[6] Avec une extension au cas 2D1/2.

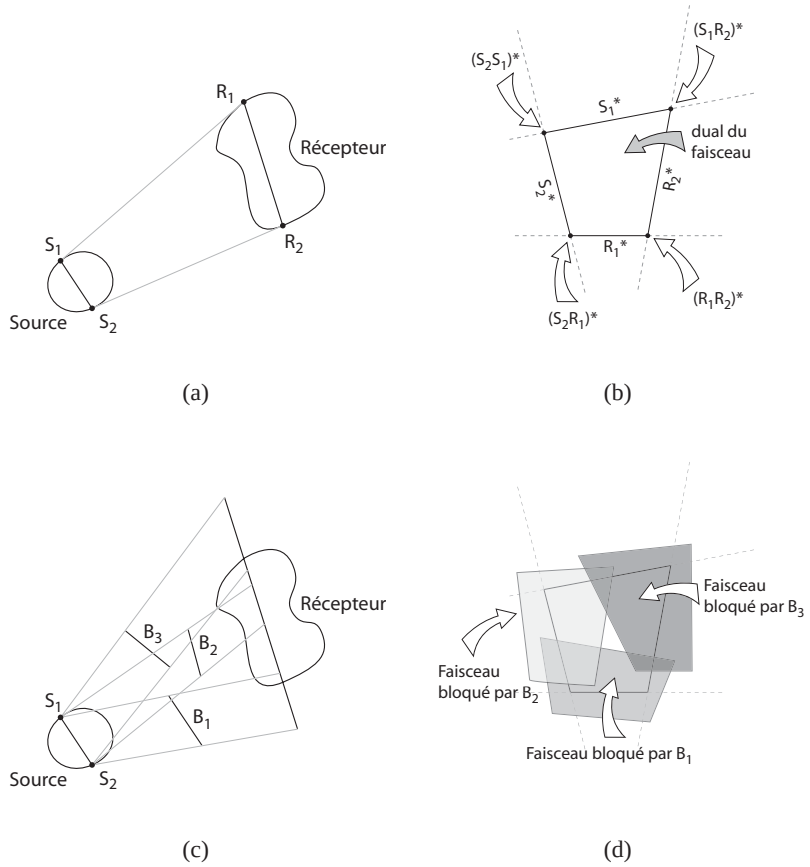


FIG. 1.4 – Visibilité dans l'espace des droites

Le faisceau des rayons (SR) *joignant la source et le récepteur (a) est représenté dans un espace dual. Un point a pour dual une droite, un rayon a pour dual un point et le faisceau a pour dual un polygone (b). Déterminer si la source est cachée par un ensemble de bloqueurs (c) revient par dualité à déterminer si les polygones duaux des faisceaux intersectés par chacun des bloqueurs recouvrent le polygone dual du faisceau entre la source et le récepteur (d). Le test de recouvrement peut être effectué de manière conservative en rasterisant les polygones à l'aide du matériel graphique.*

voxels intérieurs au modèle, que l'on dit *opaques*. Pour une cellule donnée, on classifie chaque voxel de la discrétisation comme *caché* ou comme *potentiellement visible*. Pour cela, l'algorithme fusionne les voxels opaques voisins pour obtenir des bloqueurs plus grands tout en maintenant une contrainte sur la forme de ces bloqueurs afin de faciliter le calcul de leurs cônes d'ombre. Les voxels entièrement contenus dans un de ces cônes peuvent alors être classifiés comme cachés.

L'efficacité de cet algorithme vient de l'observation que l'on peut considérer comme opaques les voxels qui ont été classifiés cachés. Intuitivement, la partie

d'une scène éclairée par une source de lumière ne change pas si l'on rajoute des objets dans l'ombre de cette source. Lors de la fusion des bloqueurs, on peut donc étendre les voxels opaques à travers les voxels cachés. Résumons les grandes lignes de l'algorithme :

Discrétisation de la scène. Le modèle polygonal est rasterisé dans l'espace discrétisé. Les voxels intersectés par les polygones sont dits *frontières*. Par un algorithme de remplissage, on peut alors déterminer les voxels qui sont entièrement à l'intérieur d'un objet et que l'on pourra donc considérer comme opaques [7].

Extension des bloqueurs. On cherche un voxel opaque non encore classifié comme caché. On étend ce voxel à travers les voxels voisins opaques. Le but est d'obtenir un pseudo-bloqueur parallèpipédique de taille maximale.

Construction du cône d'ombre. On détermine la partie de l'espace cachée par ce bloqueur vu de la cellule. Grâce à la forme simple du bloqueur, cette région a une forme simple, qui peut être déterminée efficacement avec des tables.

Recherche des parties cachées. Les voxels entièrement contenus dans le cône d'ombre sont classifiés comme cachés.

Tant qu'un voxel opaque non caché est trouvé, on recommence à l'étape 2. Le lecteur se reportera à l'article donné en annexe pour plus de détails.

L'efficacité de l'algorithme est augmentée par l'utilisation d'une discrétisation hiérarchique de l'espace. Elle permet de réduire le coût mémoire par rapport à l'utilisation d'une simple grille régulière. Elle permet d'autre part des requêtes hiérarchiques de visibilité. Lorsque le cône d'ombre d'un bloqueur est calculé, on teste si les noeuds racines de l'arbre sont entièrement, partiellement ou pas du tout inclus dedans. Tant qu'un noeud est partiellement inclus, on teste ses fils. Dès qu'un noeud est entièrement ou pas du tout inclus dans le cône, on peut directement marquer comme cachés ou visibles tout ses fils. Schaufler *et al* utilisent dans leur papier une structure d'*octree*.

1.3.1 Avantages

L'algorithme proposé par Schaufler présente plusieurs avantages. Tout d'abord il travaille sur une représentation volumique de la scène. Le modèle est donc d'une certaine manière épuré. Aucune propriété forte n'est requise sur la scène. On n'impose notamment aucune condition topologique, comme le fait d'avoir des objets qui sont des variétés, ni aucune organisation hiérarchique au niveau de la description de la scène (celle-ci est reconstruite lors de la discrétisation). La seule contrainte sur le modèle est qu'il doit représenter un volume. C'est-à-dire par exemple qu'on ne pourra l'utiliser sur un modèle d'arbre où chaque feuille est représentée par un seul polygone. Concrètement le modèle doit donc avoir un extérieur et un intérieur clairement délimités. On dit aussi qu'il doit être *étanche*, par

[7] On suppose que la scène ne contient aucun polygone transparent, et que les objets sont fermés.

référence à l'algorithme de remplissage utilisé pour déterminer l'intérieur et l'extérieur. Cette contrainte est une contrainte relativement faible car on n'impose que *l'existence* de cette séparation, et non sa spécification précise dans le modèle. Par exemple, le gratte-ciel simpliste de la Figure 1.5 est spécifié par un maillage qui n'a aucune bonne propriété topologique : il y a des arêtes qui ont plus de deux faces adjacentes et des faces s'intersectent. Mais il satisfait pourtant bien la contrainte d'étanchéité. La plupart des objets que l'on considère étant des modèles d'objets réels, ils sont par essence volumiques et sont dans le même cas que le gratte-ciel : étanche mais avec un maillage biscornu.

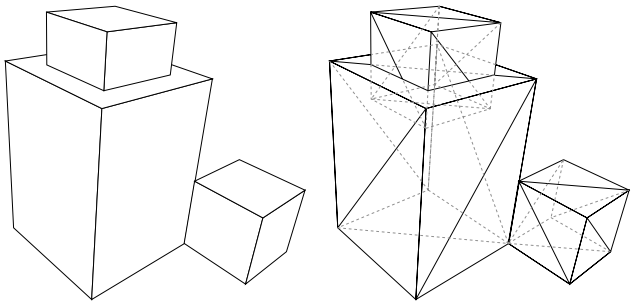


FIG. 1.5 – Un exemple de maillage dégénéré mais étanche

La représentation par voxel permet ensuite de travailler avec des bloqueurs convexes pour lesquels les calculs de visibilité sont simples. En effet, le cône d'ombre d'un tel bloqueur est convexe et pour tester l'inclusion d'un objet dans ce cône, il suffit de tester l'inclusion de son enveloppe convexe et donc de n'importe quel volume englobant. Cela permet notamment un calcul hiérarchique. En outre, la forme simple des bloqueurs et le fait qu'ils soient alignés sur les axes permettent des calculs efficaces et robustes. Enfin la discrétisation de la scène et donc des bloqueurs permet une fusion explicite des ces derniers. Cette fusion est obtenue par l'extension d'un bloqueur élémentaire (voxel opaque) à travers les voxels opaques ou cachés voisins, pour obtenir un bloqueur de taille maximale. C'est cette « extension à travers les zones d'ombres » qui confère à la méthode toute son efficacité.

Les avantages de la méthode viennent donc de la discrétisation de la scène, qui extrait de la description polygonale l'information nécessaire au calcul de visibilité, c'est-à-dire l'intérieur des objets, et la représente d'une façon adaptée à des calculs efficaces (hiérarchie de voxels alignés avec les axes). On n'utilise donc pas le modèle initial pour le calcul de la visibilité mais plutôt une représentation de son intérieur qui soit équivalente du point de vue des occlusions engendrées. Cette idée se retrouve dans les calculs d'horizons proposés par Downs *et al* [DMS01] qui utilisent des *Convex Vertical Prisms* (CVP) pour approximer les bloqueurs (voir Figure 1.6) et pouvoir ainsi évaluer dynamiquement la silhouette des objets. Et c'est précisément ce point que nous reprendrons dans notre approche. Pour calculer ce que cache un objet, on ne se préocuppe pas des tout petits détails à sa surface. Ils

fragilisent et ralentissent les calculs alors qu'ils contribuent de manière négligeable aux occlusions.

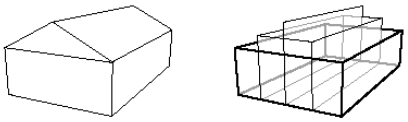


FIG. 1.6 – Convex Vertical Prisms

1.3.2 Limitations

La première des limitations de cette méthode est qu'elle ne réalise pas complètement la fusion des pénombres. Si l'on reprend l'analyse préliminaire de la Section 1.2, on voit que l'étape analytique utilise l'expression simple disponible pour une source, un récepteur et un bloqueur convexe. Comme on l'a dit a propos de travaux de Cohen-or, cette méthode ne donne des bons résultats que si l'étape algorithmique a sélectionné un bloqueur convexe de grande taille. La contribution de l'algorithme est donc l'heuristique de regroupement des bloqueurs élémentaires (les voxels) en larges bloqueurs convexes, grâce à l'algorithme d'extension à travers les zones d'ombres. Cette heuristique peut être décrite de manière équivalente de la façon suivante : on calcule l'ensemble des cônes d'ombres des bloqueurs élémentaires, on réalise l'union de ces cônes d'ombres en les discrétisant, on décompose cette union en composantes convexes maximales, on calcule le cône d'ombre de ces nouveaux bloqueurs. Cette formulation est équivalente en raison de l'ordre dans lequel le parcours des voxels et l'extension sont réalisés. L'algorithme parcourt les voxels dans l'ordre de distance croissante à la source. Un autre ordre de parcours donnerait un résultat moins optimal. Regardons par exemple la Figure 1.7. Le voxel *A* est sélectionné en premier, étendu à ses voisins et le cône d'ombre résultant est calculé *(a)*. Les voxels dans le cône sont marqués cachés *(b)*. On réitère avec le bloqueur *B (c & d)* puis avec le bloqueur *C (e & f)*. Tous les voxels opaques ont été visités. L'ombre résultante est faible, en comparaison du résultat exact (qui place toute la partie supérieure dans l'ombre).

Si par contre, comme sur la Figure 1.8, on sélectionne d'abord le voxel *E (a)* puis le voxel *F* on obtient un bien meilleur résultat *(c)*. En effet, le voxel *E* peut être étendu à travers les voxels cachés par *F* et atteindre le bloqueur *A*. Parce que l'on ne considère que les voxels opaques comme bloqueurs potentiels à étendre, l'ordre de parcours est important. Si l'on considèrait aussi les voxels précédemment marqués cachés, l'ordre n'aurait plus d'importance. Par contre, le nombre de voxels à visiter augmenterait grandement et les temps de calculs croîtraient considérablement.

En reformulant ainsi l'heuristique de sélection, il est facile d'exhiber des configurations où la méthode proposée ne réalise pas la fusion des pénombres. La Figure 1.9 en propose deux. Dans l'exemple de gauche, les cônes d'ombres sont

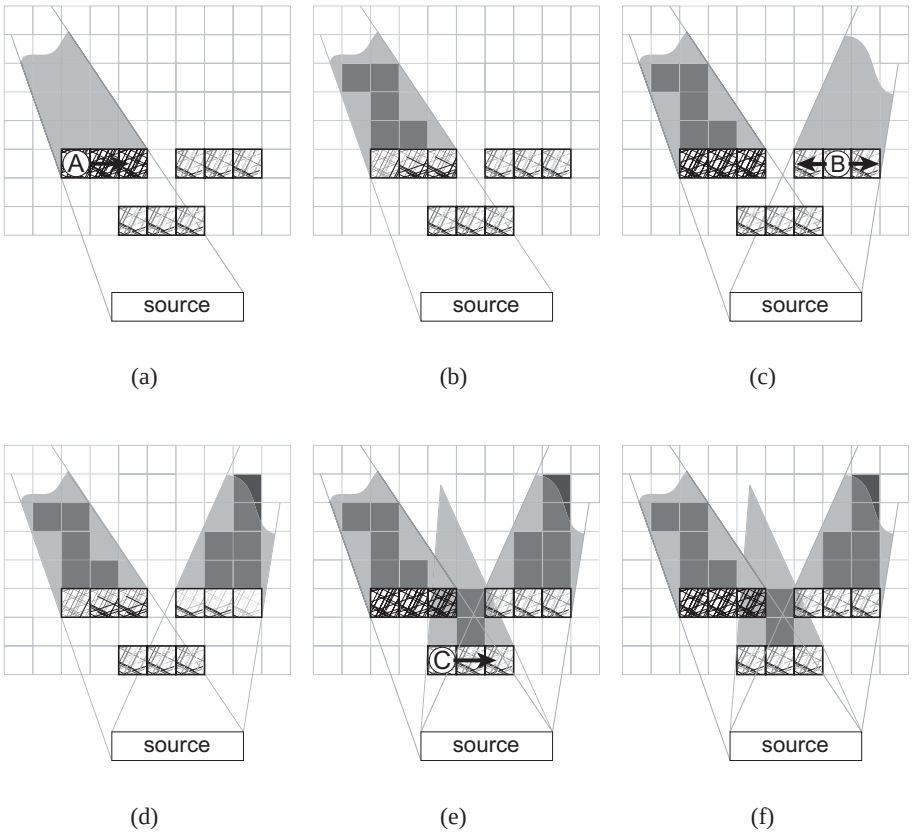


FIG. 1.7 – Importance de l'ordre de parcours

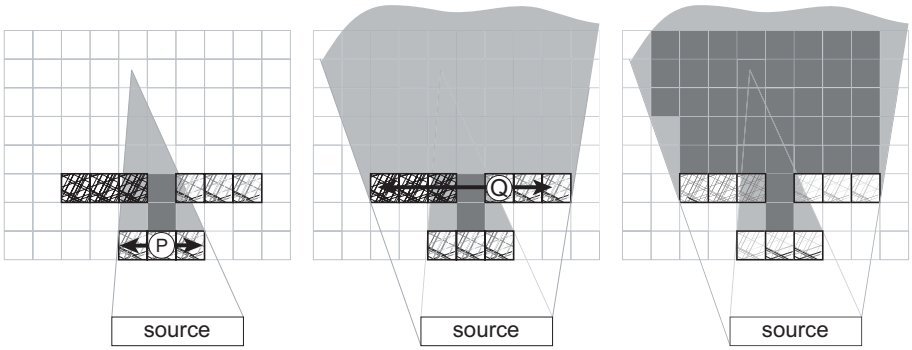


FIG. 1.8 – Importance de l'ordre de parcours (suite)

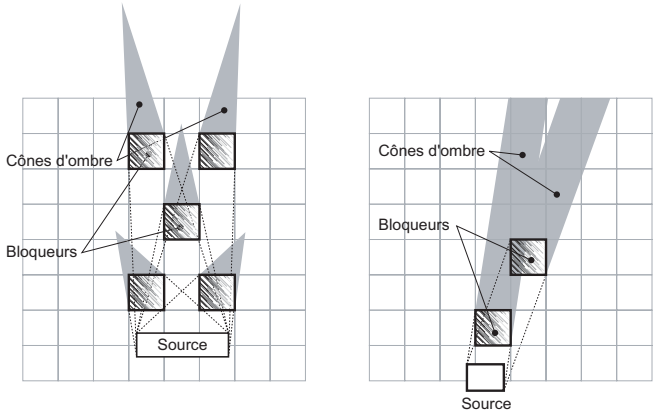


FIG. 1.9 – Fusion incomplète des pénombres

Acun voxel ne tombre entièrement dans les cônes (en gris plein), et donc aucun voxel ne sera marqué caché.

disjoints. Lorsque l'on réalise leur union, puis la décomposition en composantes convexes maximales, on retombe donc sur les bloqueurs initiaux.

L'exemple de droite met en évidence un autre problème de l'approche proposée. Dans cet exemple, les cônes d'ombre s'intersectent et pourtant, leur union n'est pas réalisée car les *discrétisations* des cônes ne s'intersectent pas. Cela vient du fait que la structure de voxels discrétise les bordures des objets et les limites d'ombres avec la même résolution.

En effet, on a dit que l'efficacité de la méthode reposait sur la discrétisation de l'espace, qui sert deux buts : représenter de façon simple et efficace d'une part l'intérieur des objets pour calculer leurs cônes d'ombre, et d'autre part les cônes d'ombre eux-mêmes pour calculer leur union et la décomposer en composantes convexes. Il y a donc en réalité deux discrétisations à prendre en compte : celle de la scène, et celle des limites d'ombre. Or ces deux discrétisations doivent être confondues pour pouvoir réaliser l'extension des bloqueurs à travers les zones d'ombre. Malheureusement, si la première discrétisation ne dépend pas de la cellule, la deuxième devrait être recalculée pour chaque cellule. Si, comme dans l'article de Schaufler, on ne fait pas ce recalcul, on risque de rater, à cause de la discrétisation hiérarchique, de larges zones d'ombre comme le montre l'exemple de la Figure 1.10. Sur cette figure, on voit un bloqueur et l'octree correspondant à sa discrétisation. La cellule 1 par exemple n'a pas été subdivisée. Deux voxels sont à l'intérieur de l'objet et sont donc considérés opaques. Le cône d'ombre entre la cellule et ces voxels opaques couvre presque toute la cellule 1, plus de 50% de la cellule 2 et presque $1/16^e$ de la cellule 3. Et pourtant aucun voxel ne sera marqué comme caché par l'objet. Ceci est du au fait que la subdivision de l'octree corres-

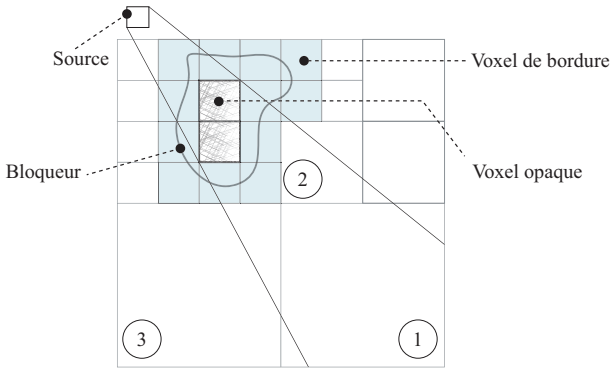
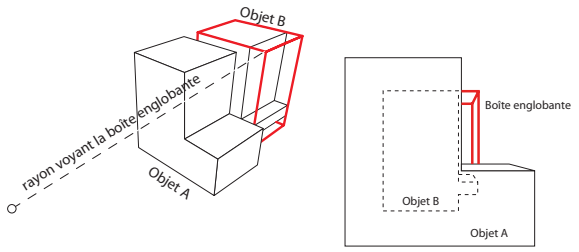


FIG. 1.10 – Perte d'ombre

pond à l'objet et non au limites d'ombre. Lorsque, comme suggéré dans l'article original, on cherche à déterminer la visibilité d'objets qui n'ont pas été insérés dans la discrétisation (parce qu'ils n'avaient par exemple que peu de chance de causer des occlusions), d'objets en mouvement ou d'objets ajoutés à la scène, on sera confronté au problème.

Au passage on notera que lorsqu'un objet dont on teste la visibilité a été inséré dans la subdivision hiérarchique, on peut faire mieux que tester sa boîte englobante, comme il est fait dans l'article. On peut à la place tester les voxels qui échantillonnent cet objet (voxels de bordure). On peut ainsi obtenir une sur-estimation du pourcentage des objets qui sont visibles. Une telle information peut être utilisée par exemple pour décider quels objets de la scène ne seront pas rendus (bien que potentiellement visibles) dans une approche orientée budget. Une telle approche garantit le temps réel en sélectionnant une partie de la scène qui peut-être rendue dans le temps alloué (budget) à la génération d'une image. L'algorithme de sélection essaye d'optimiser la qualité de l'image obtenue [FST92, MS95]. Ainsi, en sachant à quel point chaque objet est visible, on pourra choisir parmi plusieurs objets lesquels sacrifier si on ne peut les rendre tous. En outre, les voxels de bordure représentent un volume englobant des objets plus précis qu'une simple boîte englobante et permettent en conséquence de détecter plus d'occlusions. La figure 1.32 montre comment des petits détails sur l'objet (ici des balcons sur des bâtiments) influencent grandement la boîte englobante et la rendent visible alors que l'objet est caché.

La seconde et non négligeable limitation dont souffre la méthode est la nécessité de discrétiser toute la scène. Pour pouvoir trouver l'intérieur des objets, cette discrétisation doit avoir un grain suffisamment petit. Si la scène est très étendue spatialement, cela n'est pas possible. La Figure 1.12 montre en effet que pour un modèle un peu étendu, il faut subdiviser l'octree jusqu'à une profondeur de l'ordre de 8. Si la subdivision est complète, ce qui est pratiquement le cas dans l'exemple



(a) vue de dessus (b) vue de O

FIG. 1.11 – Variations d'une boîte englobante avec les petits détails

de la ville, le nombre de noeuds [8] d'un octree de profondeur n est $\frac{1}{7}(8^{n+1}-1)$. Dans notre implémentation, le nombre d'octets par noeud est de 90, la place mémoire requise est alors de 1,7Go.

On peut envisager une modification de l'algorithme qui utilise une discrétisation grossière de la scène. Pour chaque cellule, on raffine successivement des groupes de voxels et on utilise cette sous-discrétisation plus fine pour le calcul de visibilité. Ce faisant, on ne réalise alors plus la fusion des pénombres entre voxels des groupes successifs.

La solution proposée par Schaufler *et al* pour un modèle comme celui de la ville est une version 2D et demi de leur algorithme. On n'utilise non plus un octree, mais un quadtree dont les noeuds stockent une hauteur. On réduit ainsi le coût mémoire et on peut traiter des modèles plus grands. Le problème demeure cependant et il suffit d'augmenter la taille du modèle pour atteindre la limite, quelle qu'elle soit. En outre, cela impose de développer deux algorithmes (3D et 2D1/2) et de demander à l'utilisateur d'identifier le type de modèle et de choisir l'algorithme correspondant. Enfin, si les villes sont en première approximation effectivement 2D et demi, il existe cependant nombre de parties tri-dimensionnelles : pont, arches, galleries et traboules. Nous verrons plus loin que la méthode que nous proposons permet de travailler indifféremment sur des modèles 3D où 2D1/2, et également sur des modèles de très grande taille. L'algorithme travaille en effet objet par objet et n'a pas besoin de considérer la scène dans son ensemble.

Signalons enfin un point délicat de l'algorithme de [SDDS00]. Il s'agit d'un détail de l'étape d'extension des bloqueurs qui a son importance pour l'efficacité du calcul. Pour trouver des bloqueurs convexes de taille maximale, ce qui est le but de l'extension, il faut étendre dans les trois directions de l'espace. L'ordre dans lesquels ces directions sont parcourues, et les aspérités de la voxelisation peuvent gréver sérieusement l'extension, car en étendant au maximum dans une

[8] Le nombre de feuilles d'un octree de niveau n est 8^n comme on peut le voir trivialement par récurrence [Han89]. Le nombre de noeuds est la somme du nombre de feuilles à tous les niveaux, c'est-à-dire la somme des termes d'une suite géométrique de raison 8.

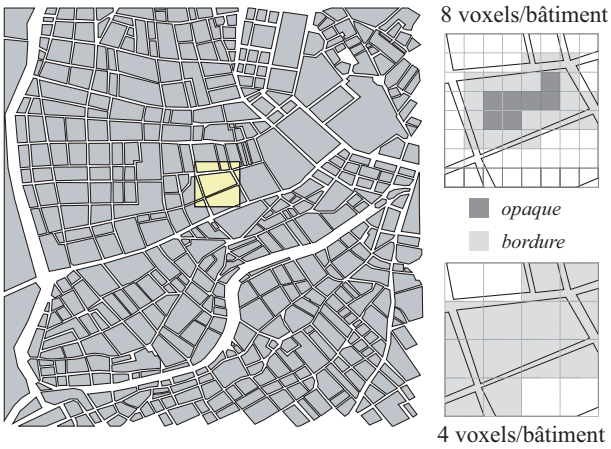


FIG. 1.12 – Modèle étendu

Dans le modèle de Vienne ci-dessus (reproduit avec la permission de Peter Wonka), la scène fait 500 unités de large. Les batîments font environ 20 unités de large. Comme on peut le voir sur la droite, un bâtiment doit être approximativement couvert par 8 voxels pour pouvoir échantillonner correctement son intérieur. Il faut donc au moins $8 \times \frac{500}{20} = 200$ *voxels, c'est-à-dire un octree subdivisé jusqu'au* 8e *niveau ce qui représente une place mémoire non négligeable (environ 1,7Go).*

direction on peut tomber sur un maximum local qui est sous-optimal globalement. Pour sortir de ces extrema locaux les auteurs proposent une heuristique d'extensions/réductions successives, dont l'efficacité n'est pas facilement évaluable. Si les auteurs constatent qu'elle marche « plutôt bien en pratique », il est assez difficile d'imaginer tout les cas particuliers et la façon d'interagir avec l'extension à travers les zones d'ombres. Il n'est donc pas évident d'évaluer la sur-conservativité de l'algorithme, c'est-à-dire les occlusions non détectées, et la cause de cette surestimation (heuristique d'extension, taille des voxels, etc...).

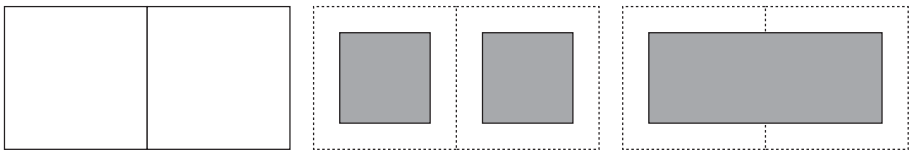
1.4 La réduction de bloqueurs

Peter Wonka et Michaël Wimmer ont récemment proposé [WWS00] une méthode de calcul de visibilité très intéressante. La source est échantillonnée par des points. En chacun de ces points, l'ombre des objets de la scène est calculée. L'union de ces ombres approxime inexactement l'ombre de la source. En effet, il se peut que des objets invisibles depuis les points échantillons soient visibles pour des positions situées entre les échantillons (imaginons par exemple une avenue bordée de rues transversales étroites et des échantillons qui ne soient pas dans l'axe de ces rues). Pour obtenir un calcul conservatif, les auteurs proposent de *réduire* les bloqueurs. Intuitivement, la réduction correspond à un « amincissement » des objets. Plus formellement, il s'agit d'une opération de morphologie mathématique appellé

érosion. L'ombre calculée en un point échantillon est alors plus petite mais possède une propriété particulière très utile : un objet qui est caché en un point par l'objet réduit, est caché par l'objet non réduit au voisinage de ce point. Le rayon de ce voisinage est le rayon dont l'objet a été réduit. Cette propriété est relativement intuitive. Parker *et al* [PMS+99] l'utilisent pour réduire les objets et simuler le rendu d'ombres douces dans le cas d'une source lumineuse sphérique. Wonka et Wimmer en donnent une démonstration formelle. Nous verrons plus loin qu'il est possible de démontrer un résultat plus fort que nous exploiterons dans notre algorithme. Pour l'instant, remarquons que cette propriété offre plusieurs avantages. Tout d'abord, elle permet de ramener le calcul de la visibilité pour une source étendue à un calcul pour une source ponctuelle, beaucoup plus facile. On peut alors utiliser différentes approches, la plus simple étant une utilisation directe du *z-buffer* comme nous le décrirons plus loin. Les auteurs choisissent d'utiliser une méthode de carte d'occlusion qu'ils ont préalablement développée [WS99]. La réduction de bloqueur permet en outre de traiter directement les artefacts de rasterisation à prendre en compte. Le second avantage de cette méthode est qu'elle permet une fusion générale des pénombres et que, comme le font remarquer les auteurs, elle discrétise les limites d'ombres *dans l'espace image*, réalisant ainsi *implicitement* la fusion des bloqueurs.

Cependant, l'impact du choix de la taille des cellules sur la visibilité calculée n'est pas clair. Si cette taille est trop grande, les bloqueurs sont réduits à l'ensemble vide, et aucune occultation n'est détectée. À l'inverse, si elle est trop petite, le nombre de cellules nécessaires pour recouvrir l'ensemble des points de vues possibles devient trop grand.

Mais la principale difficulté de cette approche réside dans le calcul de la réduction d'un bloqueur. En se limitant au cas 2D, et en utilisant des hypothèses fortes sur le modèle (connaissance que les polygones correspondent à des murs), les auteurs montrent l'efficacité de l'approche. Cependant, pour un modèle 3D, ce n'est pas aussi simple. En effet, la réduction est une propriété fondamentalement *volumique* : il faut réduire dans toutes les directions de l'espace. Or la primitive de modélisation est le triangle, qui a une épaisseur nulle, et donc une réduction égale à l'ensemble vide. Dans un modèle polygonal, l'information de volume doit d'abord être extraite pour pouvoir déterminer comment réduire chaque polygone. C'est-à-dire déterminer de quel côté du triangle il y a de la matière, et sur quelle distance. À supposer que cette information soit reconstructible, en considérant par exemple une variété (c'est-à-dire un modèle avec un intérieur et un extérieur définis), la réduction n'est pas pour autant simple à calculer. En outre, il faut être capable de regrouper certains volumes. Considérons par exemple deux cubes adjacents. S'ils sont réduits indépendamment, les occlusions qu'ils permettront de détecter seront clairement moindres que s'ils sont réduits « ensemble » comme le montre la Figure 1.13. Pour s'affranchir de ces calculs délicats, nous avons eu l'idée de voxeliser le modèle et de calculer la réduction sur un ensemble de voxels.



(a) 2 cubes (b) réduits séparément (c) réduits ensemble

FIG. 1.13 – Réduction de deux cubes adjacents

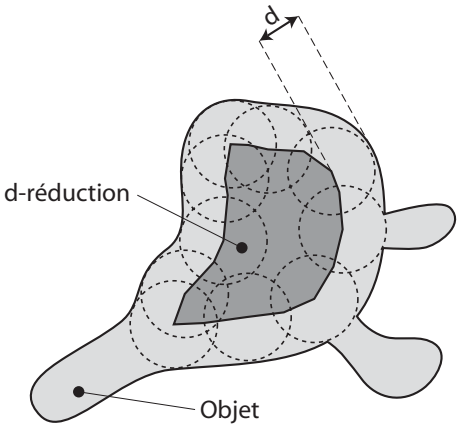


FIG. 1.14 – d-réduction d'un objet

1.5 Réduction volumique de bloqueurs voxelisés

Dans cette section, nous reprenons tout d'abord la démonstration du principe de réduction des bloqueurs et montrons un résultat supplémentaire. Nous présentons ensuite un algorithme permettant de calculer de manière simple, efficace et robuste une approximation de la réduction d'un objet. La dernière partie expose comment ces résultats sont regroupés, et présente l'algorithme final de précalcul de la visibilité.

1.5.1 Réduction de bloqueurs et de récepteurs

On définit la réduction d'un objet comme l'ensemble des points *intérieurs* à l'objet et situés à une distance d donnée de la surface de l'objet. La distance d est appelée *diamètre de réduction*. La Figure 1.14 montre un exemple de réduction. Formellement, on définit la d-réduction d'un objet O par :

$$O_d = \{P \in \mathbb{R}^3 \,/\, \mathcal{B}_d(P) \subset O\} \tag{1.1}$$

où $\mathcal{B}_d(P)$ est la boule de rayon d centrée sur P. La réduction est assez trivialement unique. Considérons en effet deux ensembles $\mathcal{O}_1$ et $\mathcal{O}_2$ qui satisfont l'équation (1.1). Quelque soit $P \in \mathcal{O}_1$, on a $\mathcal{B}_d(P) \subset \mathcal{O}_1$. On peut donc trouver $Q \in \mathcal{O}$ tel que $|\overrightarrow{PQ}| < d$. Or pour ce Q on a $\mathcal{B}_d(Q) \subset \mathcal{O}_2$, d'où $P \in \mathcal{O}_2$ et $\mathcal{O}_1 \subset \mathcal{O}_2$. Par symmétrie on a alors $\mathcal{O}_1 = \mathcal{O}_2$.

Avec cette définition, Wonka et Wimmer montrent le théorème suivant, dont nous reproduisons la preuve donnée en annexe de [WWS00] :

Théorème 1 (Réduction de bloqueurs) *Tout point P qui est caché par $\mathcal{O}_d$ vu d'un point V est aussi caché par $\mathcal{O}$ s'il est vu d'un point V' tel que $|\overrightarrow{VV'}| \leq d$.*

►**preuve:** Supposons qu'il y ait un point V' qui ne soit pas caché. Alors tout point le long du segment $[V'P]$ n'est pas contenu dans $\mathcal{O}$. Comme P est caché, il y a au moins un point $Q \in \mathcal{O}_d$ sur le segment $[VP]$. $VV'P$ forme un triangle et $|\overrightarrow{VV'}| \leq d$, donc il doit y avoir un point Q' sur le segment $[V'P]$ avec $|\overrightarrow{QQ'}| \leq d$. Par définition, tout les points dans un d-voisinage de Q sont dans $\mathcal{O}$ donc Q' est dans $\mathcal{O}$ et P est caché. CQFD. ◄

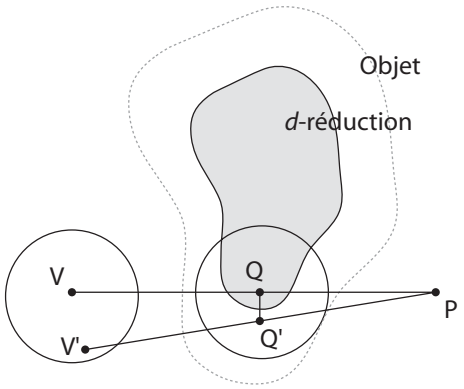


FIG. 1.15 – Réduction de bloqueur, démonstration

La définition (1.1) de la réduction que nous avons donnée utilise une boule pour reprendre la définition utilisée par Wonka et Wimmer. Mais on pourrait utiliser n'importe quel ensemble. La réduction correspond en morphologie mathématique à une érosion d'élément structurant cette boule [SM93]. On peut écrire :

$$\mathcal{O}_d = \{P \,/\, \forall \mathbf{x} \in \mathbf{B}_d \;\; P + \mathbf{x} \in \mathcal{O}\}$$

Wonka et Wimmer utilise comme élément structurant une boule, mais nous allons généraliser leurs résultats aux élément structurants convexes. Nous allons en plus montrer qu'on peut réduire non seulement le bloqueur mais aussi l'objet caché. C'est ce qu'exprime le théorème fondamental 2. Avant de donner son énoncé, définissons quelques notations utilisées.

On définit la *dilatation* d'un ensemble de points $\mathcal{O}$ et d'un ensemble de vecteurs $\mathbf{X}$, aussi connue comme la somme de Minkowski [Min03] des deux ensembles par :

$$\mathcal{O} \oplus \mathbf{X} = \{P + \mathbf{x} \,|\, P \in \mathcal{O} \text{ et } \mathbf{x} \in \mathbf{P}\}$$

En morphologie mathématique, $\mathbf{X}$ est appelé *élément structurant*. On définit également la somme de Minkowski de deux ensembles de vecteurs :

$$\mathbf{X} \oplus \mathbf{Y} = \{\mathbf{x} + \mathbf{y},\ \mathbf{x} \in \mathbf{X} \text{ et } \mathbf{y} \in \mathbf{Y}\} \tag{1.2}$$

Enfin, on définit *l'érosion*, opération symétrique de la dilatation, par

$$\mathcal{O} \ominus \mathbf{X} = \{P \,/\, \forall \mathbf{x} \in \mathbf{X} \ \ P + \mathbf{x} \in \mathcal{O}\} \tag{1.3}$$

On utilise la notation $\ominus$ pour rappeller le $\oplus$ de la dilatation mais on prendra garde qu'il ne s'agit pas de la différence de Minkowski qui est définie par $\mathcal{O} \oplus (-\mathbf{X})$. Ces deux quantités sont différentes comme le montre la figure 1.16. L'érosion est en fait la complémentaire de la dilatation par le complémentaire, ce que l'on peut écrire :

$$\mathcal{O} \ominus \mathbf{X} = (\mathcal{O}^C \oplus \mathbf{X})^C \tag{1.4}$$

Minkowski-vs-erosion.eps.eps

FIG. 1.16 – Différence entre érosion et différence de Minkowski

Quelques propriétés se montrent immédiatemment : $\oplus$ est commutatif et distributif sur l'union. Par complémentarité, l'opérateur $\ominus$ est distributif sur l'intersection. Par contre $\ominus$ n'est pas distributif sur l'union ce qui traduit l'observation que nous avons faite que l'on ne peut pas éroder un objet en érodant indépendamment ses parties (voir figure 1.13). Grâce à ces notations nous montrerons plusieurs autres résultats intéressants et notamment le théorème fondamental suivant :

Théorème 2 (Réduction bloqueurs et bloqués) *Si un segment* $[SR]$ *intersecte* $\mathcal{O} \ominus \mathbf{X}$*, où* $\mathbf{X}$ *un ensemble vectoriel convexe, alors tout segment* $[S'R']$ *avec* $S' \in \{S\} \oplus \mathbf{X}, R' \in \{R\} \oplus \mathbf{X}$ *intersecte* $\mathcal{O}$*.*

►**preuve:** Soit $S' \in \{S\} \oplus \mathbf{X}$ et $R' \in \{R\} \oplus \mathbf{X}$. Par définition, il existe $\mathbf{x}_S \in \mathbf{X}$ et $\mathbf{x}_R \in \mathbf{X}$ tels que :

$$\begin{aligned} S' &= S + \mathbf{x}_S \\ R' &= R + \mathbf{x}_R \end{aligned}$$

Par hypothèse, il existe un point Q sur le segment $[SR]$ qui soit dans $\mathcal{O} \ominus \mathbf{X}$. Pour ce point, il existe $t \in [0, 1]$ tel que :

$$Q = tS + (1 - t)R$$

Soit $Q' = tS' + (1-t)R'$. C'est un point de $[S'R']$, et on a :

$$\begin{aligned} Q' &= t(S + \mathbf{x}_S) + (1-t)(R + \mathbf{x}_R) \\ &= tS + (1-t)R + t\mathbf{x}_S + (1-t)\mathbf{x}_R \\ &= Q + \mathbf{x} \end{aligned}$$

avec $\mathbf{x} = t\mathbf{x}_S + (1-t)\mathbf{x}_R$. Comme $\mathbf{X}$ est convexe, on a clairement $\mathbf{x} \in \mathbf{X}$. Comme $Q \in O \ominus \mathbf{X}$, on a $Q + \mathbf{x} \in O$. Donc le segment $[S'R']$ intersecte O en Q', ce qu'il fallait démontrer. ◀

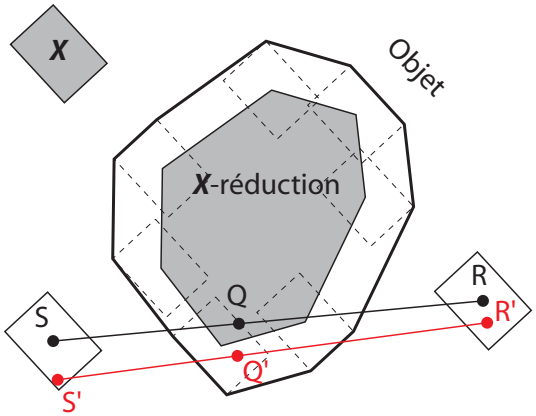


FIG. 1.17 – Réduction bloqueur et bloqués, démonstration

Cette propriété de la **X**-réduction est plus forte et plus générale que celle démontrés par Wonka *et al*. Le corollaire suivant exprime la conséquence essentielle de ce résultat :

Corollaire 1 *Si pour un point de vue V donné, la* **X***-réduction d'un objet est caché par la ou les* **X***-réductions d'un ou plusieurs bloqueurs, alors l'objet est caché par ces même bloqueurs pour tout point de vue dans une région de forme* **X** *autour de V.*

Autrement dit, on peut réduire non seulement les bloqueurs mais aussi les objets bloqués. Pour calculer la visibilité, on dispose alors de l'approche globale suivante :

– fabriquer un masque d'occlusion avec les objets réduits ;
– tester par rapport à cette carte d'occlusion si les objets *réduits* sont visibles.

L'intérêt de cette approche est que pour calculer le masque d'occlusion et effectuer le test par rapport à cette carte, on peut utiliser n'importe lequel des nombreux algorithmes existants pour le calcul de la visibilité en un point. Le résultat se trouve étendu à une région autour du point en question, la forme de la région étant définie par l'élément structurant **X** utilisé pour la réduction. Une approche très simple est par exemple décrite à la section suivante.

Exemple d'utilisation de la propriété de réduction

La façon la plus simple est d'utiliser le *z-buffer*. La carte d'occlusion est obtenue en effectuant un rendu OpenGL et en récupérant le *z-buffer*. On teste alors la visibilité d'un objet de la manière suivante :

- on rasterise l'objet réduit dans la carte ;
- si un pixel au moins passe le test en z, on conclue que l'objet est visible, sinon qu'il est caché.

Si on est capable de calculer *exactement* la **X**-réduction, l'algorithme ci-dessus est encore plus simple à mettre en œuvre. En effet, pour à la fois calculer le masque d'occlusion et tester les objets, il suffit de rendre une vue des objets réduits en fausses couleurs (chaque couleur identifie uniquement un objet) pour le point de vue de référence. L'image obtenue est courrament appelée *item buffer*. On parcourt ensuite l'image pour retrouver les couleurs présentes : ce sont celles des objets visibles. Un objet dont la couleur identifiante n'est pas présente dans l'image peut être classifié comme caché. En effet, puisqu'aucun pixel de sa réduction n'est visible dans l'image, cela veut dire que tout rayon joignant la réduction et le point de référence est bloqué par la réduction d'un autre objet[9]. Et donc en vertu du corollaire, l'objet est caché pour tout point de vue dans la cellule **X**.

Comme une seule vue ne peut couvrir toutes les directions d'observation autour du point de vue, il faut en prendre plusieurs, par exemple six couvrant les six faces d'un cube. Ces vues peuvent être rendues côte à côte dans une seule image, sur deux lignes et trois colonnes (Figure 1.18). Cela permet de lire un seule fois le

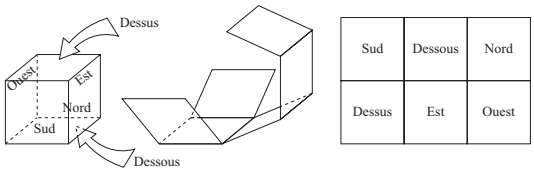


FIG. 1.18 – Rendu des six faces d'un cube dans une seule vue

frame buffer, ce qui est une opération coûteuse.

L'intérêt de cet algorithme est que l'on effectue en rendu en un point et que l'on obtient un résultat valable pour une région. Or le rendu en un point est quelque chose de bien connu en informatique graphique et de nombreux procédés d'accélération existent, qui peuvent être utilisés pour effectuer ce rendu. D'ailleurs, une façon naïve et non conservative, mais pourtant parfois utilisée en raison de sa simplicité de mise en œuvre, de calculer la visibilité consiste à utiliser le procédé de fausses couleurs décrit ci-dessus en prenant des images des objets en des points échantillons. La Figure 1.19 montre justement deux rendus en fausses couleurs d'un point *(c)*, l'un effectué sans réduction (rangée du dessus), et l'autre effectué avec une réduction (rangée du bas) correspondant à une cellule sphérique (en

[9] À la résolution de l'image donnée, mais la réduction permet aussi de s'affranchir de ce point.

jaune au centre de l'image *(b)*). Les bâtiments visibles correspondant à ces vues sont montrés en vert *(a)*. On voit que la réduction permet de déterminer que les bâtiments dans les rues perpendiculaires et « en diagonale » sont potentiellement visibles si on se déplace du centre de la cellule, d'où a été calculée la visibilité, en autre point de la cellule. Ce qui n'est évidemment pas détecté par la méthode naïve sans réduction.

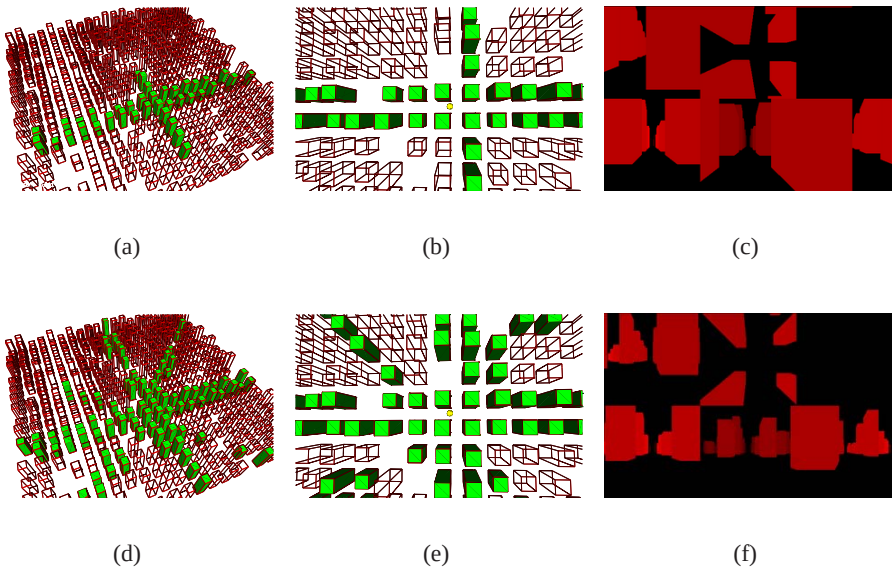


FIG. 1.19 – Calcul simple *versus* calcul conservatif de la visibilité

Pour que l'algorithme en fausse couleur basée sur les réductions soit exact, il faut cependant faire attention aux objets dont la réduction est l'ensemble vide. En effet, qu'un tel objet soit potentiellement visible ou caché, il ne produira aucun pixel dans la vue en fausses couleurs. La solution que nous choississons pour traiter ce problème est la suivante : on considère simplement tous ces objets comme visibles. On est ainsi un peu trop conservatif, mais si trop d'objets ont une réduction vide, c'est de toute façon que la taille de l'élément structurant est mal adaptée. Une autre solution utilisée par Durand *et al* [DDTP00] consiste à tester le pixel « au centre de l'objet ».

Nous avons implémenté l'algorithme décrit-ci dessus dans l'esprit d'*instant visibility* [WWS01]. Le modèle que nous avons considérer (voir Figure 1.19) est une ville de cubes alignés. C'est un modèle « théorique » mais notre but était de valider l'approche par réduction. En effet, comme nous allons le voir dans la section suivante, il faut pouvoir calculer la réduction *exacte* des objets de la scène pour que le théorème 2 puisse s'appliquer.

1.5.2 Réduction approximative

Le problème de la **X**-réduction est qu'elle n'est pas facile à calculer exactement dans le cas général. On peut par contre disposer d'approximations. On suppose que l'on dispose de deux approximations, l'une entièrement contenue dans la réduction, que l'on appelle *interne*, et l'autre contenant entièrement la réduction, que l'on appelle *externe*. On les note :

$$\underline{O_{\mathbf{X}}} \subset O_{\mathbf{X}} \subset \overline{O_{\mathbf{X}}}$$

On fera bien attention qu'une réduction externe d'un objet est définie comme contenant la réduction exacte, mais ne contient pas nécessairement l'objet lui- même.

Si l'on travaille avec ces réductions approximatives, l'algorithme décrit à la section précédente doit être modifié pour être conservatif. Il faut en effet clairement utiliser les réductions *internes* des objets pour calculer le masque d'occlusion, et utiliser les réductions *externes* pour tester leur visibilité. On ne peut alors plus effectuer ces deux calculs en une seule passe. La carte d'occlusion peut-être calculée comme précédemment en utilisant le matériel graphique, mais on ne peut plus utiliser les identifiants couleurs présents dans la carte d'occlusion pour tester si un objet est visible. La carte d'occlusion est donc générée en ignorant les couleurs (c'est-à-dire que l'on obtient simplement une carte de profondeur), et on effectue ensuite les tests par rapport à cette carte « à la main » en rasterisant chaque réduction externe dans la carte, sans la mettre à jour, pour déterminer si au moins un pixel passe le *z*-test.

En réalité, grâce aux nouvelles fonctionnalités des cartes graphiques, cette phase peut-être effectuée de manière simple et efficace. Le test contre une carte d'occlusion stockée dans le *z-buffer* est en effect disponible sous forme de requête d'occlusion sur les dernières cartes graphiques (ATI, HP, SGI et plus récemment NVidia GeForce3 ou GeForce4). Son emploi est très simple : après avoir activé un mode spécial, et désactivé la mise à jour des *buffers*, on appelle simplement les commandes OpenGL que l'on utiliserait pour rendre l'objet que l'on veut tester. On récupère alors le nombre de pixels qui passent le *z*-test. Cette fonctionnalité très pratique doit cependant être utilisée à bon escient si l'on veut être optimal. La première des choses à faire est d'utiliser une hiérarchie de volumes englobants pour classer d'emblée un ensemble d'objets. Il suffit de tester la visibilité du volume englobant. Si il est caché, on peut directement classifier comme cachés tout les objets qu'il englobe. Sinon, on descend dans la hiérarchie. On utilisera évidemment la réduction du volume englobant pour le test. Pour des volumes englobants simples (sphères, cubes pas forcément alignés avec les axes), une réduction exacte est triviale à calculer. Sinon, on utilisera une réduction externe.

Dans le cas général où l'on ne dispose que d'approximation des réductions d'objets dans la scène, nous pouvons donc quand même mettre en œuvre des calculs de visibilités efficaces. Nous verrons à la Section 1.6 comment cela peut-être utilisé pour précalculer la visibilité d'un gros modèle. Pour l'instant, nous allons exposer la méthode employée pour calculer les approximations des réductions.

1.5.3 Réduction continue de voxels

La propriété de réduction est une propriété fondamentalement volumique. Sa définition par l'équation (1.1) fait en effet intervenir une inclusion de l'élément structurant dans l'objet. Si on considère un polygone, sa réduction est vide car il est de dimension nulle dans la direction de sa normale. Lorsque l'on considère un objet décrit par des facettes polygonales, il ne faudra donc pas s'intéresser aux facettes en tant que telles, mais plutôt au volume qu'elles décrivent. On retrouve là l'idée des travaux de Schaufler selon laquelle c'est l'intérieur des objets qui nous intéresse. On décide de voxeliser cet intérieur, avec comme but de calculer la réduction sur ces voxels. On retrouve là une deuxième idée, qui est d'utiliser une représentation plus adaptée au calcul que l'on souhaite faire, au prix d'une perte de détails dont l'impact sur ledit calcul est négligeable. En effet, nous montrons maintenant qu'à partir d'une voxelisation, il est possible de calculer une réduction interne et externe de façon robuste, simple et efficace.

Partant d'un objet polygonal, nous déterminons un ensemble de voxels intérieurs à l'objet. Cette méthode est exactement celle utilisée par Schaufler (section 1.3) mais nous en donnons ici les détails. Le principe est de dicrétiser les faces dans une structure hiérarchique de grille 3D et de labeliser chaque cellule de la grille comme *frontière*, *extérieure* ou *intérieure*. Pour cela, une boîte englobante *cubique* de l'objet est déterminée. Elle n'est pas nécessairement alignée avec les axes mais peut au contraire être choisie de manière à minimiser son volume. Un octree est construit avec une cellule racine correspondant à cette boîte, subdivisée de manière à discrétiser les faces de l'objet, c'est-à-dire qu'on marque comme frontière et on subdivise tous les voxels qui intersectent une face. La subdivision est stoppée à une profondeur maximale qui définit la résolution de la voxelisation. L'algorithme 1.1 donne la procédure qui est appliquée au noeud racine pour chaque face. Elle utilise une routine d'intersection polygone/cube qui est celle de Don Hatch et Daniel Green [GH95]. Dans cette procédure, les cellules qui ne sont pas

Algorithme 1.1 Algorithme de discrétisation

Fonction *RasterFaceIn*
Entrée: une face f, un cellule d'octree c, une profondeur maximum d_{max}
Sortie: rien (l'octree est subdivisé)

```
1.  si (d_max > 0 et intersect(f,c))
2.     alors c.status ←frontière
3.           subdiviser c en 8
4.           pour i ←0 à 7
5.                 faire RasterFaceIn(f,c.fils(i),d_max − 1)
6.     sinon c.status ←intérieure
7.  retourner
```

marquées frontières sont marquées intérieures. On applique ensuite un algorithme de remplissage pour marquer toutes les cellules qui sont en réalité extérieures. Pour

cela, on récupère la liste des cellules de l'octree qui touchent une des faces de la cellule racine, c'est-à-dire une des faces de la boîte englobante, autrement dit l'extérieur. Une telle cellule, si elle n'est pas frontière doit être classifiée extérieure. On la marque comme telle et on propage récursivement à toutes les cellules adjacentes qui ne sont pas frontières. Par cellule adjacente, on entend toute cellule *pas nécessairement de même taille* qui touche une des *faces* de la cellule. On ne considère pas l'adjacence par les arêtes ou les sommets. On fait de la 8-adjacence et non de la 27-adjacence. En d'autres termes, on utilise la distance de Manhattan. À la fin de ces propagations, l'octree est donc subdivisé et chacune de ses cellules est marquée soit extérieure, soit frontière, soit intérieure. On peut noter qu'on a les propriétés suivantes :

- toute cellule extérieure ou intérieure est une feuille de l'octree ;
- toute cellule qui n'est pas une feuille est nécessairement frontière ;
- aucune cellule intérieure n'a de face en contact avec une cellule extérieure.

Les cellules intérieures ne sont cependant pas nécessairement de même taille. Pour appliquer l'algorithme de Schaufler, cela n'est pas gênant (et même souhaitable). Dans notre cas, nous voulons obtenir un ensemble de voxels de même taille, et nous subdivisons donc les cellules intérieures jusqu'au niveau maximum. Nous obtenons alors un ensemble de cellules qui approxime le volume intérieur de l'objet. Ce volume, éventuellement constitué de plusieurs composantes connexes, est celui que nous allons maintenant réduire. Pour cela, il n'est pas nécessaire de considérer toutes les cellules intérieure. Seules celles qui touchent une cellule frontière sont requises, c'est-à-dire les cellules qui sont « à la surface » du volume intérieur. Ces cellules peuvent être de neuf types différents qui sont répertoriés sur la figure 1.20. Ces types sont libellés en fonction du nombre de faces qui sont en contact avec une cellule frontière, et de leur placement relatif sur la cellule (faces opposées (O) ou adjacentes (A) par un sommet ou une arête).

À l'issue de cette première phase de voxelisation, nous stockons donc un ensemble de cellule « de surface », que nous appellerons *briques*, et un ensemble de *faces* sur ces briques. Les structures de données correspondantes sont indiquées sur la Figure 1.21. On notera que cette représentation est très compacte du fait que l'on ne stocke pas les voxels à l'intérieur du volume. Imaginons par exemple que le modèle que l'on voxellise soit une sphère de diamètre R grand par rapport au côté c des voxels. Le nombre de briques stockées est approximativement l'aire de la surface de la sphère divisée par l'aire d'une face d'une brique, soit $\pi R^2/c^2$. C'est beaucoup moins que le nombre de cellules intérieures qui lui est approximativement le volume de la sphère divisé par le volume d'une brique, soit $\frac{4}{3}\pi R^3/c^3$. Le gain est de l'ordre de R/c. En outre, la structure de donnée 1.21 est optimisée. On ne stocke pas la dimension du cube, qui est constante et est stockée dans une variable globale ; et on stocke les coordonnées du centre avec des coordonnées entières (les coordonnées réelles étant obtenues à partir d'une position de référence en multipliant par la dimension du cube).

Nous allons maintenant étudier comment éroder un tel ensemble de briques. Pour cela nous commencons par considérer un élément structurant parallèpipé-

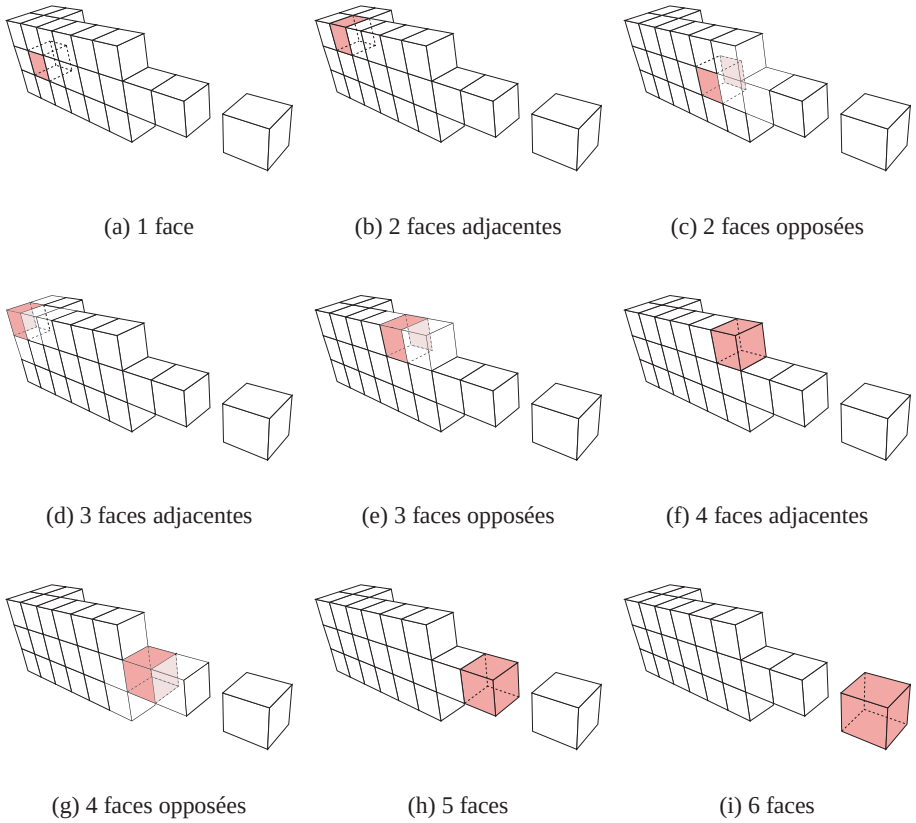


FIG. 1.20 – Types de cellule à l'interface intérieure/frontière

Seules les cellules intérieures sont représentées. Les faces en rouge sont celles qui touchent une cellule de bordure.

```
float cote;          // c t  du cube
struct Brique {
  Point centre;      // centre du cube, 3 int
  Face* faces[6];    //  ventuellement NULL
  short type;        // de 0   8
};
struct Face {
  Brique* brique;
  short pos;         // de 0   5 : quelle face de la brique
};
```

FIG. 1.21 – Structure de données pour représenter l'intérieur

dique, dont les axes sont alignés avec ceux de la voxelisation , puis nous verrons comment les résultats obtenus nous permettent de calculer des érosions internes et externes pour des éléments structurants quelconques. La raison pour laquelle nous considérons un **X** parallèpipédique vient de l'observation qu'un tel ensemble peut se décomposer comme la somme de Minkowski de trois segments, comme illustré sur la figure 1.22.

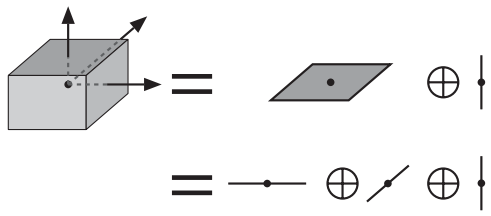


FIG. 1.22 – Décompositin d'un parallèlépipède comme la somme (commutative) de Minkowski de 3 segments.

Cette décomposition est particulièrement intéressante car nous avons la propriété d'associativité suivante :

$$\mathcal{O} \ominus (\mathbf{X} \oplus \mathbf{Y}) = (\mathcal{O} \ominus \mathbf{X}) \ominus \mathbf{Y} \tag{1.5}$$

dont voici une preuve :

$$\begin{aligned} M \in (\mathcal{O} \ominus \mathbf{X}) \ominus \mathbf{Y} \quad &\Longleftrightarrow \quad \forall \mathbf{y} \in \mathbf{Y} \;\; M + \mathbf{y} \in \mathcal{O} \ominus X \\ &\Longleftrightarrow \quad \forall \mathbf{y} \in \mathbf{Y} \; \forall \mathbf{x} \in \mathbf{X} \;\; M + \mathbf{y} + \mathbf{x} \in \mathcal{O} \\ &\Longleftrightarrow \quad \forall \mathbf{z} \in \mathbf{X} \oplus \mathbf{Y} \;\; M + \mathbf{z} \in \mathcal{O} \\ &\Longleftrightarrow \quad M \in \mathcal{O} \ominus (\mathbf{X} \oplus \mathbf{Y}) \end{aligned}$$

Autrement dit pour éroder un ensemble de points par un parallèlépipède, il suffit d'éroder sucessivement le long des trois axes du parallèlépipède, comme le résume l'équation ci-dessous :

$$\mathcal{O} \ominus \Big((\mathbf{S}_x \oplus \mathbf{S}_y) \oplus \mathbf{S}_z\Big) = \Big((\mathcal{O} \ominus \mathbf{S}_x) \ominus \mathbf{S}_y\Big) \ominus \mathbf{S}_z \tag{1.6}$$

Érosion d'une couche de briques

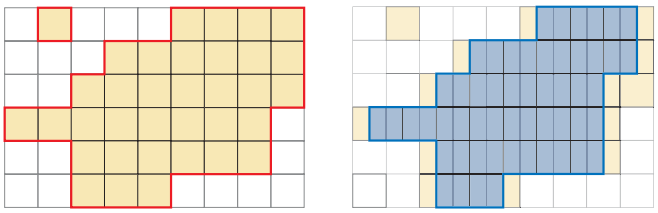
Nous avons que l'opération élémentaire a effectuer est une érosion par un segment **S**. Si nous considérons maintenant un ensemble de voxels, et en faisant l'hypothèse que le segment est aligné avec l'un des axes, par exemples l'axe des x, cette opération est particulièrement simple à effectuer, en remarquant que le segment peut de nouveau être décomposé en la somme de segments dont la longueur est égal au coté c d'un voxel et d'un segment de longueur inférieure à c, ce que nous notons

$$\begin{aligned} \mathbf{S} &= \mathbf{S}^c \oplus \ldots \oplus \mathbf{S}^c \oplus \mathbf{S}^r \text{ avec } r < c \\ &= \mathbf{S}^i \oplus \mathbf{S}^r \end{aligned}$$

En appliquant de nouveau la propriété d'associativité 1.5, on peut procéder en deux étapes, une succession d'érosion de longueur c, que nous appellerons *entières* et une dernière érosion *résiduelle*.

Érosion entière

L'érosion entière est très simple à effectuer. En effet, un ensemble de voxels ainsi érodé donne un ensemble de voxels comme le montre la Figure 1.23 dans le cas 2D. Chaque paire de voxels adjacents est remplacée par un un seul voxel au centre.



(a) (b)

FIG. 1.23 – Érosion d'un ensemble de voxels par un segment de longueur égale au coté d'un voxel. On obtient un nouvel ensemble de voxels.

Si nous utilisions une grille 3D complète pour représenter les voxels, repérer de tels paires serait trivial. Nous avons vu que pour être efficaces, nous ne représentons que les briques de surface, et il faut donc un peu plus de travail. Nous utilisons une grille « creuse » basée sur une table d'association (Figure 1.24). Cette table associe à une position i, j dans la grille un numéro indiquant si à cette position se trouve une brique de surface (on stocke alors un pointeur vers la brique), où si on se trouve à l'extérieur du volume sur une brique adjacente à une brique de surface (on stocke la valeur `NULL`). Ainsi, pour trouver une paire de briques adjacentes, on parcourt la table d'association. Si on est sur un élément i, j non `NULL`, il y a deux groupes possibles pouvant contenir cette position, le groupe (i, j),$(i+1, j)$ et le groupe (i, j) et $(i+1, j)$. Pour chacun d'entre eux, on regarde dans la table d'association si la position autre que i, j n'a soit pas de valeur associée, soit une valeur non `NULL`. Dans ce cas et dans ce cas seulement, on a un groupe de deux briques adjacentes pour lequel on peut créer une brique réduite.

On notera que lors d'une réduction entière, certaines briques disparaissent (par exemple la brique isolée en haut à gauche sur la figure 1.23. Il s'agit de toutes les briques qui ont deux faces adjacentes. Le plus représentatif de ces cas est celui des briques de type 4, qui sont réduites à un point. Autrement dit, la réduction interne ne préserve pas la topologie. Remarquons aussi la propriété suivante : si nous répétons deux fois une réduction entière, l'ensemble des briques obtenu correspond à l'érosion, classique en morphologie mathématique, des voxels intérieurs initiaux

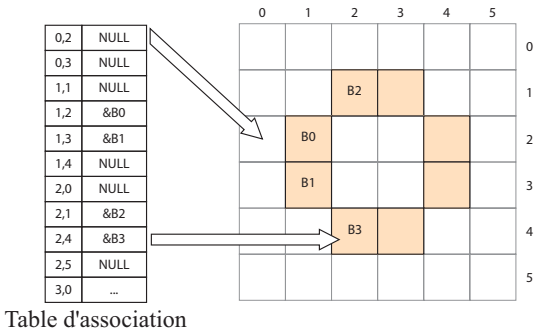


FIG. 1.24 – Grille creuse

comme le montre la Figure 1.25. Nous avons donc au passage un algorithme capable de réaliser cette opération d'érosion en ne travaillant que sur les pixels de bordure.

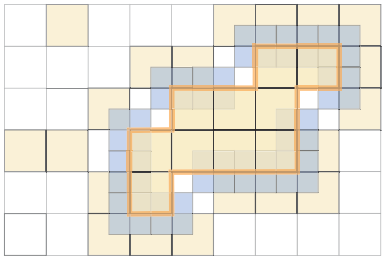
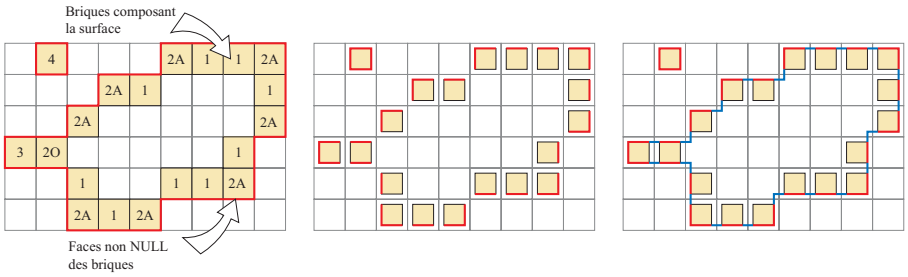


FIG. 1.25 – Réduction et érosion morphologique

Deux réductions entière successives correspondent à éroder une couche de voxels (ou de pixels en 2D). En bleu la première réduction entière, en rouge le résultat de deux érosions entières.

Érosion résiduelle

Une fois les différentes érosions entières effectuées, on dispose d'un ensemble de voxels de coté c qu'il reste à éroder par un segment de longeur $r < c$. La première chose à remarquer est que cette opération, contrairement à l'érosion entière, ne modifie pas la topologie. En effet, un voxel érodé ne peut pas « disparaître ». On a vu (Figure 1.13) qu'on ne pouvait pas se contenter d'éroder chaque voxel indépendamment. Mais puisqu'on sait que la topologie est préservée, une solution consiste à éroder chaque voxel, puis à « boucher les trous » entre voxels initialement adjacents. La Figure 1.26 illustre ce principe en deux dimensions. La réduction de chaque brique est triviale. Il faut par contre décrire où et comment boucher les trous. Ce point est expliqué directement en trois dimensions. La matière à faire apparaître forme des patches rectangulaires à placer à certains sommets et certaines arêtes des briques. On commence donc par construire une structure

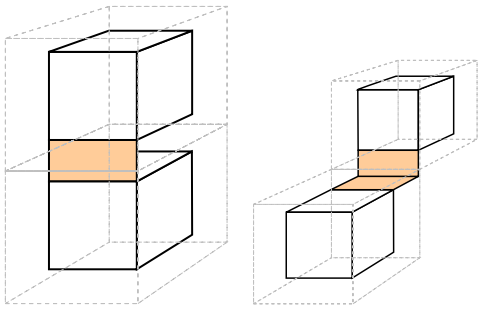


(a) (b) (c)

FIG. 1.26 – Réduction continue 2D

Considérant un ensemble de briques et de faces décrivant un volume (a)*, on réduit chaque brique individuellement* (b)*. Puis on crée des faces supplémentaires à la jonction entre chaque brique* (c)*.*

des sommets et d'arêtes stockant les relations d'ajacence. Chaque arête connait les deux faces situées de chaque côté[10], et chaque sommet connait la liste des arêtes qui le contiennent. Chaque arête est soit convexe, soit concave, soit plate en fonction des deux faces qui lui sont adjacentes. Pour chaque arête non convexe, on crée alors un ou deux rectangles comme indiqué sur la Figure 1.27. Les sommets peuvent être



(a) (b)

FIG. 1.27 – Matière à créer pour une arête concave

Après avoir érodé deux briques adjacentes (en pointillé), il faut faire apparaître de la matière (en saumon) aux arêtes de contact qui sont plates (a) *ou concave* (b)*.*

de différents types, selon le nombre de faces auxquels ils appartiennent et la configuration de ces faces. La Figure 1.28 liste les 13 types possibles, étiquettés par le

[10]Si une arête appartient à quatre faces, ce qui se produit quand deux briques sont tangentes par une arête, alors on sépare l'arête en deux arêtes distinctes.

nombre de faces et la convexité dominante des arêtes en jeu. Le label « ignoré » signifie qu'aucune matière n'est à créer pour ces sommets. Pour les autres, la matière à créer est un carré, indiqué en gris sur la Figure 1.29.

Algorithme final

Maintenant que nous avons décrit comment effectuer les opérations élémentaires dont nous avons besoin, érosions entière et résiduelle le long d'une direction, il suffit de procéder successivement le long des 3 axes en effectuant d'abord toutes les réductions entières, ce qui nous laisse avec un ensemble de voxels, puis en effectuant les dernières réductions résiduelles sur cet ensemble. Ces trois dernières réductions sont effectuées simultanément. L'équation ci-dessous résume la façon de procéder :

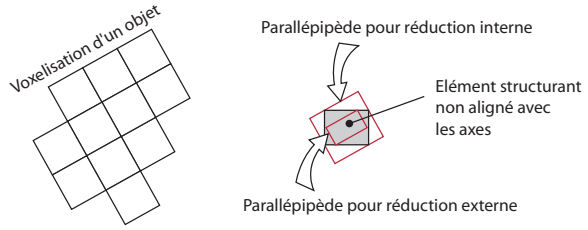
$$\begin{aligned} & \mathcal{O} \ominus (\mathbf{S}_x \oplus \mathbf{S}_y \oplus \mathbf{S}_z) \\ &= \mathcal{O} \ominus \Big((\mathbf{S}_x^i \oplus \mathbf{S}_x^r) \oplus (\mathbf{S}_y^i \oplus \mathbf{S}_y^r) \oplus (\mathbf{S}_z^i \oplus \mathbf{S}_z^r) \Big) \\ &= \underbrace{\Big(\big((\mathcal{O} \ominus \mathbf{S}_x^i) \ominus \mathbf{S}_y^i \big) \ominus \mathbf{S}_z^i \Big)}_{\text{first term}} \underbrace{\ominus (\mathbf{S}_x^r \oplus \mathbf{S}_y^r \oplus \mathbf{S}_x^r)}_{\text{second term}} \end{aligned}$$

Érosions interne et externe

Ce que nous avons décrit pour l'instant est le calcul exact de l'érosion d'un ensemble de voxels par un parallèlépipède aligné avec les axes. On se sert de ce résultat pour calculer des érosions externes et internes d'un ensemble quelconque par un élément structurant convexe. Pour cela, on utilise les propriétés suivantes :

$$\begin{array}{lll} \text{si } \mathcal{O} \subset \mathcal{Q} & \text{alors} & \mathcal{O} \ominus \mathbf{X} \subset \mathcal{Q} \ominus \mathbf{X} \\ \text{si } \mathbf{X} \subset \mathbf{Y} & \text{alors} & \mathcal{O} \ominus \mathbf{Y} \subset \mathcal{O} \ominus \mathbf{X} \end{array}$$

Pour calculer une érosion interne d'un objet par un élément **X**, il suffit de le voxeliser comme décrit précédemment (c'est-à-dire prendre tous les voxels entièrement contenu dans l'objet), puis de calculer l'érosion par le plus petit parallèlépipède contenant **X** et aligné avec les axes de la voxelisation. Pour une réduction externe, on considérera un tel parallèlépipède contenu dans **X**, et on utilisera une voxelisation légérement différente : on intégrera les voxels de bordure pour obtenir un ensemble de voxels qui contiennent entièrement l'objet.



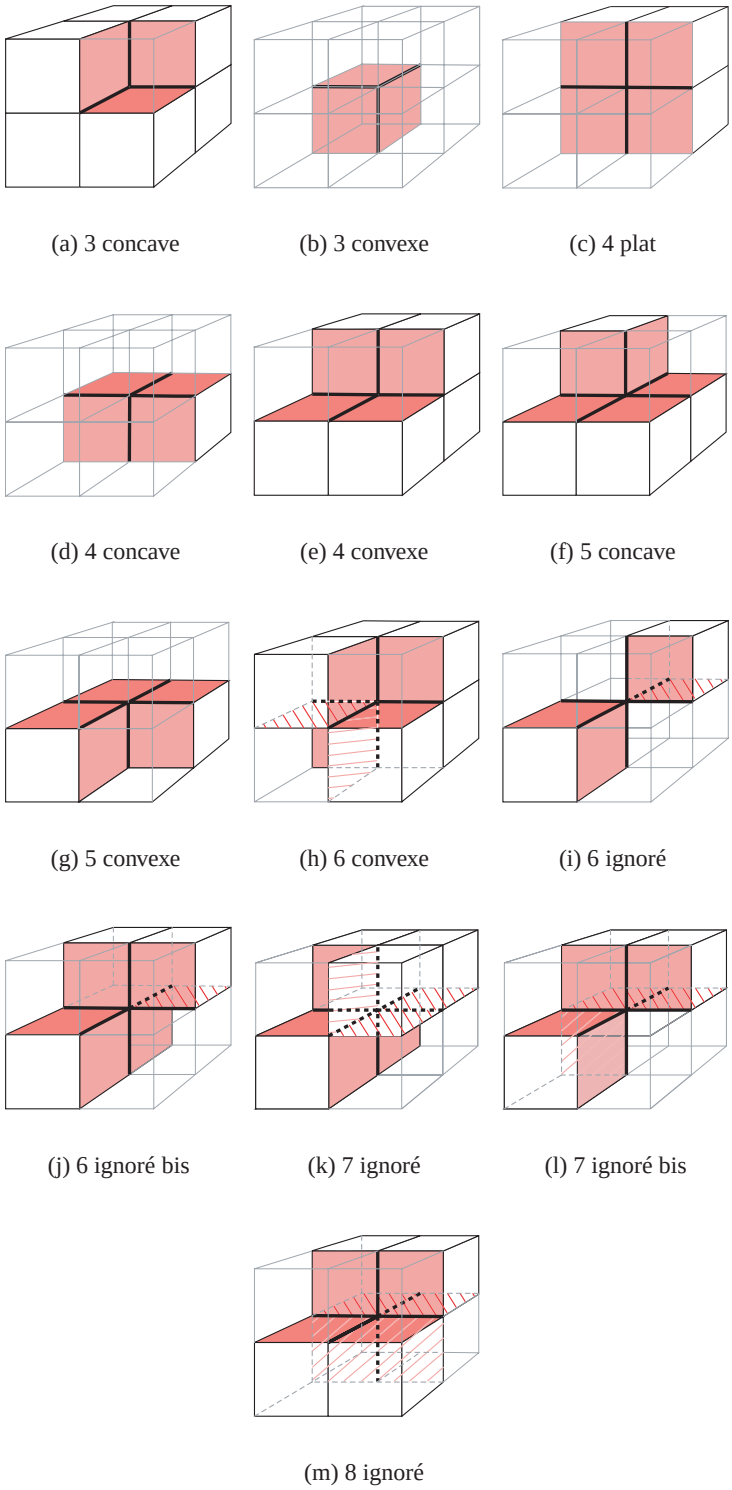


FIG. 1.28 – Les 13 types de sommets

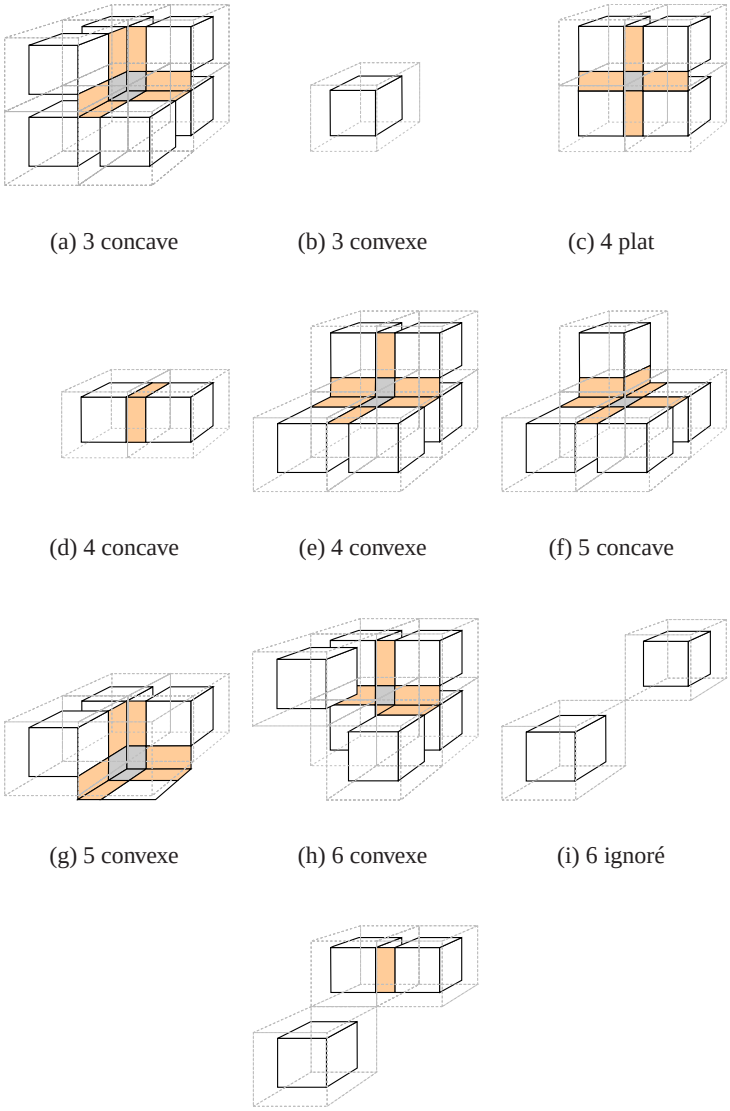


FIG. 1.29 – Matière à créer pour les différents types de sommets

Seuls deux cas « ignoré » sont montrés pour illustrer le fait que pour les sommets de ce type, il n'y a pas de matière à créer.

1.6 Résultats

1.6.1 Réduction

Nous avons appliqué l'algorithme de réduction à plus d'une dizaine de modèles variés afin de vérifier sa robustesse. Il marche sur tous les modèles sans exception. Dans cette section, nous présentons quelques images des résultats obtenus et donnons quelques chiffres. La Figure 1.30 montre un exemple sur un modèle de dinosaure *(b)* avec ses voxelisations interne *(a)* et externe *(c)*. La profondeur de l'octree utilisée est de 6. C'est à dire que la taille d'un cube est $1/2^6$ fois la plus grande dimension de la boîte englobante du dinosaure. Dans l'application dont sont

(a) (b) (c)

FIG. 1.30 – Dinosaure, voxelisations interne et externe

tirées ces images, la réduction continue est calculée en temps réel en manipulant un curseur. La Figure 1.31 montre deux étapes de cette réduction. Le pourtour bleu indique la voxelisation interne originale, et met en évidence la réduction effectuée.

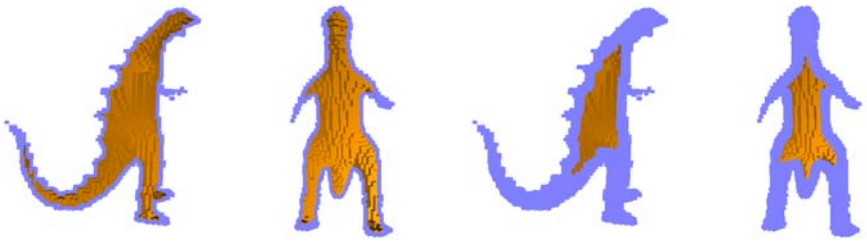
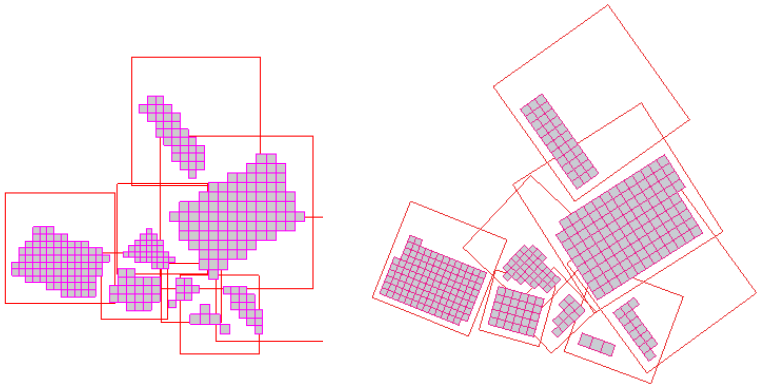


FIG. 1.31 – Dinosaure, réduction continue

1.6.2 Pré-calcul de la visibilité

Nous avons utilisé l'algorithme basé sur le théorème de réduction et sur notre approximation par réduction de voxels pour pré-calculer la visibilité pour deux modèles de villes. Ces modèles sont des modèles procéduraux générés à partir d'empreintes au sol de bâtiments. Le modèle est fondamentalement 2D et demi mais nous le traitons comme un modèle 3D pour montrer que notre algorithme est générique. en outre, notre générateur procédural intégrera bientôt des effets 3D (ponts, couloirs, etc...). Le graphe des rues est retrouvé en utilisant un algorithme que nous avons développé et publié [DS02]. On en déduit l'ensemble des positions que peut occuper l'utilisateur et on recouvre cet ensemble de positions par des cellule sphériques. La taille de ces cellules est choisie de manière à ce qu'une rue soit couverte en largeur par deux cellules en moyenne. Cette taille est la même pour toute les cellules.

Le programme procède en voxelisant chaque bâtiment et en calculant une fois pour toutes leurs réductions externes et internes correspondant à la taille des cellules. Les boîtes englobantes utilisées pour la voxelisation sont choisies objets par objets ce qui nous permet d'avoir une meilleure discrétisation comme le montre la Figure



(a) (b)

FIG. 1.32 – Voxelisation des objets avec des boîtes englobantes alignées (a) ou non (b) avec les axes. Les boites englobantes sont choisies cubiques, d'où leur étendue horizontale causée par la hauteur des bâtiments.

Pour chaque cellule nous effectuons un rendu dans les six directions couvrant les six faces d'un cube, en placant la caméra au centre de la cellule. Les réductions internes sont utilisées pour ce rendu. Nous obtenons une carte d'occlusion. Nous utilisons ensuite le mécanisme d'*Occlusion query* des cartes NVidia GeForce4 pour déterminer pour chaque objet si sa réduction externe est visible ou non et nous en déduisons le PVS de la cellule.

Cette implémentation est directe et brutale. Comme nous l'avions dit, l'intérêt de l'approche par réduction est que l'on peut utiliser différentes méthodes pour

accélérer le rendu des six vues du centre d'une cellule. Dans notre implémentation, nous utilisons seulement un algorithme de *Hierarchical Frustum culling* [AM00]. Les temps obtenu pour chaque rendu sont de 5 à 10 fois moindre que si l'on rend naïvement toute la scène à chaque fois. La hiérarchie que nous utilisons est un arbre binaire construit par un algorithme de clusterisation [Mur83]. Cette hiérarchie pourrait être utilisée pour le test d'occlusion, mais nous n'avons pas implémenté cette partie là pour l'instant. En effet, la fabrication de la carte d'occlusion et la série de tests par bâtiments sont pour l'instant faites successivement. Mais il est clair que ces phases pourraient être effectuées en parallèle pour ne pas rendre dans la carte d'occlusion les bâtiments qui ne sont pas visibles et ainsi accélérer le calcul.

Le tableau 1.1 indique les différents paramètres et les temps de calcul pour les deux modèles que nous avons considérés et qui sont montrés sur la Figure 1.33. Le temps total est le temps moyen par cellule multiplié par le nombre de cellules, plus quelques minutes pour calculer intialement les réductions. Les temps de calcul sont tout à fait raisonnable étant donné la taile des modèles traités, et compte tenu du fait que nous faisons un pré-calcul. Nous pensons cependant pouvoir diviser ces temps au moins par deux en implémentant des methodes d'accélération de rendu, et notamment en utilisant des *display lists*.

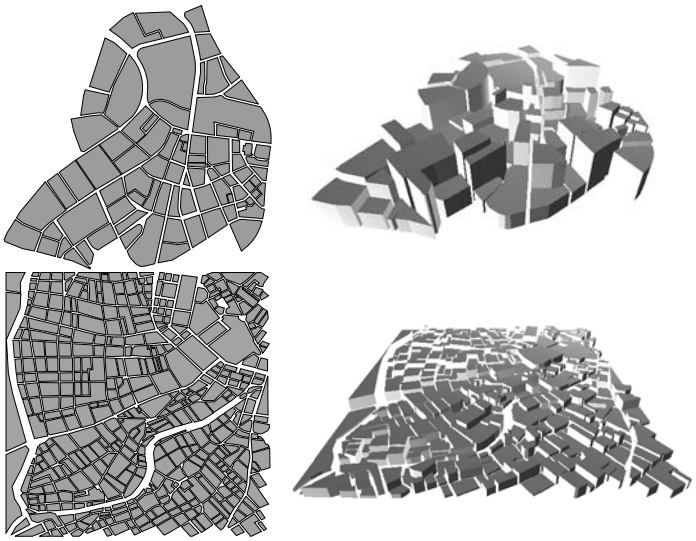


Fig. 1.33 – Boston et Vienne

	# de	# de	# de	temps de pré-calcul	
	bâtiments	triangles	cellules	par cellule	total
Boston	87	8 044 tris	7339	859 ms	1h47
Vienne	458	39 679 tris	12094	1 485 ms	5h05

Tab. 1.1 – Résultats du précalcul

Il est difficile de montrer le résultat du calcul de visibilité dans un document manuscrit. La Figure 1.34 montre ce qui est calculée pour une cellule donnée. Comme nous l'avons dit, la visibilité calculé en le centre de la cellule (en jaune) n'est pas une bonne approximation car ce point n'est pas dans l'axe de la rue. Avec la réduction, le résultat calculé est correct.

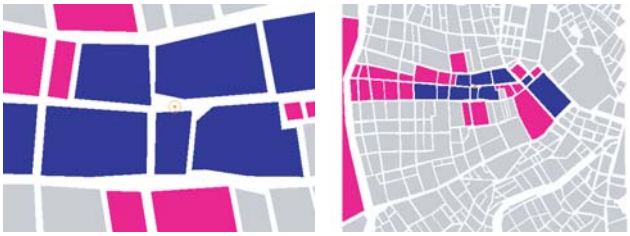


FIG. 1.34 – Un exemple de précalcul : en bleu les bâtiments visibles du centre de la cellule, en rose les bâtiments potentiellement visibles de la cellule.

Les résultats que nous avons présentés dans cette section sont préliminaires, et nous pensons qu'un certain nombre d'applications méritent d'être explorées. Nous aimerions notamment réaliser le programme qui exploite les PVS calculés lors de la visualisation du modèle. Sur un plan plus théorique, nous aimerions utiliser le programme de pré-calcul dont nous disposons pour essayer d'analyser les évènements visuels dans une ville. Une première piste consiste à étudier la façon dont évolue le PVS entre cellules contigues. L'idée sous-jacente est d'essayer d'optimiser le placement et la forme des cellules. Si plusieurs cellules proches ont presque le même PVS, peut-être faut-il regrouper ces cellules. Si un groupe d'objets sont toujours cachés ou visibles simultanément, ces objets peuvent aussi être regroupés. Cette information pourrait éventuellement servir pour un algorithme de clusterisation. Nous aimerions aussi étudier l'impact du choix de la taille des cellules sur la sur-conservativité de l'algorithme.

1.7 Analyse et perspective

Ce chapitre comporte plusieurs contributions que nous discutons ici. Nous avons tout d'abord présenté une analyse détaillée des travaux de Schaufler *et al* où nous étudions le type d'occlusion (c'est-à-dire les cas de fusion des pénombres) que cette méthode peut gérer. Nous montrons également les limitations de l'algorithme. La raison profonde de ces limitations est qu'une seule et même structure hiérarchique est utilisée pour discrétiser deux choses différentes : le modèle d'une part et les limites d'ombres d'autre part. Or ces deux choses n'ont pas du tout la même forme. Ce n'est pas parce que l'on discrétise finement le modèle que l'on discrétise correctement les limites d'ombre, et c'est pourquoi on peut ne pas détecter certaines occlusions. Une autre limitation est le fait que l'on travaille dans l'espace objet. La principale conséquence est que l'octree utilisé doit contenir *tout*

le modèle et qu'il est donc impossible de traiter des modèles trop étendus. En outre cet octree doit résider en mémoire principale, ce qui peut présenter une sérieuse contrainte [Lin00].

Nous avons alors présenté un algorithme de visibilité capable de traiter des modèles de taille quelconque, en prenant en compte tous les types d'occlusion. Cet algorithme est basé sur une propriété de réduction des bloqueurs, introduite par Wonka *et al*, et sur une propriété de réduction des objets cachés, que nous avons démontrée. Grâce à ces propriétés, le problème de la visibilité étendue est ramené à un problème ponctuel, qui se résoud en fabriquant une carte d'occlusion et en testant les objets par rapport à cette carte. Cela permet de s'affranchir des limitations discutées précédemment. En effet, le calcul de visibilité est effectué dans l'espace image et non plus dans l'espace objet. D'une part cela veut dire que la discrétisation des limites d'ombre, et donc de la visibilité, correspond à la résolution de l'image et se trouve donc d'une part décorrélée de toute autre discrétisation, et d'autre part adaptée à l'utilisation que l'on en fait. De la même manière, toute hiérarchisation des occlusions dans l'espace image (par exemple les *Hierarchical Occlusion Map* [ZMHKEH97]) peut être décorrélée d'une hiérarchisation des requêtes, qui pour sa part a lieu dans l'espace objet. D'autre part, l'image est de taille mémoire fixe, et la carte d'occlusion est générée en y accumulant les contributions des objets. Cela veut donc dire qu'il n'est pas nécessaire d'avoir toute la scène en mémoire. Enfin, l'utilisation de cartes d'occlusion permet en pratique de réduire la complexité. À priori cette complexité est linéaire : chaque objet est parcouru une fois pour fabriquer la carte puis une fois pour tester sa visibilité. Mais la fabrication de la carte, non contente d'être accélérée par le processeur graphique, peut utiliser n'importe quelle méthode existante : *Hierarchical Frustum Culling*,*Hierarchical Occlusion Maps*, précalcul grossier, volumes d'ombres de gros bloqueurs convexes, etc... En pratique, on obtient ainsi une complexité sub-linéaire. Signalons enfin que dans le cas où l'on sait calculer exactement les réductions des objets de la scène, nous avons montré que la fabrication de la carte d'occlusion et le test des objets par rapport à cette carte peut être effectué très simplement en une seule et même passe. Dans le cas où l'on ne dispose que d'approximations des réductions, deux passes sont nécessaires, mais la deuxième peut se faire efficacement en utilisant les nouvelles fonctionnalités des processeurs graphiques.

La dernière contribution est justement la fabrication de réductions approximatives encadrant la réduction exacte (réduction interne et externe). Nous avons montré comment réduire de n'importe quelle distance un ensemble de voxel. Les points forts de la méthode sont les suivants. Tout d'abord, la méthode proposée est très robuste car les cas possibles sont en nombre fini et facilement identifiables. En outre, nous avons montré une implémentation efficace qui permet de réduire les coûts mémoires habituellement élevés quand on utilise des discrétisations volumiques. Enfin et surtout, la méthode peut réduire de n'importe quelle distance, indépendamment de la taille des voxels. Cela veut dire qu'il est possible de discrétiser par des voxels grossiers, tout en travaillant avec des cellules de visibilité de

petite taille (c'est-à-dire une petite distance de réduction). Une réduction naïve par simple érosion morphologique (dont on a vu qu'elle correspond à une réduction du côté d'un cube par notre méthode) obligerait à travailler à des discrétisations éventuellement très fines sous peine d'effectuer des réductions trop brutales. Non seulement nous n'avons pas cette contrainte, mais nous pouvons en outre utiliser différents degrés de voxelisation pour les différents objets de la scène, par exemple en fonction de leur taille ou encore de leur forme. Signalons enfin que si l'utilisation de voxels est inspirée des travaux de Schaufler, la façon dont nous les utilisons est plus souple. En effet, ils servent à approximer l'intérieur des objets et faciliter le calcul de la réduction, mais pas directement à calculer la visibilité ou à hiérarchiser les requêtes. La principale conséquence est que chaque objet a sa voxelisation propre. Cela permet d'une part de choisir une orientation des voxels adaptée *pour chaque objet*, et d'autre part de ne pas discrétiser le vide entre les objets comme c'est le cas si on utilise une structure englobant toute la scène.

Toute science serait superflue si l'apparence des choses coïncidait avec leur essence.

Karl Marx

2

Travaux antérieurs

Pour visualiser un modèle en temps constant, une solution théorique existe : stocker les images du modèle correspondant à tous les points de vue possibles. Il suffit alors dynamiquement de récupérer l'image correspondant au point de vue courant, et de l'afficher. Évidemment, le coût de stockage est infini et cette solution n'est pas réalisable. Or, il est clair qu'il y a une redondance d'information et que l'on pourrait partager de l'information entre les différents points de vue.

Pour cela, on peut *factoriser* l'information sous une forme qui sera utilisable pour tous les points de vue, au prix d'une opération d'extraction. C'est ce que fait un modèle polygonal. La forme du modèle est décrite par les coordonnées tridimensionnelles de ses sommets. Pour générer des images, il faut alors *projeter* en deux dimensions. Cette fois, il y a redondance de traitement. Pour obtenir l'image 2D, il faut projeter tout le modèle, même si certaines parties auront été au final traitées pour rien (partie cachées par exemple, ou détails supérieurs à la résolution de sortie).

On peut voir ces deux approches comme les cas limites d'un compromis entre représentation et coût de traitement. La première, la représentation à base d'image est optimale au niveau du traitement car l'image associée à un point de vue contient exactement l'information à afficher et rien de plus. La deuxième, la représentation polygonale est optimale au niveau de la représentation car l'information permet de générer une image pour n'importe quel point de vue. Chaque représentation a ses limitations. La représentation polygonale pose un problème de résolution. Comme nous l'avons déjà dit, selon le point de vue, une partie de l'information contenue dans une description polygonale est inutile, soit parce qu'elle n'est pas visible, soit parce qu'elle comporte des détails dont la résolution est nettement inférieure

à celle de l'image que l'on veut obtenir. Autrement dit, le modèle polygonal est trop complexe, et on a besoin de le simplifier. La section 2.1 détaille ce problème et présente différents travaux qui y sont consacrés.

De son côté, la représentation à base d'image pose le problème de la taille des données, le nombre d'image étant potentiellement infini. La section 2.2 présente différents travaux pour réduire le nombre d'image, et obtenir celles manquantes par interpolation. Pour bénéficier des avantages des deux représentations, il paraît naturel d'essayer de les combiner. Dans la section 2.2 nous présenterons aussi différentes méthodes hybrides proposées dans la littérature.

2.1 Simplification de maillage

La simplification automatique de maillage et la génération de niveaux de détails est un problème important en informatique graphique, qui a donné lieu à de nombreux travaux et publications. Le livre de Luebke et de ses coauteurs [LRC+02], récemment paru et qui fait désormais référence sur la question recense 339 articles (voir `http://lodbook.com/pubs/`) sur le sujet ! Le lecteur qui souhaite approfondir la question pourra se référer à cet ouvrage. Notre but dans cette section est de donner notre lecture de ces publications et un aperçu des différentes stratégies (construction incrémentale, clusterisation, re-construction d'une approximation) et des différentes variations proposées (choix de métriques, de critères). Étant donné la richesse des publications et l'existence d'un ouvrage récent et exhaustif, nous avons choisi de donner un aperçu des papiers que nous considérons commes marquants. Nous espérons ainsi donner au lecteur un sens de parcours, un mini-guide pour découvrir ce vaste sujet.

D'autres avant nous ont proposé différents tours d'horizon. Pour une introduction en douceur, présentant brièvement et avec des illustrations claires les différents algorithmes, le lecteur pourra commencer par lire l'aperçu donné par Erikson [Eri96]. Pour une taxonomie et des comparaisons plus poussées, il pourra ensuite se référer aux deux études de Luebke [Lue97] et Heckbert et Garland [HG97]. À la même période, Krus a également présenté sous différentes formes, et notamment en langue française, un tour d'horizon sur les niveaux de détails et la simplification polygonale [KBGT96, KBGT96, KBGT97]. Signalons également l'analyse publiée par Véron et Léon l'année précédente [VL96].

2.1.1 Simplification de maillage, quand et pourquoi faire ?

Les maillages que l'on considère sont de différents types. Ils sont par exemple produits par des scanners 3D qui échantillonnent des points sur des objets puis essayent de reconstruire la surface en triangulant ces points. Selon la résolution du scanner, et donc le nombre de points, le maillage peut être très complexe. La statue digitalisée du David de Michel-Ange [LPC+00b] contient ainsi 480 millions de polygones correspondant à des détails très fins (0,29mm). D'autres maillages

sont obtenus par extraction d'une isosurface d'un champ de potentiel. C'est le cas par exemple dans l'étude et la visualisation de certaines fonctions mathématiques, ou encore en physique-chimie pour analyser la forme d'un champ magnétique autour d'une molécule. Ces isosurfaces sont en général extraites par un algorithme de *marching cubes* [LC87]. Un autre exemple important est celui où le champ de potentiel est en fait un ensemble de données volumiques issues notamment de reconstruction par tomographie (imagerie médicale, ou analyse de blocs mécaniques en *reverse engineering*). Selon la résolution de la grille utilisée, pour les données volumiques ou pour l'algorithme de *marching cubes*, la surface polygonale récupérée peut là encore contenir un très grand nombre de polygones. Une autre grande classe de maillages sont les maillages issus de logiciels de CAD. Ces maillages sont obtenus par triangulation à partir d'une spécification par CSG du modèle. Outre le fait qu'ils contiennent également un grand nombre de polygones, ces maillages ont la particularité d'être généralement « mal conditionnés ». C'est-à-dire qu'il ne sont pas topologiquement simples (ce ne sont pas des variétés), présentent des dégénerescences, et sont disponibles sous forme d'une « soupe de polygones » exempte de toute information de hiérarchie ou d'adjacence. Même si il existe un certain nombre d'outils et d'algorithmes pour « nettoyer » ces maillages [BK97, BS95], ils restent difficiles à traiter et sont la pierre de touche des algorithmes qui se veulent génériques. Ensuite, il y a tous les maillages issus de logiciels de modélisation et produits par les graphistes travaillant pour l'industrie du cinéma et celle du jeu. Ces maillages sont en général plus propres et disposent souvent d'une organisation hiérarchique. S'ils sont souvent moins complexes géométriquement (moins de polygones), ils présentent une autre forme de complexité liée aux attributs qui leur sont attachés (couleur, textures, matériaux) et qui posent des problèmes pour les algorithmes de simplification. Distinguons enfin une dernière classe de maillage, les terrains, pour laquelle des algorithmes spécifiques peuvent être développés, qui exploitent les caractéristiques particulières (champ de hauteur, vue sous des angles rasants) que nous ne considérerons pas ici (voir le site `http://www.vterrain.org/LOD/Papers/index.html` pour tout ce qu'il faut savoir sur les terrains).

Tous ces maillages ont besoin d'être simplifiés pour plusieurs raisons. D'une part, le maillage peut contenir des informations à différentes résolutions. Par exemple la statue digitalisée du David de Michel-Ange a une échelle de détail allant de la forme globale au micro-relief des coups de ciseaux du sculpteur. Selon ce que l'on souhaite visualiser ou éditer, une partie de l'information peut être filtrée. On parlera alors de *multi-résolution*. Dans ce cadre, on cherchera non seulement à effectuer un filtrage en satisfaisant certaines propriétés, mais aussi à encoder l'information de manière à pouvoir accéder efficacement, et idéalement de manière adaptative, aux différentes résolutions. D'autre part, le maillage peut-contenir plus d'information que l'on n'est capable d'en traiter. L'exemple typique est celui de l'affichage. Un modèle de personnage dans un jeu vidéo n'est pas forcément intrinsèquement trop complexe, mais lorsque l'on doit l'afficher en plus d'autres éléments de la scène, il contient plus de polygones que le budget alloué. Dans ce cas, on

cherchera à générer différentes versions de complexité décroissante qui seront utilisées alternativement pour obtenir la meilleure qualité en respectant la contrainte budgétaire. On parlera alors de *niveaux de détails*, notion introduite dès 1976 par Clark [Cla76] qui montre comme d'autres plus tard [CS81, FST92, HG94, Sew97] le bénéfice que l'on peut tirer d'avoir plusieurs représentations d'un même modèle.

Selon le cas, ce ne sont pas les mêmes propriétés que l'on attendra de l'algorithme de simplification. Par exemple, si l'on visualise une molécule chimique, on désirera que le genre topologique de la surface soit préservé car les tunnels et autres trous dans la surface sont les éléments que l'on étudie pour déterminer les propriétés de la molécule. Si l'on simplifie un bâtiment pour l'afficher au loin dans un jeu, on peut au contraire ignorer complètement ce genre de contraintes et par contre s'attacher à préserver la silhouette qui forme la signature visuelle du bâtiment. Les différentes propriétés que le lecteur devra garder à l'esprit en lisant les descriptifs des différents algorithmes sont les suivantes :

Préservation de la topologie. L'algorithme conserve-t-il le nombre de composantes connexes et le genre (nombre de trous) du maillage initial ? Si cette préservation peut être cruciale en fonction de l'application, elle impose aussi d'avoir des maillages pour lesquels la topologie est bien définie. En outre, cette topologie doit être représentée (généralement sous forme d'une structure d'adjacence) et induit des coûts mémoire supérieurs, notamment parce que la structure doit résider entièrement en mémoire.

Gestion d'une soupe de polygones. Certains algorithmes sont capables de traiter des soupes de polygones, c'est-à-dire des ensembles non structurés de polygones n'ayant généralement pas une topologie bien définie. Cette propriété sera recherchée pour des algorithmes capables de traiter n'importe quel type de modèle.

Coût mémoire. Si le but est de simplifier des maillages complexes, on peut être amené à considérer des maillages dont la taille (taille brute et taille supplémentaire requise par l'algorithme) peut excéder les capacités d'une machine donnée.

Facilité d'implémentation et d'utilisation. Certains algorithmes ont des implémentations non triviales, qui peuvent notamment poser des problèmes de robustesse. En outre, ils peuvent requérir que l'utilisateur spécifie des paramètres plus ou moins intuitifs. Dans le choix d'un algorithme, ces deux aspects peuvent jouer un rôle important.

Encodage. Le problème de la simplification est lié au problème de la représentation. Si l'on génère différents niveaux de détails pour un modèle, comment les représenter ? Un bon encodage permettra par exemple de transmettre progressivement un maillage complexe en communiquant successivement les différents niveaux de détails.

Transition continue. Un bon encodage permettra également une transition continue entre deux niveaux de détails successifs, limitant ainsi les artefacts visuels qui résulteraient d'une transition abrupte.

Utilisation dépendante du point de vue. Un bon encodage permettra enfin de représenter différentes parties d'un maillage à différents niveaux de détails. Des algorithmes *dynamiques* pourront alors utiliser les résultats du calcul *statique* des niveaux de détails pour décider, en fonction du point de vue, comment seront affichées les différentes parties d'un maillage.

Prise en compte des attributs. La plupart des algorithmes de simplification ont été developpés en se concentrant sur la géométrie, c'est-à-dire la forme du maillage. Or les sommets d'un maillage qui sont redondants au niveau de la forme peuvent être indispensables à la spécification d'attributs tels que la couleur ou les coordonnées textures. Nous verrons comment certains algorithmes peuvent être adaptés pour prendre en compte les attributs spécifiés sur un maillage.

Orienté erreur ou budget. Un algorithme de simplification peut travailler dans deux directions. Soit il simplifie au maximum un maillage dans les limites d'une erreur maximale autorisée. Soit il simplifie le maillage jusqu'à une complexité donnée, en minimisant l'erreur commise.

Pour une comparaison et une classification des différents algorithmes selon ces propriétés, le lecteur se référera plutôt à [Lue97, HG97, LRC+02], notre présentation se limitant à un aperçu historique.

Nous distinguons trois grandes classes d'approches pour simplifier un maillage. Elles sont étudiées dans les trois sections suivantes. Nous aborderons ensuite le problème de la gestion des attributs (section 2.1.5) et présenterons enfin différents travaux et approches supplémentaires que nous considérons d'intérêt.

2.1.2 Clusterisation de sommets

En 1993, Rossignac et Borrel introduisent l'idée de simplification par clusterisation [RB93]. L'idée principale est de regrouper les sommets d'un maillage, ce qu'on appelle *clusteriser*, et de remplacer tous les points d'un cluster par un représentant bien choisi. Ce faisant, un certain nombre de primitives sont dégénérées : un triangle dont deux ou trois sommets sont dans un même cluster deviendra une arête ou un point. En éliminant les redondances parmi ces primitives dégénérées, on diminue la complexité du maillage. On notera que les auteurs n'éliminent pas les triangles dégénérés mais les redondances. Ainsi dix triangles dont les sommets sont tous dans le même cluster seront simplifiés et rendus comme un seul point. Cela implique de déterminer les redondances et donc d'avoir une structure adaptée. Pour calculer efficacement cette structure, les auteurs requièrent que le modèle soit au préalable triangulé, alors que la méthode est *a priori* valable pour des polygones quelconques.

Dans cette approche, il faut choisir la taille et le placement des clusters. Les auteurs utilisent une grille régulière dont la résolution est fixée par l'utilisateur. Il faut aussi définir l'heuristique de choix d'un représentant. Pour cette dernière, les auteurs attribuent à chaque sommet un poids et proposent de prendre comme re-

présentant la moyenne pondérée des sommets, ou le sommet de poids maximal. Le premier choix lisse la surface, alors que le second élimine les détails en préservant les autres faces. Le poids fait intervenir deux critères : l'appartenance probable à la silhouette, estimée par l'inverse de l'angle dihédral maximal θ entre deux arêtes incidentes au sommet, et l'appartenance à une grande face, car cette dernière ne devrait pas être modifiée par la suppression de petits détails (et donc ses sommets doivent être préservés, c'est-à-dire avoir un poids plus grand). La complexité de l'algorithme est $O(n)$ où n est le nombre de sommets initial, car chaque sommet n'est parcouru qu'une seule fois (les représentants des clusters peuvent être construits au vol).

Quatre ans plus tard, Low et Tan reprennent l'idée en proposant quelques améliorations [LT97]. Il utilisent un autre schéma de clusterisation baptisé *floating-cell*. Les sommets sont toujours notés (attribution d'un poids), mais un cluster est construit centré sur le sommet de poids maximal et contenant tous les sommets dans un rayon donné. Le processus est réitéré jusqu'à clusterisation totale. Lorsqu'un sommet appartient à plusieurs clusters, il est assigné à celui dont le centre est le plus proche. Les auteurs font remarquer que ce schéma réduit la probabilité que deux sommets de poids fort soient dans le même cluster c'est-à-dire regroupés ensembles, ce qui ferait disparaître des détails importants (indiqués par le poids fort). Les auteurs montrent également qu'il est plus efficace d'estimer, dans le calcul du poids, la probabilité d'appartenance à la silhouette par $\cos(\theta/2)$ plutôt que par $1/\theta$. En raison de la nécessité de trier les sommets par poids décroissant, la complexité de leur algorithme est $O(n \log n)$.

De leur côté, Schaufler et Stürzlinger proposent en 1995 une méthode similaire mais utilisant un schéma de clusterisation différent [SS95]. Un arbre binaire de cluster de points est construit en utilisant une technique de clusterisation hiérarchique [Mur83]. On commence avec un cluster par point, et on regroupe les deux clusters les plus proches. La distance, appellée *dissimilarité* par les auteurs, entre deux clusters est la distance entre les deux barycentres des points dans chaque cluster. Partant de la racine, on descend dans l'arbre en s'arrêtant aux noeuds dont la dissimilarité entre les deux fils est inférieure à un seuil fixé (correspondant au diamètre des détails que l'on veut négliger). Les points de ces clusters terminaux sont remplacés par un représentant, et on élimine les triangles dégénérés comme précédemment. Les auteurs choisissent comme représentant le point du cluster le plus éloigné du centre de l'objet, en constatant que le choix du point moyen, s'il donne des surfaces plus lisses, a tendance à faire diminuer le volume des niveaux de détails. L'algorithme a une implémentation en $O(n^2)$ en raison de la recherche des paires de points proches, mais il est possible de réduire cette complexité [Mur83]. L'intérêt de cette méthode est d'une part que la clusterisation est adaptée à l'objet, et d'autre part qu'elle permet, une fois construite, de fabriquer différents niveaux de détails en changeant le critère de descente dans la hiérarchie. En 1997, Gernot Schaufler et Dieter Schmalstieg représenteront la méthode d'une manière plus détaillée, en explicitant une façon efficace de coder, de compresser et d'utiliser dynamiquement leur arbre de clusterisation [Sch97a]. Un niveau de détail est initiale-

ment construit, puis peut-être sélectivement raffiné ou simplifié en développant ou en refermant les noeuds de l'arbre. Cela peut-être fait en temps réel en conservant une liste de noeuds, de sommets et de triangles actifs.

La même année, Luebke et Erikson publient à *Siggraph* une approche similaire [LE97]. Leur méthode, baptisée *Hierarchical Dynamic Somplification* (HDS) utilise aussi un arbre de sommets dont les noeuds sont developpés ou refermés dynamiquement. Le critère utilisé pour cela tient compte de la taille, dans l'image, de la projection du volume englobant d'un cluster. Si il est supérieur à un seuil fixé, le noeud est developpé. Pour ajouter plus de détail sur la silhouette, le seuil est inférieur pour les noeuds qui sont potentiellement sur la silhouette que pour les autres. Cela est déterminé en comparant le cône des normales des faces dans un cluster, et le cône d'apex la caméra et englobant le cluster. On détecte au passage les clusters dont toutes les faces sont orientées vers l'arrière, clusters qui peuvent être éliminés du rendu. Pour que le parcours de l'arbre soit efficace, la cohérence temporelle est exploitée en maintenant une liste des triangles actifs. Pour construire la hiérarchie de sommets, les auteurs proposent d'utiliser n'importe quelle méthode de regoupement itératif de sommets.

En effet, comme le remarquera [Lin00], la clusterisation de sommets n'est qu'un cas particulier d'une opération élémentaire : le regroupement de deux sommets. Le regroupement des points d'un cluster peut être décomposé en regroupement deux à deux. Parallèlement aux approches par clustering, de telles approches incrémentales ont été developpées, où le maillage est simplifié en *décimant* successivement les sommets.

2.1.3 Décimation de primitives

L'idée la plus naturelle pour simplifier un maillage contenant trop de détails est d'enlever successivement des sommets. Cette idée a été d'abord proposée à *Siggraph* en 1992 par Schroeder, Zarge et Lorensen [SZL92]. Il faut déterminer deux choses : quels sont les sommets que l'on peut enlever et dans quel ordre, et que faire des triangles qui passaient par ce sommet ? Pour qu'un sommet puisse être enlevé, il faut que localement on soit sur une surface. On commence par déterminer les *types* des sommets en construisant une structure d'adjacence sommets, arêtes, faces. Si chacune des faces incidentes à un sommet a une ou deux faces adjacentes, alors on est sur la surface ou sur une bordure. Le sommet peut être enlevé si la distance à la surface locale est inférieure à un seuil fixé. Cette distance est calculée différemment selon le type de sommet. Si l'on est sur la bordure, on calcule la distance à l'arête joignant les deux sommets adjacents sur la bordure. Sinon, on calcule la distance au plan moyen des sommets adjacents. Pour un tel sommet, on peut également regarder si deux exactement des arêtes adjacentes ont un angle dihédral supérieur à un seuil. Dans ce cas, on considére que l'on est sur une arête d'intérêt et l'on calcule la distance à cette arête. Une fois le sommet enlevé, il laisse un trou dans la surface qui doit être retriangulé. La triangulation se fait en projetant la bordure du trou sur un plan « moyen » et en triangulant en 2D. Comme la bor-

dure a une forme étoilée, un algorithme efficace de triangulation peut-être utilisé. Cet algorithme n'aboutit cependant pas toujours, auquel cas l'enlèvement du sommet est simplement annulé. En vertu de la relation d'Euler [dBvKOS00], chaque enlèvement de sommet supprime deux triangles (un si l'on est sur une frontière).

La même année, Turk propose, à *Siggraph* également, un algorithme de décimation de maillage [Tur92]. Alors que l'approche de Schroeder produit des maillages s'appuyant sur les points originaux, Turk propose plutôt de distribuer un nombre fixé de points à la surface du maillage (en placant plus de points au endroits de forte courbure gaussienne) et de retrianguler ces points. Pour effectuer cette triangulation, qui n'est pas à priori une chose aisée puisqu'il s'agit ni plus ni moins que de reconstruire une surface à partir de points échantillons, il commence par trianguler *mutuellement* les anciens et les nouveaux points, puis par supprimer les anciens points lorsque cela ne modifie pas la topologie locale, procédant en cela de la même manière que Schroeder *et al*.

L'année suivante, en 1993, Hoppe et ses coauteurs de l'université de Washington publient, toujours à *Siggraph*, *Mesh Optimisation* [HDD+93]. Le papier est plutôt présenté comme un problème de reconstruction d'une surface à partir d'un ensemble non-structuré de points. Partant d'une première reconstruction, par exemple obtenue par [HDD+92], le maillage est optimisé pour obtenir une meilleure reconstruction. Mais comme le font remarquer les auteurs, en prenant comme maillage initial un maillage quelconque, la méthode peut-être appliquée pour simplifier ce maillage. Ce papier introduit plusieurs concepts clés. Il présente tout d'abord le problème comme la minimisation d'une fonction d'énergie sur l'ensemble des maillages. Cette énergie fait intervenir trois termes, un qui mesure la fidélité au maillage initial, un qui mesure la complexité du maillage obtenu, et un qui mesure la bonne conformation spatiale des triangles. Autrement dit, on cherche le maillage approximant le mieux, comportant le moins de sommets et n'ayant pas de triangles dégénérés. Pour minimiser l'énergie, on part du maillage initial et on effectue des modifications permettant de diminuer la fonction d'énergie, en optimisant dans premier temps le positionnement des sommets (optimisation continue) puis en optimisant sur l'espace des complexes (optimisation discrète) la façon de mailler ces points. Dans cette deuxième phase, il introduit le concept de modification incrémentale d'un maillage par application d'un opérateur topologique élémentaire, et liste les différents opérateurs : la contraction d'arête (*edge collapse*), le basculement d'arête (*edge swap*) et le partage d'arête (*edge split*). Ces notions sont fondamentales car toutes les méthodes de simplification peuvent être décrites par applications de ces opérateurs. Par exemple, la suppresion de sommet n'est qu'une contraction d'arête suivie d'un ou plusieurs basculements.

En 1996, Hoppe, désormais chez Microsoft Research, publiera à *Siggraph* un autre papier clef [Hop96]. Il y introduit la notion de maillage progressif. Un maillage est simplifé par applications successives de contractions d'arêtes jusqu'à un maillage très simple. En appliquant l'opération inverse de la contraction, l'éclatement de sommet (*vertex split*), à ce maillage de base, on reconstruit *progressivement* le maillage complet. Il est ainsi possible d'encoder un maillage sous

une forme qui permet la simplification, la transmission progressive (le client recoit initialement un premier modèle grossier et au fur et à mesure qu'il reçoit des éclatements de sommets, le modèle est raffiné), la compression (les modifications sont locales et un codage prédictif marche assez bien) et le raffinement sélectif (en n'effectuant les éclatements que sur les sommets dans une région d'intérêt par exemple). En outre, cette représentation permet une transition continue entre le modèle le plus simple et le plus compliqué : entre une arête « contractée » et une arête « éclatée », on peut construire un géomorphe continu en écartant progressivement les sommets de l'arête. Lorsqu'une arête est contractée, la position du sommet résultant est choisie en minimisant une fonction d'énergie comme dans [HDD+93]. La même année, Cohen *et al* publient une autre méthode procédant par enlèvement de sommets [CVM+96]. Le problème étant de contrôler et idéalement de borner la déformation causée par la simplification, les auteurs proposent une méthode pour construire des volumes englobant, les *Simplification Envelopes*, pour guider le choix des sommets que l'on peut enlever. Cette méthode est plus un cadre de travail pour contrôler et borner l'erreur, et peut être utilisée avec n'importe quel schéma de réduction, même si les auteurs en décrivent deux en particulier.

L'année 1997 est une année faste pour le domaine. Tout d'abord, Garland et Heckbert présente *Surface Simplification Using Quadric Error Metrics* [GH97]. Ce papier généralise la contraction d'arête en introduisant la contraction de paires arbitraires de sommets. Pour ne pas considérer les n^2 paires potentielles de sommets, une première passe est effectuée pour déterminer les paires à traiter. Les arêtes initiales et les sommets géométriquement proches sont retenus. Les auteurs décrivent ensuite une métrique simple et efficace pour déterminer le coût lié à la contraction d'une paire, c'est-à-dire pour déterminer dans quel ordre les contractions doivent être effectuées. En outre, cette métrique permet de déterminer la position du sommet remplaçant une paire en minimisant le coût. À chaque sommet est associée une *quadrique*, qui mesure la distance du sommet au maillage initial. Pour les sommets originaux, cette quadrique évalue la distance aux plans incidents. Quand deux sommets sont contractés, le nouveau sommet reçoit comme quadrique la somme de deux quadriques. L'intérêt de cette métrique est qu'elle est simple à stocker, à évaluer, et comme on vient de le voir à propager au long de la simplification. L'algorithme de Garland et Heckbert est très rapide, et va devenir la référence pour la simplification, notamment parce qu'une implémentation est disponible.

Toujours en 1997, Popović et Hoppe publient une généralisation des travaux de 1996 avec les *Progressive Simplicial Complexes* [PH97]. Sur le même principe que les *Progressive Meshes* mais, suivant la même voie que Garland et Heckbert, ils contractent cette fois n'importe quelle paire de sommets. Ils proposent également de rendre les points et les lignes obtenus par simplification comme des sphères et des cylindres en traquant au long de la simplification l'aire des triangles simplifiés. Contrairement aux PM où le maillage de base avait la même topologie que le maillage initial, le maillage de base d'un PSC est toujours une sphère, et des surfaces de n'importe quelle topologie, non nécessairement régulières, peuvent être simplifiées. Parallèlement, Hoppe présente un algorithme de raffinement sélectif

dépendant du point de vue, basé sur les *Progressive Meshes* [Hop97]. Des travaux proches avaient été présentés l'année précédente par Xia et Varshney [XV96]. De son côté, Schroeder étend ses travaux de 1992 en montrant comment coder progressivement la décimation, et comment gérer le changement de topologie [Sch97b]. Pour cela, il ne retriangule plus le trou résultant de l'enlèvement d'un sommet mais comble le trou en effectuant un basculement d'arête. Ce faisant, il rend possible la fermeture de trous initialement présents dans le modèle, et le rattachement de deux variétés. En outre, lorsqu'il ne peut plus enlever de sommets (parce qu'il ne reste plus de sommets simples, ou que les basculements d'arêtes sont impossibles) et que le taux de simplification n'est pas atteint, il autorise le « déchirement » du maillage le long des arêtes vives, des coins et des sommets non simples.

Enfin, dans cette même année, Cohen, Manocha et Olano publient leurs travaux sur les *Successive Mappings* [CMO97]. Les auteurs s'intéressent à la construction de bijections entre les différents niveaux de simplification. Le schéma de simplification est une contraction d'arête. Le voisinage de chaque arête est projeté sur un plan. L'arête est contractée dans ce plan, et une bijection est construite entre ces deux maillages locaux. Pour que cette bijection existe, le plan de projection, et la position du sommet remplaçant l'arête doivent être correctement choisis, garantissant notamment qu'il n'y aura pas localement d'auto-intersection dans la surface simplifiée. L'algorithme décrit par les auteurs est générique et peut en particulier être utile aux algorithmes d'enlèvement de sommet (qui ont besoin d'un plan de projection pour retrianguler les trous causés par la suppression d'un sommet). La bijection est utilisée pour « remonter » en trois dimensions le sommet en lequel la projection de l'arête a été contractée. C'est-à-dire placer ce point le long de la direction de projection. Grâce à la bijection, ce point peut être placé de manière à minimiser la distance entre tout point de la surface et son image par la bijection, c'est-à-dire son équivalent sur la surface simplifiée. Il est ainsi possible de contrôler, comme avec les *Simplification Envelopes*, la distance entre les surfaces dans la direction de la normale à la surface, mais également, comme le font remarquer les auteurs, d'éviter que la surface ne glisse sur elle-même lors de la simplification. Cela permet notamment une meilleure préservation des attributs comme par exemple les coordonnées de textures. Enfin, la distance ainsi définie constitue la métrique pour déterminer l'ordre de contraction des arêtes.

À partir de 1997, la contraction de paires de sommets s'impose comme la méthode de prédilection. Comme le dit Peter Lindstrom « [cet opérateur] *est très attractif car il permet de placer le nouveau sommet d'une manière qui aide à préserver la position et la forme de la surface. C'est aussi une opération plus atomique que l'enlèvement de sommet et ça ne nécessite pas d'invoquer un algorithme de triangulation* ». Si le formalisme des PM et des PSC et l'approche par minimisation d'une fonction d'énergie sont très élégants, ils restent cependant compliqués à comprendre, difficiles à implémenter et surtout assez lents à éxécuter. À l'inverse, les quadriques d'erreurs sont très simples à comprendre et à manipuler : une fonction d'erreur est attachée à chaque sommet, qui permet d'évaluer le coût d'une contraction de paire et de placer le point résultant d'une contraction de manière à

minimiser ce coût. Les paires sont contractées dans l'ordre de coût croissant. Ces méthodes affichent en outre des temps de calculs compétitifs. Plusieurs travaux vont améliorer leur utilisation. Lindstrom en 1998 montre qu'en introduisant l'aire des faces des triangles dans le calcul des quadriques, on minimise non seulement le déplacement du sommet, mais aussi la variation de volume [LT98] et montre qu'on peut ainsi obtenir une déviation géométrique comparable à celle obtenue par optimisation de maillage [HDD+93] en étant plus rapide de plusieurs ordres de grandeur. En 1999, Erikson et Manocha apportent avec GAPS deux autres améliorations [EM99]. Dans la contraction de paire de sommets, un problème est de déterminer les paires à considérer. En général, l'utilisateur spécifie un « rayon de proximité ». Les auteurs proposent un schéma adaptatif qui détermine ce rayon automatiquement et le fait évoluer au cours de la simplification pour s'adapter à la partie de la scène restant à simplifier. En outre, ils montrent qu'il n'est pas optimal de contracter les paires dans l'ordre de coût croissant. En ajoutant un critère de préservation d'aire, on évite la disparition de triangles isolés en favorisant d'abord le rattachement de tels triangles.

Le coût mémoire d'un algorithme de simplification peut parfois poser problème. Non seulement le modèle doit être chargé en mémoire, mais en plus une information de coût, la quadrique, est attachée à chaque sommet pour traçer l'historique des erreurs commises sur ce sommet. Lindstrom montre qu'on peut s'affranchir de cette contrainte et recalculer le coût à chaque étape [LT98]. Il baptise sa méthode *Memory efficient*. Mais le modèle doit toujours être chargé en mémoire. Cette contrainte est abolie peu de temps après [Lin00]. Avec *Out-of-Core Simplification of Large Polygonal Models*, Lindstrom et Turk présentent une avancée majeure. La méthode est similaire à un algorithme de clusterisation, mais en utilisant la quadrique d'erreur, il n'est plus nécessaire d'avoir le modèle en mémoire. Celui-ci est parcouru une seule fois, par exemple sur disque. Pour un triangle, les clusters de trois sommets sont déterminés, le triangle est ajouté au résultat s'il est non dégénéré (différant en cela des autres algorithmes de clusterisation), et une quadrique est calculée pour ce triangle et ajoutée aux quadriques stockées dans chaque cluster. Quand tous les triangles ont été parcourus, la quadrique d'un cluster est utilisée pour déterminer le placement de son représentant. Seule la simplification est donc stockée en mémoire. Avec Silva ils montreront que cela n'a pas forcément besoin d'être en mémoire principale [LS01]. Le disque peut-être efficacement utilisé moyennant un codage intelligent. Dans ce papier, les auteurs montrent également comment préserver les frontières en utilisant une astucieuse « quadrique d'arête » qui mesure le déplacement d'un sommet par rapport à la frontière, sans avoir besoin d'identifier explicitement ces frontières. Et donc sans avoir à construire de structure d'adjacence en mémoire.

Pour terminer sur les méthodes incrémentales, citons les travaux de Velho sur les clusters à quatre faces [Vel01]. Kobbelt, Campagna et Seidel ont également présenté une très intéressante analyse de la décimation de maillage [KCS98], qui replace le problème dans le contexte d'une optimisation gloutonne et isole les différents degrés de liberté d'un algorithme (opérateurs, mesures, prédicats et oracles

à spécifier). Les auteurs affirment dans ce papier que le placement du sommet lors d'une contraction n'a pas d'importance et qu'il suffit de choisir une des deux extrémités (ce qu'on peut formaliser comme la contraction d'une demi-arête). Nous ne partageons pas leur opinion au vu des travaux postérieurs, notamment ceux de Lindstrom *et al*. Nous conseillerons plutôt la lecture des travaux de Southern, Marais et Blake [SMB01]. Les auteurs y présentent une bonne analyse des différentes méthodes existantes et proposent un algorithme générique, capable d'utiliser différentes métriques au cours de la simplification. Les auteurs montrent en outre comment réduire le coût mémoire lié aux resources qu'il faut stocker par sommet pour évaluer ces métriques et baptisent leur algorithme *memoryless* pour reprendre la terminologie introduite par Lindstrom [LT98].

2.1.4 Autres approches

Reconstruction par approximation

Toutes les méthodes de simplification ne procèdent pas par altérations locales et successives de la géométrie. On a déjà vu [Tur92] qui reconstruit une version simplifiée de la surface à partir de nouveaux points, même si pour cela il procède par décimation. Précédemment DeHaemer et Zyda proposaient un algorithme de simplification par reconstruction [DJZ91]. Se basant sur les travaux de Schmitt [SBhD86], ils reconstruisent un maillage en ajustant itérativement des polygones à un ensemble de points, de telle sorte qu'aucun point ne soit à plus d'une distance seuil donnée d'un polygone. Une approche qui ressemble aux méthodes de clustering est proposée par He, Hong, Kaufman, Varshney et Wang [HHK+95]. La surface est filtrée dans une grille par convolution. Chaque voxel contient donc une « densité » qui indique la quantité de surface dans la grille. Une surface simplifiée est alors reconstruite par un algorithme de *marching cubes*. En faisant varier l'iso-densité utilisée par cet algorithme, les auteurs construisent plusieurs surfaces auxquelles sont attribuées des transparences adaptées pour réduire les effet d'aliassage lors du rendu. On trouve également les travaux de Eck *et al* [EDD+95]. Les auteurs décrivent une méthode permettant de générer, à partir d'un maillage quelconque, un maillage ayant la propriété de continuité de subdivision. On peut alors, en appliquant les travaux de Lounsbery [Lou93], coder la surface par ondelettes et naturellement extraire différents niveaux de détails en filtrant les coefficients.

Clusterisation de faces

La simplification d'un maillage n'est pas forcément explicite. On entend par là qu'il n'est pas forcément nécessaire de reconstruire une version simplifiée, mais qu'il peut suffire d'identifier des composantes de différentes résolutions dans le maillage. À ce titre, les méthodes par clusterisation de sommets que nous avons vues à la section 2.1.2 identifient des groupes de sommets (les clusters) qui peuvent être considérés comme un seul élément (le point représentant) dans certaines conditions (quand le cluster est vu de loin par exemple). Dans le même esprit, on peut

chercher à regrouper les polygones par clusters de faces qui permettront d'approximer un groupe de faces par une forme simple (planaire) par exemple dans un calcul de radiosité, ou de collision. Évidemment ces regroupements peuvent être utilisés pour simplifier le modèle et c'est en général l'application qui est décrite. Paul Hinker et Charles Hinsen proposent ainsi de grouper les polygones adjacents dont la normale est proche en régions coplanaires et de retrianguler la bordure de ces « groupes de faces » [HH93]. Avec *Superfaces*, Kalvin et Taylor utilisent une approche similaire [KT96]. Les groupes de faces sont itérativement construits par extension à partir d'une face « graine ». L'extension utilise cette fois des critères plus évolués que la simple comparaison des normales. Le ratio d'apect des triangles, une borne d'erreur et des tests pour préserver la topologie sont appliqués. Une autre contribution est un schéma d'étirement des bordures des superfaces. En effet, comme elles épousent les bords des polygones, ces frontières ont de grandes chances d'être en dents de scie. Un schéma de subdivision est appliqué pour les remplacer par des bordures polygonales plus simples. Les groupes de faces ainsi délimitées par ces nouvelles frontières sont ensuite triangulés. Sheffer propose également un algorithme de regroupement de face [She00] basé sur la construction d'un graphe dual des faces dont les arcs sont itérativement contractés. Garland, Willmott et Heckbert publient une méthode similaire [GWH01] qui utilise l'erreur quadrique pour déterminer dans quel ordre effectuer ces contractions, et construit une hiérarchie de groupe de faces. De leur côté, Brodsky et Watson posent le problème de la simplification comme un problème de quantification vectorielle et en déduisent une méthode de clusterisation de faces en fonction de la proximité des normales [BW00]. Signalons pour finir que la clusterisation de faces permet de reparamétriser une surface dans le but d'en faire une subdivision hiérarchique et de répresenter l'éclairage sur la surface [Tur02].

2.1.5 La gestion des attributs

Comme nous l'avons dit en introduction, la gestion des attributs est un problème pour la simplification, qui s'intéresse initialement, et historiquement, à la simplification de la *forme* du maillage en supprimant les points et les faces qui y contribuent le moins. Or un certain nombre des maillages que nous considérons, notamment les maillages produits par les graphistes, possèdent des attributs attachés aux sommets et aux faces du maillage. Certains sommets d'un maillage sont ainsi introduits non pas pour spécifier un détail de forme, mais pour indiquer un changement d'attribut. En supprimant un tel sommet, on peut altérer grandement l'apparence d'un modèle. En outre, les attributs spécifiés aux sommets sont généralement interpolés sur la face, et le déplacement d'un sommet, à travers la modification de la forme des faces, peut influer grandement sur ce mécanisme.

Deux approches ont été proposées pour s'attaquer à ce problème. La première consiste à intégrer les attributs dans la métrique utilisée pour déterminer l'ordre de simplification et le placement des nouveaux sommets. Garland et Heckbert étendent ainsi leur quadrique [GH98], suivis l'année d'après par Hoppe [Hop99],

qui proposait déjà dans les *Progressive Meshes* une façon de prendre en compte les attributs [Hop96]. De leur côté, Certain *et al* étendent les travaux de Eck et De Rose pour gérer les couleurs [CPD+96]. Dans ces approches, l'idée consiste à augmenter le coût des arêtes représentant une discontinuité dans les attributs, et lorsque ces arêtes sont simplifiées, à aligner les nouvelles arêtes le long des discontinuités. Dans le cas particulier des coordonnées textures, lorsqu'un nouveau sommet est introduit, il est placé, et ses coordonnées textures sont choisies, de manière à minimiser le « glissement » de la texture sur la surface. L'inconvénient de ces approches, comme le font remarquer Garland et Heckbert, est qu'elles ne marchent bien que si les discontinuités ne sont pas très nombreuses. Un maillage où chaque face a une couleur et une texture différente ne sera pas correctement traité.

À l'opposé de ces approches, Cignoni, Montani, Rocchini et Scopigno attaquent le problème de manière orthogonale [CMSR98]. Ils n'interviennent pas durant le processus de simplification, mais une fois le maillage simplifié. Une texture est alors générée, qui habillera le maillage pour restituer l'apparence spécifiée par les attributs. La méthode consiste à convertir un triangle du maillage basse résolution en points échantillons sur le maillage haute résolution, par un algorithme de *scan-conversion*. Les attributs en ces points échantillons sont utilisés pour construire une texture triangulaire qui sera associée au triangle basse-résolution. Pour être efficace, ces textures triangulaires sont regroupées dans une seule texture en essayant de minimiser la taille de cette texture (problème d'arrangements de triangles dans le plan). L'intérêt de cette approche déjà proposée par [Mar95, SGR96] est que la texture peut-être utilisée pour encoder n'importe quelle information scalaire ; les attributs d'une part, mais aussi un champ de normales par exemple, ou une carte de relief. Ces informations, combinées avec des algorithmes utilisant astucieusement les possibilités textures des cartes graphiques permettent d'augmenter la qualité visuelle des niveaux de détails (*bump mapping*, *relief textures* [Bli78, Coo84, RE00, OBM00]).

La même année, Cohen, Olano et Manocha présentent à *Siggraph* une approche identique [COM98], mais plus complète et qui fait référence sur le sujet. Ils définissent une métrique de déviation pour contrôler l'erreur commise dans le placement de la texture. Les auteurs proposent également de guider la phase de simplification géométrique pour optimiser la phase, pourtant décorrélée, de préservation des attributs.

Signalons enfin un algorithme qui lui aussi intervient après la phase de simplification, et cherche à augmenter la qualité du rendu des modèles simplifiés. Il s'agit de *Silhouette Clipping* de Sander, Gu, Gortler, Hoppe et Snyder [SGG+00]. Cet algorithme est motivé par le fait que la silhouette d'un objet est un élément déterminant de son apparence. Or lorsqu'un modèle est simplifié, sa silhouette se trouve assez grossièrement polygonalisée. Imaginez par exemple une sphère finement tesselée : sa silhouette est un cercle. Lorsque la sphère est simplifiée, sa silouhette devient un polygone qui saute aux yeux lors du rendu. Les auteurs proposent intelligement de rendre une tel modèle de manière simplifiée mais de *clipper* sa silhouette en utilisant la version haute résolution du modèle. Cette dernière peut en

effet être calculée de manière rapide, grévant donc peu le coût de rendu du modèle simplifié, et le *clipping* peut être fait efficacement en utilisant un masque stocké dans le *stencil buffer*. Pour que l'algorithme fonctionne, il faut que le modèle simplifié soit « plus large » que le modèle non simplifié. Cette méthode, très élégante ne gère cependant pas complètement ni simplement les silhouettes internes de l'objet.

2.1.6 Contrôle de l'erreur

Pour terminer cette présentation des papiers de simplification qui ont inspiré et guidé les travaux que nous présentons dans cette thèse, voici quelques publications concernant le contrôle d'erreur. Nous avons vu que les *Simplification Envelopes* de Cohen *et al* peuvent être employées avec différents schémas de simplification pour garantir que le modèle simplifié se trouve à une certaine distance du modèle original. Ces enveloppes ont cependant le désavantage de ne pas être triviales à calculer. Récemment, Steve Zelinka et Michael Garland ont proposé les *Permissions Grids* [ZG02], une voxelisation adaptative du modèle original qui peut être avantageusement et beaucoup plus simplement utilisée pour arriver aux mêmes fins.

De leur côté, Peter Lindstrom et Greg Turk proposent une approche tout à fait novatrice [LT00]. Ils introduisent dans l'évaluation du coût d'une simplification élémentaire une métrique *dans l'espace image* là où les méthodes classiques travaillent *dans l'espace objet*. Pour mesurer le coût d'une opération, ils comparent les images du modèle après application d'une simplification et celles du modèle sans simplification. Ils parlent alors d'*Image-Driven Simplification*. Un certain nombre de problèmes sont encore à régler (choix de la métrique dans l'espace objet, optimisation du rendu des images), mais l'approche est intéressante et ouvre de nouvelles voies.

2.2 Rendu à base d'image

Ces dernières années, un assez grand nombre de travaux de recherche ont aussi été consacrés à la modélisation et au rendu à base d'images. C'est même devenu un domaine de recherche en soi, désigné par le terme d'*Image Based Rendering*. Les chercheurs réunis sous cette bannière mettent en avant que les approches à base d'image devraient permettre un rendu de haute qualité de scènes très complexes, le tout en temps interactif. Bien qu'alléchante, cette affirmation doit être tempérée et même si de très intéressantes solutions purement à base d'images ont été proposées, la tendance actuelle est plutôt à l'utilisation d'images *en conjonction* avec les primitives polygonales traditionnelles.

2.2.1 Méthodes purement image

En 1996, Marc Levoy et Pat Hanrahan [LH96] et Gortler *et al* [GGSC96] ont introduit simultanément une nouvelle représentation. Appellée *lightfield* par Levoy

et *lumigraph* par Gortler, cette primitive est intéressante car elle constitue la première tentative pour reconstruire systématiquement un sous-ensemble conséquent de la fonction *plénoptique*. Introduite par Adelson et Bergen [AB91], cette fonction représente l'ensemble des rayons lumineux visibles en tout point de l'espace, au cours du temps et pour différentes longueurs d'ondes. C'est un formalisme qui permet d'asseoir les travaux de rendu à base d'image sur un socle théorique solide. Dans les travaux de Levoy ou de Gortler, cette fonction est échantillonnée en mettant de côté les aspects temporels, de longueur d'onde, et les phénomènes participatifs. En restreignant le point de vue à une région convexe et ne comportant pas d'obstacle, on peut considèrer que les rayons lumineux à représenter ont quatre degrés de liberté. C'est-à-dire qu'ils sont par exemple paramétrés par deux points sur deux plans parallèles, situés de chaque côté du modèle. Une façon de représenter ces rayons est alors de prendre une image du modèle avec une caméra placée sur des points échantillons du plan « avant » et regardant le plan « arrière » (le fait de prendre une image échantillonnne aussi ce plan à la résolution de l'image). Un *lightfield* ou un *lumigraph* est alors la collection de ces images, proprement encodée et compressée pour gérer le coût de stockage d'un aussi grand nombre d'images, et la donnée d'un *algorithme de reconstruction* permettant de générer de nouvelles vues à partir de cet échantillonnage de la fonction plénoptique. Dans leur papier, Gortler *et al* montre comment utiliser les fonctionnalités texture disponibles sur les cartes graphiques pour effectuer efficacement ce rendu. Une solution plus générale utilisant le matériel graphique suivra dans [BBM+01]. Pour déterminer le nombre d'images nécessaires pour pouvoir effectuer une bonne reconstruction (sans aliassage), Chai *et al* [CCST00] s'appuyent sur l'analyse de Fourier et sur deux hypothèses : les profondeurs maximale et minimale de la scène sont connues, et on considère une scène diffuse (sans effets spéculaires). Cette dernière hypothèse limite l'utilisation de leurs résultats.

Les *lightfields* ont donné lieu à de nombreuses publications, par exemple sur leur utilisation pour représenter des sources lumineuses complexes [HKS98], pour acquérir la reflectance d'un visage humain [DHT+00], ou encore pour intégrer des effets dynamiques de profondeur de champ [IMG00]. Malgré des travaux pour accélérer le rendu [SCG97, SHS00], ou proposer différents schémas d'échantillonnage ou différentes stratégies de compression, les coûts de stockage inhérents à la méthode ne la rendent envisageable en pratique que pour des objets modérément petits. En outre, toutes ces méthodes s'appuyent sur une reconstruction (*forward mapping*) des pixels, qui pour l'instant doit être effectuée de manière logicielle et rend les méthodes peu propices au temps réel pour des scènes complexes (comportant plusieurs dizaines d'objets modélisés chacun par des *lightfields*). Pour plus de détails sur le *warping*, le lecteur pourrra consulter l'ouvrage de Gomes *et al* [GDCV98].

Une autre solution, proposée par Chen [Che95] et popularisée par une implémentation largement distribuée - Quicktime VR - consiste à n'autoriser l'observateur à occuper qu'un ensemble discret de positions. Pour chacune de ces positions un panorama complet a été obtenu, c'est-à-dire que l'on a reconstruit une sous par-

tie de la fonction plénoptique en fixant le point de vue et en considérant toutes les directions d'observation (même si en pratique on limite souvent le panorama à un cylindre). L'utilisateur peut donc « tourner continument » la tête mais nes pourra que « sauter » de point de vue en point de vue.

2.2.2 Méthodes hybrides

Dans toutes ces méthodes, l'idée est d'utiliser les images comme approximations du modèle. La forme la plus simple, et la plus révélatrice des avantages qu'offre la représentation par image est l'application de *textures* [Hec89] à des maillages. Cela permet de simuler du détail sur une surface plate offrant ainsi une grande richesse visuelle. Le *mip-mapping* [Wil83], le filtrage bilinéaire (et plus récemment anisotrope) disponibles sur les cartes graphiques permettent, quand une texture est projetée sur une image, de corriger les effets d'aliassage dus à la distorsion perspective. Si les détails que la texture permet d'ajouter à la surface d'un polygone ont d'abord été des détails de couleurs, la versatilité de la représentation par texture a vite été exploitée pour rajouter des détails de forme (petites perturbation locales de la normale à la surface à un polygone [Bli78]) ou d'éclairage (*environnement mapping* [Gre86] et *shadow mapping* [SKvW+92]).

La texture, rapide à rendre sur toutes les cartes graphiques modernes, et offrant une grande richesse visuelle, est donc utilisée pour enrichir un modèle polygonal, de manière statique. Mais ces qualités peuvent aussi être exploitées pour remplacer dynamiquement des parties d'un modèle (comportant des polygones et aussi des textures). C'est Maciel et Shirley qui ont ainsi introduit le concept *d'imposteur* [MS95]. Une image d'un objet est utilisée à la place de l'objet lui-même en texturant un polygone avec l'image, à la manière d'un décor de western. Shade *et al* [SLS+96] et Schaufler *et al* [SS96] ont développé simultanément un système de cache dynamique exploitant le principe des imposteurs. Une structure hiérarchique est utilisée pour représenter la scène, et certains noeuds de la structure, c'est-à-dire des parties de la scène sont remplacées dynamiquement par des imposteurs. Une métrique d'erreur permet d'évaluer la distorsion entre l'imposteur et la géométrie remplacée lorsque l'utilisateur se déplace. Quand la distorsion est supérieure à un seuil fixé, les imposteurs sont mis à jour. La méthode suppose donc une grande cohérence temporelle, car des changements brusques de point de vue invalident tous les imposteurs et leur mise à jour forcée ralentit le système. En outre ces méthodes requièrent de pouvoir transférer dynamiquement et très rapidement des images dans une texture, ce qui n'est pas encore disponible sur la plupart des cartes graphiques.

D'autres auteurs ont proposé d'utiliser les imposteurs pour simplifier certaines parties d'un modèle. Aliaga et Lastra proposent ainsi de remplacer les parties d'un modèle visible à travers un hublot par des textures générées au vol [AL97], ou en combinant deux textures parmi un ensemble de textures pré-calculées. Snyder et Lengyel [LS97, SL98] généralisent l'utilisation des *sprites*, version simplifiée des imposteurs utilisée par l'industrie du jeu pour représenter et composer les diffé-

rents éléments d'une scène, et montrent comment combiner correctement plusieurs *sprites* pour représenter différentes parties de plusieurs objets en mouvement. Leurs travaux s'inscrivent dans une tentative d'implémentation matérielle d'une architecture de rendu axée sur les sprites, baptisée *Talisman* [TK96].

Paul Debevec *et al* [DTM96] ont proposé un système de rendu de bâtiments architecturaux mixant représentation polygonale et rendu à base d'image. Leur papier introduit la notion de *modélisation* à partir d'image. La reconstruction d'une forme polygonale, ensuite utilisée par leur système de rendu pour combiner les différente images du modèle, est faite de manière semi-automatique. Des techniques de reconstruction sont utilisées pour retrouver les coordonnées dans l'espace des sommets de primitives simples (cube, sphère, cylindre), dont l'utilisateur indique la projection dans différentes images. Cette idée d'utiliser l'image comme primitive de reconstruction d'un modèle tri-dimensionnelle se retrouve dans les travaux de Horry *et al* [HAA97]. Oh *et al* ont plus tard étendu ce principe et proposent un système [OCDD01] complet de modélisation et d'édition à partir de photo, intégrant notamment la prise en compte des effets d'éclairage.

L'imposteur dans sa version initiale décrit un objet par une image et un polygone plat, ignorant donc toute information de profondeur. Les effets de parallaxe qui peuvent être restitués de cette manière sont donc limités. On peut au contraire décider de conserver, ou de synthétiser, une information de profondeur par pixel de l'image et utiliser cette information pour restituer des effets de parallaxes complexes. L'opération de base dans une telle approche consiste à reprojeter les pixels dans les nouvelles vues que l'on génère en utilisant cette information de profondeur et la position de la caméra. Cela fut proposé par exemple par Chen [CW93] et par MacMillan [MB95], ce dernier décrivant notamment dans quel ordre les pixels d'une image doivent être reprojetés pour restituer correctement les occlusions. La difficulté dans ces approches est que deux pixels contigus dans une texture mais ayant des profondeurs associées différentes peuvent se reprojeter à plusieurs pixels d'écart dans une nouvelle vue, laissant voir un « trou ». Ce trou vient du fait que l'information correspondant à ce que l'on devrait voir entre les deux pixels quand on change le point de vue n'est pas présente dans l'image initiale (les pixels y sont adjacents). Une solution consiste alors à utiliser plusieurs images afin que l'information non présente dans une image puisse être retrouvée dans les autres. Cela implique à priori de stocker plusieurs images par imposteurs, mais une solution élégante proposée par Schade [SGHS98] et baptisée *Layered Depth Images* consiste à stocker plusieurs « couches » de profondeur par pixel dans une unique image. Ces LDI peuvent être obtenues par lancer de rayon ou en reprojetant plusieurs images dans une vue commune. Chang, Bishop et Lastra proposent une extension multi-résolution des LDI [CBL99].

De leur côté, Bishop et Oliveira proposent un algorithme, théoriquement implémentable comme une fonctionnalité matérielle des cartes graphiques, pour habiller des polygones avec une texture *en relief* [OBM00]. Ils réécrivent astucieusement la formule de reprojection de McMillan pour la factoriser en une passe 1D qui peut être effectuée très efficacement et une passe qui est effectuée implicitement par le

mécanisme de *texture mapping*.

Plutôt que d'attacher une profondeur à chaque pixel, on peut utiliser plusieurs couches de pixels superposées et partiellement transparentes, rendues en exploitant les accélérations des cartes graphiques. Schaufler [Sch98] proposa ainsi une méthode pour « trancher » ainsi un objet en plusieurs couches représentant chacune un petit intervalle de profondeur, et rendre ces couches efficacement. Une approche similaire a été développée par Meyer et Neyret pour visualiser des textures volumiques complexes [MN98].

Une dernière alternative pour restituer plus de parallaxe qu'avec une simple image sur un polygone plat consiste à plaquer l'image sur un maillage en relief. Une solution pour ce faire consiste à transformer l'image (et sa carte de profondeur associée, ce qu'on appelle généralement une *range image*) en un maillage en créant des micro-triangles pour chaque pixel [MMB97]. Sillion *et al* proposent plutôt d'utiliser des techniques de vision pour construire un maillage qui épouse le relief de la carte de profondeur. La difficulté de ces approches est double : d'une part le nombre de faces du maillage est grand, d'autre part il se produit un artefact dit *de triangles élastiques* (en anglais, *rubber sheet triangles*) : le maillage forme une nappe continue et relie entre eux des éléments éventuellement disjoints dans le modèle. À l'inverse des trous qui apparaissent dans les reprojections de pixels, on a ici des triangles étirés qui bouchent la vue. Mark *et al* [MMB97] proposaient quant à eux une heuristique pour détecter les endroits où cela peut se produire. De leur côté, Décoret *et al* [DSSD99] utilisent une information de haut niveau sur le modèle pour répartir les différents objets du modèle sur plusieurs imposteurs et reduire ainsi les artefacts.

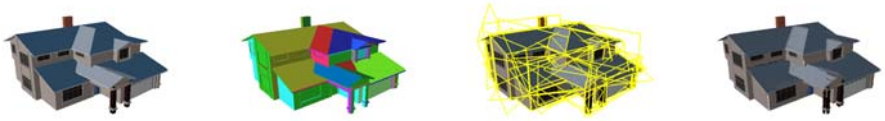
Oublions le mot « vérité », privilégions le mot « authenticité »

Albert Jacquard

3

Simplification de modèles polygonaux

Dans ce chapitre, nous décrivons une nouvelle représentation d'un modèle 3D à partir de polygones texturés avec de la transparence. Cette représentation peut être utilisée à différentes fins, dont la simplification de modèle. Outre l'introduction de cette nouvelle représentation, la contribution de ce chapitre est un algorithme automatique de construction à partir d'un modèle polygonal quelconque, par exemple la maison *(a)* ci-dessous. L'enjeu de cet algorithme est de déterminer un ensemble de plans dominants dans un modèle (en fausse couleur sur la figure *(b)*), puis de simplifier les faces du modèle en les projetant sur ces plans pour générer des textures *(c)*, et d'obtenir ainsi un modèle simplifié *(d)* visuellement très semblable au modèle original. Un plan dominant n'est pas nécessairement un plan du modèle original ; pour une façade richement détaillée, il s'agira par exemple du plan moyen des fenêtres, du mur et des divers ornements.



(a) (b) (c) (d)

3.1 Les nuages de *Billboards*

Une classe de modèles qui résume bien les limitations de méthodes actuelles de simplification est celle des arbres. Un arbre est un objet de très grande complexité géométrique (centaines de milliers de feuilles de très petite taille, silhouette riche en détails) et d'apparence compliquée (richesse des couleurs, possibilité de

voir à travers le feuillage, interaction complexe avec la lumière). En outre, un arbre est rarement seul et un modèle un tant soit peu réaliste d'environnement extérieur comporte facilement plusieurs centaines d'arbres de caractéristiques différentes. En conséquence, d'une part le budget disponible par arbre est grandement limité et d'autre part, il faut pouvoir gérer une grande variété de modèles, car l'instanciation d'un petit nombre d'arbres pour créer par exemple une forêt, produit une sensation de répétition très désagréable pour l'œil humain. Ces contraintes rendent les méthodes de simplification de maillage décrites au chapitre 2 assez inadaptées. Il faut en effet travailler à un niveau de simplification extrême (quelques dizaines de polygones par arbre), et pouvoir regrouper un grand nombre d'objets en une entité simple, par exemple une forêt située au lointain. Or non seulement chaque modèle a une topologie dégénérée (les arbres produits par AMAP[1] par exemple sont des collections de triangles non connectés et avec un grand nombre d'intersections), mais en plus les différents objets s'interpénètrent de façon arbitraire.

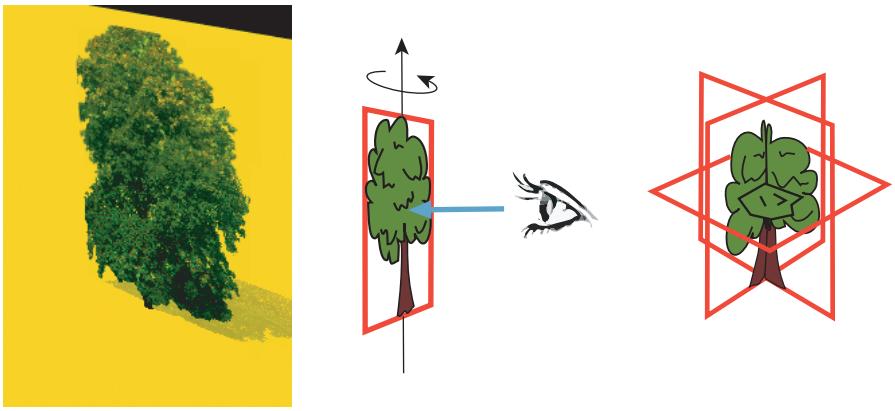
Pour les arbres, l'industrie du la simulation a adopté une solution simple et efficace : les *billboards* [RH94]. En français, ce terme pourrait se traduire par affiche ou poster mais il perdrait alors la connotation qu'a le mot anglais dans la communauté graphique. Nous aurions pu utiliser le terme d'*imposteurs*, mais ce terme a déjà un sens particulier dans la littérature d'informatique graphique (voir section 2.2), causant déjà parfois des confusions. Nous lui préférerons donc le terme anglais.

Un *billboard* est un rectangle texturé avec de la transparence comme montré sur la Figure 3.1(a). Vu de coté, un *billboard* est évidemment plat et la parallaxe fait qu'il ne représente pas du tout correctement l'objet qu'il remplace. Pour éviter cela, il existe deux approches. Soit on maintient le plan du *billboard* perpendiculaire à la direction d'observation (voir Figure 3.1(b) : le *billboard* « tourne » pour toujours faire face à l'observateur [Gra]), soit on place quelques *billboards* « en croix » (Figure 3.1(c)). Les *billboards* sont donc bien adaptés aux objets ayant une symétrie de révolution, ce qui est notamment le cas des arbres.

Les *billboards* sont en pratique relativement efficaces et certains jeux vidéos, comme *Black & White* de LionHead Studios[2], arrivent à des résultats assez saisissants. Malheureusement, leur usage est limité. En effet, si le placement en croix est trivial pour un objet ayant une forte symmétrie de révolution autour d'un axe, il n'en est pas de même pour un objet quelconque. De plus, il n'y a aucun contrôle sur le nombre de plans. Combien en faut-il au minimum ? Si deux est le nombre généralement retenu, sait-on ce qu'on gagnerait à en utiliser plutôt trois ? Une fois les *billboards* placés, comment génère-t-on les textures qui doivent les habiller ? Ces questions sont sans réponse car, en pratique, les *billboards* sont placés et fabriqués « à la main » par les artistes.

Dans cette thèse, nous proposons de généraliser l'usage des *billboards*, en considérant un nombre arbitraire de *billboards* enchevêtrés de manière complexe.

[1] `www.bionatics.com`
[2] `www.bwgame.com`



(a) (b) (c)

FIG. 3.1 – Un *billboard* d'arbre

La Figure 3.2 montre un exemple de ce que nous appellerons *un nuage de billboards*. Les raisons pour lesquelles nous proposons cette nouvelle représentation

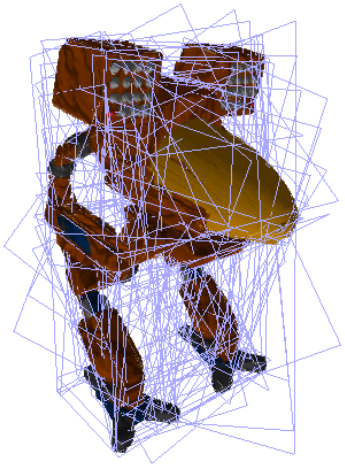


FIG. 3.2 – Un nuage de *billboards*

sont discutées dans la Section 3.2. Nous proposons ensuite un algorithme pour générer *automatiquement* cette représentation à partir d'un modèle *quelconque*. Dans la Section 3.8, nous présentons les résultats obtenus et discutons des applications possibles.

3.2 Motivations

L'utilisation de textures permet de coder efficacement des variations d'apparence sans avoir besoin d'introduire des sommets artificiels. Rappelons aussi que cela permet de filtrer l'information et donc de réduire les effets d'aliassage. En y adjoignant la transparence, la texture permet également de représenter une partie de l'information de forme. En effet, elle permet d'obtenir une silhouette de complexité arbitraire (limitée par la résolution de la texture) à un coût fixe. De son côté, l'utilisation de rectangles supports tri-dimensionnels permet de représenter, grâce au mécanisme de projection et d'interpolation des cartes graphiques, la forme de l'objet. Plus précisément, en « plaçant » les textures correctement dans l'espace les unes par rapport aux autres, on peut restituer les effets de parallaxe et d'occlusion liés à la forme tri-dimensionnelle de l'objet.

3.3 Algorithme de construction

Le problème à résoudre est une optimisation : étant donné un ensemble de faces représentant un modèle, construire un ensemble de plans et de textures associées qui approxime au mieux ces faces. Pour quantifier l'approximation, on introduit classiquement une fonction d'erreur et une fonction de coût. La « meilleure » approximation dépend alors de l'objectif que l'on se donne. Nous distinguons deux approches :

approche orientée budget dans laquelle le coût maximum autorisé est fixé et où l'on cherche une approximation minimisant l'erreur ;

approche orientée erreur dans laquelle l'erreur maximale autorisée est fixée et où l'on cherche une approximation minimisant le coût.

Dans la première approche, l'objectif est de rendre le mieux possible le modèle dans le temps imparti. L'approximation la plus violente consiste à ne pas rendre un élément lorsque le budget alloué est dépassé. On préfère cependant rendre quelque chose pour cet élément, quitte à ce que ce soit très approximatif. L'algorithme sert donc à construire la meilleure approximation qui permette de satisfaire la contrainte budgétaire.

Dans la deuxième approche, on cherche au contraire à optimiser la façon dont est représenté le modèle. L'erreur fixée définit d'une certaine manière un filtrage sur la résolution des détails conservés dans l'approximation. L'approximation supprime l'information dont la résolution est inférieure à l'erreur autorisée. Il est à noter que l'approximation, non seulement réduit le coût d'extraction de l'information, mais permet aussi de réduire l'aliassage et donc d'augmenter la qualité finale.

C'est dans la deuxième approche (erreur fixée) que nous nous plaçons pour décrire l'algorithme. Nous verrons à la fin que le formalisme et les outils introduits permettent aussi de travailler à coût fixé.

3.3.1 Vue d'ensemble

Pour résoudre le problème d'optimisation décrit ci-dessus, il faut définir les fonctions d'erreur et de coût associées au schéma de simplification utilisé. Puisqu'on travaille à erreur fixée, il faut aussi définir quelles sont les simplifications qui respectent la borne d'erreur. Ces points sont abordés aux sections 3.3.2 et 3.3.3. Le problème à résoudre étant NP-complet, comme nous le verrons à la section 3.3.4, nous choississons une stratégie gloutonne : à chaque itération, un plan est choisi qui permet d'approximer le maximum de faces, tout en respectant la borne d'erreur. Pour mettre en œuvre cette stratégie, il faut :

Considérer l'ensemble des plans. Pour cela, nous introduisons à la section 3.4 une paramétrisation et une discrétisation de l'espace des plans.

Mesurer l'intérêt des plans. La section 3.5 explicite une fonction dite *de densité* que nous utilisons pour trouver, à chaque itération de l'algorithme glouton, le « meilleur » plan pour simplifier ce qui ne l'a pas encore été.

Déterminer un ensemble de plans. Muni de la discrétisation de l'ensemble des plans et de la fonction de densité, nous détaillons à la section 3.6 les itérations de l'algorithme glouton. Nous verrons que la fonction de densité aide à localiser un bon plan candidat, mais qu'un algorithme de raffinement adaptatif est nécessaire pour expliciter ce plan.

3.3.2 Fonction d'erreur et fonction de coût

Dans l'approximation par des *billboards*, nous allons projeter les sommets du modèle sur différents plans pour générer des textures. Les sommets sont donc « déplacés » de leur position initiale vers leur équivalent sur une texture. La fonction d'erreur que nous utilisons mesure ce déplacement. Cette mesure est effectuée soit dans l'espace objet soit dans l'espace image. Dans le premier cas, l'erreur est la distance entre la position originale d'un sommet dans le modèle et le point correspondant dans l'approximation. Dans le deuxième cas, l'erreur est la distance (en pixels) entre la *projection* dans l'image du sommet et la projection du point correspondant dans l'approximation. Ces deux mesures sont liées. En effet, si deux points (un sommet et son approximation) sont à une distance bornée dans l'espace objet, leurs projections pour un point de vue donné sont aussi à une distance bornée. L'inverse n'est cependant pas vrai car deux point qui se projettent en un même pixel ne sont pas confondus mais simplement alignés avec le centre de projection. Nous distinguons donc une erreur *indépendante* du point de vue (mesure dans l'espace objet) et une erreur *dépendante* du point de vue (mesure dans l'espace image pour un ensemble fixé de points de vue). Dans l'exposé de l'algorithme nous nous placerons dans le cas indépendant du point de vue. La Section 3.10 reprendra l'algorithme et explicitera les modifications dans le cas dépendant du point de vue.

La fonction de coût mesure quant à elle le temps de rendu de l'approximation. Elle prend en compte le nombre de *billboards*, et idéalement la taille des textures qui les habillent. Pour l'instant, nous ne considérons que le nombre de *billboards*.

Nous verrons à la fin du chapitre comment nous envisageons dans le futur d'intégrer la prise en compte du coût lié à la texture.

3.3.3 Validité

Nous travaillons à erreur fixée. Le concept de validité modélise le fait qu'une approximation du modèle respecte cette borne sur l'erreur. Parmi toutes les approximations « valides », nous chercherons celle qui minimise le coût. La validité est définie pour les sommets et les faces du modèle.

Un point de l'espace est dit valide pour un sommet si la distance entre ce point et le sommet est inférieure à l'erreur autorisée, que nous notons ε. Pour un sommet, nous définissons le domaine de validité comme l'ensemble des points valides. Dans le cas indépendant du point de vue, il s'agit donc d'une sphère de rayon ε centrée sur le sommet. La forme du domaine dans le cas dépendant du point de vue sera explicité à la Section 3.10.

Un plan est dit valide pour un sommet s'il intersecte son domaine de validité. Autrement dit, la « projection » du point sur le plan est à moins de ε du point. Pour un sommet S nous notons $v_\varepsilon(S)$ l'ensemble des plans qui sont valides. Un plan est dit valide pour une face s'il est valide pour chacun de ses points. Par linéarité de la projection, il suffit qu'il soit valide pour chacun des sommets. Pour une face f, nous notons $v_\varepsilon(f)$ l'ensemble des plans qui sont valides. Soient (S_i) les sommets de la face. On a :

$$v_\varepsilon(f) = \bigcap_{i=1}^{n} v_\varepsilon(S_i) \tag{3.1}$$

Pour un plan $\mathcal{P}$, on peut considérer l'ensemble des faces pour lesquelles le plan est valide, que l'on notera $v_\varepsilon(\mathcal{P})$. On a trivialement :

$$f \in v_\varepsilon(\mathcal{P}) \iff \mathcal{P} \in v_\varepsilon(f) \tag{3.2}$$

3.3.4 Couverture géométrique

Le problème d'optimisation à résoudre est une variation du problème de couverture géométrique [Hoc96] dont la formulation intuitive est la suivante : étant donné un ensemble de sachets de billes de différentes couleurs, choisir un nombre minimal de sachets de telle sorte que l'on ait au moins une bille de chaque couleur. Dans notre problème, nous avons un ensemble de faces (f_i) et leurs domaines de validité $(v_\varepsilon(f_i))$, et nous cherchons un ensemble minimal de plans $\mathcal{P}_k$ tels que pour chaque face, un des plans au moins soit valide. Autrement dit :

$$\forall i \; \exists k \quad \mathcal{P}_k \in v_\varepsilon(f_i)$$

D'après l'équation (3.2) cela s'écrit aussi :

$$\forall i \; \exists k \quad f_i \in v_\varepsilon(\mathcal{P}_k)$$

Nous cherchons donc à recouvrir les faces par les ensembles de validité d'un nombre minimal de plans. Pour voir apparaître le problème de couverture géométrique, il suffit de considérer les faces comme les couleurs des billes, et de prendre comme sachets toutes les intersections des domaines de validité des faces. La Figure 3.3 illustre cette formulation. Dans notre cas, la couverture géométrique n'est pas tout. Parmi tous les sachets de billes retenus, plusieurs peuvent contenir une bille d'une couleur donnée, par exemple rouge. Il faut donc choisir dans lequel de ces sachets prendre la bille rouge. Autrement dit, dans l'ensemble de plans sélectionné, parmi tout ceux qui sont valides pour une face, lequel devons-nous choisir pour approximer la face ?

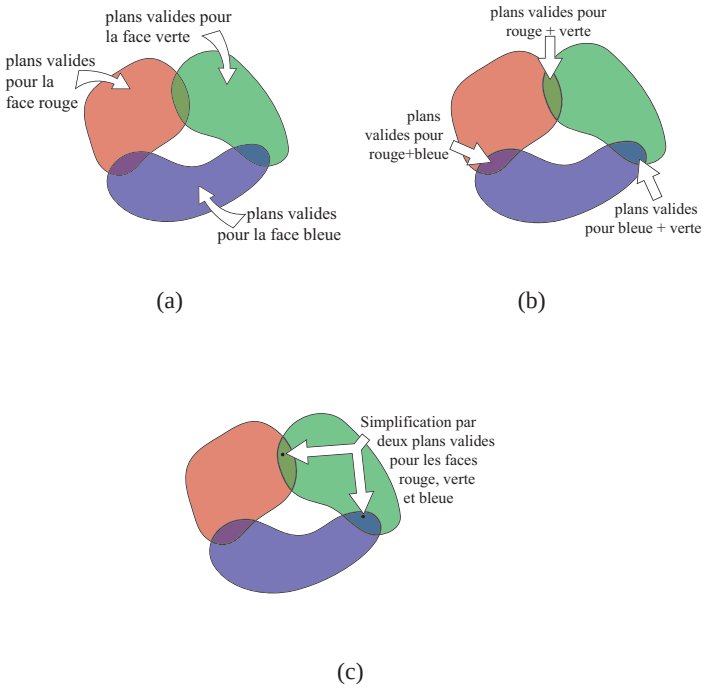


FIG. 3.3 – Couverture géométrique

Le problème de couverture géométrique est connu pour être NP-dur dans sa version continue. Dans sa version discrète (où l'on échantillonne l'ensemble des plans), il reste NP-complet [GJ79, Hoc96]. Pour le résoudre, nous choisissons un algorithme glouton : à chaque étape, on choisit le sachet de billes qui nous permet de récupérer le plus de couleurs manquantes. Dans les sections suivantes, nous explicitons la paramétrisation des plans, la façon dont nous échantillonnons cette paramétrisation pour nous ramener à un problème discret, et enfin la fonction de gain utilisée par l'algorithme glouton pour déterminer quel plan choisir à chaque pas.

3.4 Espace des plans

Pour travailler sur les plans, nous choisissons une paramétrisation de l'espace des plans. Cette paramétrisation est inspirée de la transformation de Hough [DH72] qui est utilisée en vision pour déterminer des droites dans un ensemble de points en deux dimensions. Il est naturel d'adapter cette transformation pour trouver des plans dans un ensemble de points en trois dimensions.

Un plan est paramétré par les coordonnées sphériques (ρ, θ, ϕ) du vecteur joignant un point fixe O et sa projection sur le plan (voir Figure 3.4). Le choix du point fixe n'est pas anodin. S'il est mal choisi, la paramétrisation ne sera pas optimale. En effet, nous voulons répartir l'ensemble des plans du modèle au mieux dans notre paramétrisation pour que, lorsque l'on discrétisera, l'ensemble des plans soit bien échantillonné. En outre, ce point ne doit appartenir à aucun des plans du modèle si l'on veut que la paramétrisation soit bijective. Ceci est toujours possible car les plans sont en nombre fini. Pour l'instant, le point est pris au centre de masse (pondéré par l'aire des faces) du modèle.

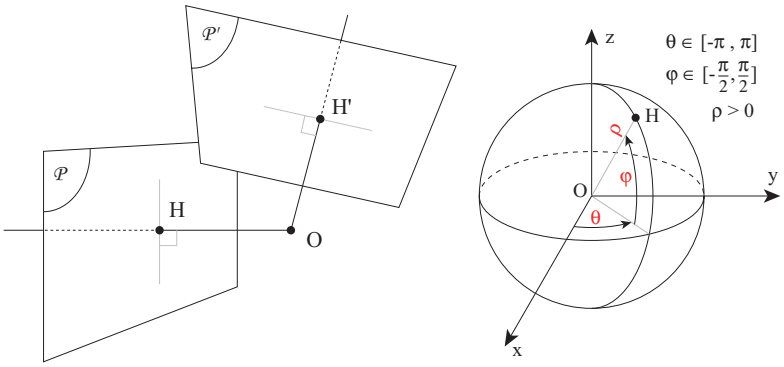


FIG. 3.4 – Paramétrisation d'un plan

3.4.1 Dualisation et représentation

Nous appellerons *dual* d'un plan le point de coordonnées cartésiennes (ρ, θ, ϕ). Nous représenterons cet espace dual comme sur la Figure 3.5. Il est pratique de voir cette représentation comme un champ de hauteur. La grille indique le plan (θ, ϕ) ; se déplacer le long de la grille c'est faire varier la direction de la normale d'un plan. L'élévation au dessus de la grille correspond à ρ. Changer l'élévation, c'est faire varier la distance du plan à l'origine O que nous avons choisie.

Dans l'espace dual, le point (ρ, θ, ϕ) est le dual du plan dont une normale $\mathbf{n}$ et

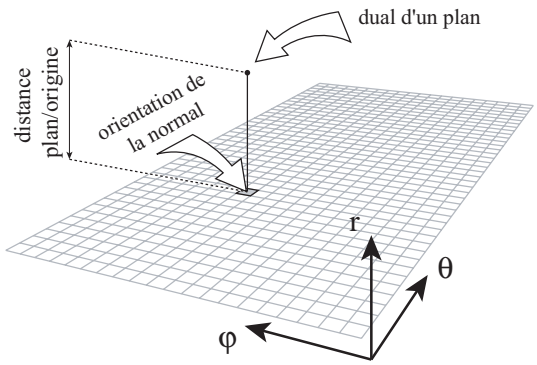


FIG. 3.5 – Représentation duale de l'ensemble des plans

Dans cet espace, un plan est représenté par un point dont les coordonnées sont les coordonnées sphériques du vecteur normal du plan, multiplié par la distance de l'origine au plan.

un point M sont donnés par :

$$\begin{aligned} \mathbf{n} &= \begin{pmatrix} \cos\theta\cos\phi \\ \sin\theta\cos\phi \\ \sin\phi \end{pmatrix} \\ M &= O + \rho\mathbf{n} \end{aligned}$$

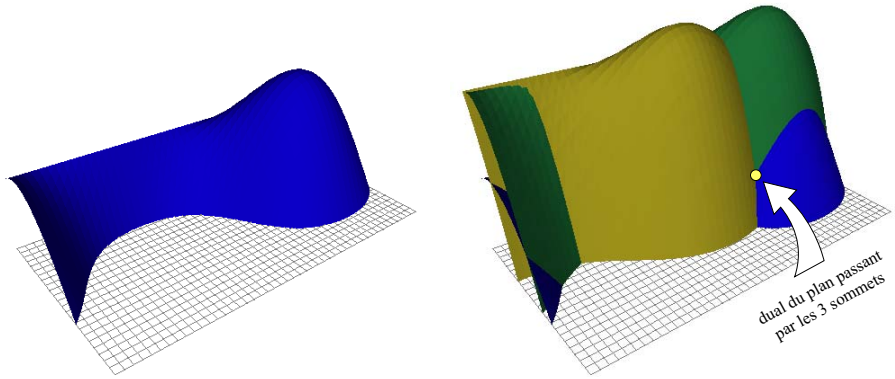
Réciproquement, considérons un plan $\mathcal{P}$ donné par un point M et son vecteur normal unitaire $\mathbf{n}$. Comme nous considérons des plans non orientés, $\mathbf{n}$ est choisi « s'éloignant de l'origine ». Autrement dit, on a $\overrightarrow{OM}.\mathbf{n} > 0$, le cas nul étant exclu du fait que O n'appartient pas au plan par hypothèse. Le dual de $\mathcal{P}$ a pour coordonnées :

$$\begin{aligned} \rho &= \overrightarrow{OM}.\mathbf{n} \\ \phi &= \sin^{-1}(z_{\mathbf{n}}) \\ \theta &= \begin{cases} \tan^{-1}(\frac{y}{x}) & \text{si } x > 0 \\ +\frac{\pi}{2} & \text{si } x = 0 \text{ et } y \geq 0 \\ -\frac{\pi}{2} & \text{si } x = 0 \text{ et } y < 0 \\ \tan^{-1}(\frac{y}{x}) + \pi & \text{si } x < 0 \text{ et } y \geq 0 \\ \tan^{-1}(\frac{y}{x}) - \pi & \text{si } x < 0 \text{ et } y < 0 \end{cases} \end{aligned}$$

Regardons quels sont les duaux des différents éléments que nous manipulons. Le dual d'un plan est un point. Le dual d'un point M est une *nappe* dans notre représentation (Figure 3.6(a)). En effet, pour une direction $\mathbf{d}$ définie par les angles (θ, ϕ), il y a un seul plan perpendiculaire à cette direction qui passe par M. Sa distance à l'origine est donnée par :

$$\rho_M(\theta, \phi) = \overrightarrow{OM}.\mathbf{d} = x\cos\theta\cos\phi + y\sin\theta\cos\phi + z\sin\phi \tag{3.3}$$

Ceci définit un champ de hauteur. On ne considère que la partie $\rho > 0$ qui donne la nappe. Les points de cette nappe sont les duaux des plans qui passent par M. Ainsi, les nappes duales de trois points A, B, C non alignés s'intersectent en un unique point qui est le dual du plan défini par A, B et C, comme illustré sur la Figure 3.6(b).



(a) (b)

FIG. 3.6 – Dual d'un point

Le dual d'un point est l'ensemble des plans qui passent par ce point. Il forme une nappe (a) *dans notre représentation. Les nappes de 3 points non alignés s'intersectent en un point qui est le dual du plan passant par ces 3 points.*

Le dual d'une sphère de centre C et de rayon ε est défini comme l'ensemble des points duaux de plans intersectant la sphère. C'est une *couche* d'épaisseur constante 2ε délimitée par deux nappes qui sont les translatées de $\pm\varepsilon$ de la nappe duale de C (Figure 3.7). En effet, pour une direction (θ, ϕ), les plans perpendiculaires à cette direction qui intersectent la sphère sont situés à une distance ρ vérifiant :

$$\rho_C - \varepsilon \leq \rho \leq \rho_C + \varepsilon \tag{3.4}$$

où ρ_C est la distance du plan passant par C pour la direction (θ, ϕ). L'ensemble des plans qui intersectent trois sphères, c'est-à-dire le domaine de validité d'une face joignant les sommets des sphères, a pour dual l'intersection de trois couches.

3.4.2 Discrétisation

Il nous faut discrétiser l'ensemble des plans ; d'une part pour considérer un nombre fini d'éléments dans la résolution gloutonne du problème de couverture

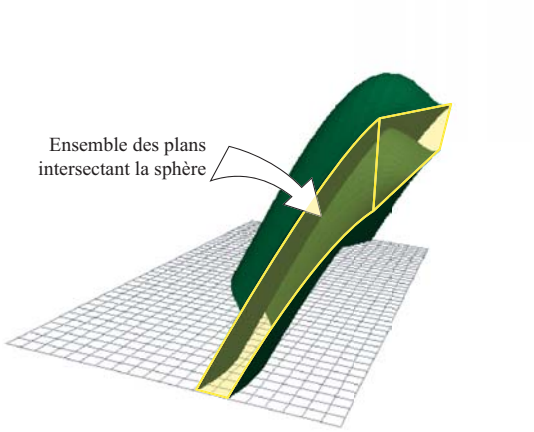


FIG. 3.7 – Dual d'une sphère : les points dans le volume souligné en jaune (situé entre les deux couches) sont les duaux de plan intersectant une sphère de diamètre la distance verticale entre les deux couches.

géométrique, et d'autre part parce que l'expression analytique du domaine de validité d'une face n'est pas simple à manipuler (rappelons qu'il s'agit de l'intersection de trois couches, chacune délimitée par des nappes dont l'équation est de la forme (3.3)).

Il y a plusieurs façons de discrétiser. Tout d'abord, on peut considérer soit un ensemble de plans échantillons, soit une partition de l'espace des plans. La première solution n'est pas la bonne. La Figure 3.8 montre comment cela conduit à « rater » le plan que l'on souhaiterait trouver. On y voit aussi que rajouter les plans supports des faces originales du modèle ne résoudrait rien : ils ont des directions complètement aléatoires par rapport à la forme globale. C'est qu'on appelle l'effet « crépi » : si l'on modélise à l'échelle géométrique un mur crépi, les normales aux polygones décrivant les aspérités du crépi ne sont pas du tout perpendiculaires au mur.

Nous choisissons donc de partitionner l'espace dual avec une grille 3D régulière et de discrétiser les ensembles de validité des faces dans cette grille. Il y a alors à nouveau deux choix. Pour une face donnée, on peut considérer les cellules de la grille qui sont entièrement dans son domaine de validité, ou alors considérer celles qui intersectent le domaine de validité. Dans le premier cas, on parle de cellule *uniformément* valide car tous les plans dont les duaux sont dans la cellule sont valides. Dans le deuxième cas, on parle de cellule *simplement* valide car il existe un plan valide dont le dual est dans la cellule. La validité uniforme n'est pas non plus une bonne solution, car si la grille n'est pas assez fine, la discrétisation du domaine de validité donnera un ensemble de cellules vide et l'on « ratera » des plans. Nous

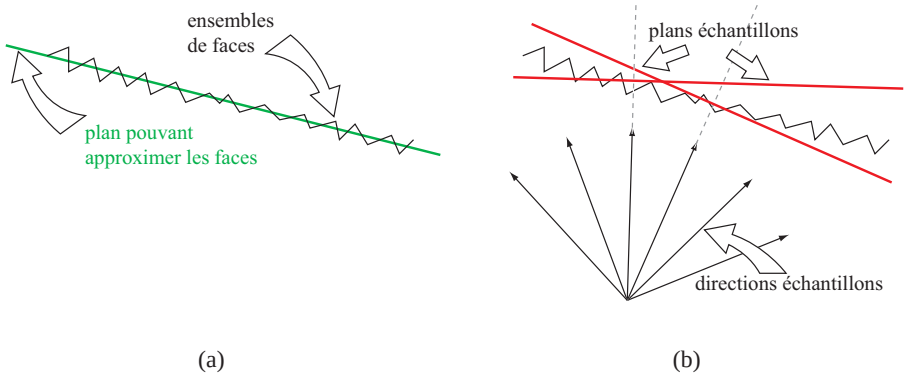


FIG. 3.8 – Échantillonnage de l'espace des plans

On souhaiterait approximer l'ensemble de faces indiqué en (a) par le plan « moyen » (en vert). Si l'on ne considère que des directions échantillons dans notre espace des plans (b), on ne trouvera pas de plan échantillon valide pour toutes les faces.

travaillons donc avec la validité simple. La Figure 3.9 montre la discrétisation de l'ensemble de validité d'une face à différentes résolutions.

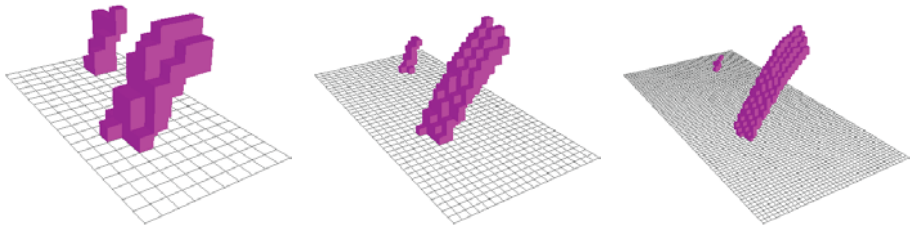


FIG. 3.9 – Discrétisation simple du domaine de validité d'une face à différentes résolutions

Si le choix de la validité simple permet une certaine indépendance à la résolution, il a aussi une conséquence importante. Qu'une cellule soit valide pour une face signifie seulement qu'il existe un ensemble de plans valides pour la face dans la cellule. Si la cellule est valide pour *deux* faces, les plans en question ne sont pas forcément les mêmes pour chaque face et en conséquence, il n'existe pas forcément de plan valide simultanément pour les deux faces. Nous reviendrons sur ce point plus tard.

3.4.3 Détermination de la validité simple

En première lecture, on pourra passer cette section et continuer directement à la section 3.5, page 87. Ici sont donnés les détails techniques de l'algorithme de détermination des cellules valides pour une face donnée.

Considérons une face f à trois sommets A, B, C. En combinant les équations (3.1),(3.3) et (3.4), on trouve qu'un plan (ρ, θ, ϕ) est valide pour la face si et seulement si il vérifie :

$$\begin{aligned} & |\rho - x_A \cos\theta\cos\phi + y_A \sin\theta\cos\phi + z_A \sin\phi| < \varepsilon \\ \text{et} \quad & |\rho - x_B \cos\theta\cos\phi + y_B \sin\theta\cos\phi + z_B \sin\phi| < \varepsilon \\ \text{et} \quad & |\rho - x_C \cos\theta\cos\phi + y_C \sin\theta\cos\phi + z_C \sin\phi| < \varepsilon \end{aligned} \tag{3.5}$$

L'équation ci-dessus définit une fonction implicite. Discrétiser une telle fonction dans une grille volumique est un problème connu. On peut appliquer un algorithme de *marching cubes* [LC87] pour déterminer les cellules qui intersectent l'ensemble ainsi défini. La solution ne sera cependant pas exacte car certaines cellules intersectant des détails très fins de l'ensemble peuvent être ratées. En outre, appliquer un algorithme de ce type pour les fonctions implicites définies pour chaque face du modèle est trop coûteux pour être envisagé. Nous avons donc dû développer un algorithme efficace qui discrétise de manière conservative l'ensemble de validité d'une face. En validité simple, toutes les cellules intersectant l'ensemble sont trouvées, mais certaines cellules trouvées n'intersectent en réalité pas l'ensemble (cette surestimation tend rapidement vers 0 quand on augmente la résolution de la grille). Au lieu de discrétiser le domaine de validité de la face, nous discrétisons ceux de ses sommets et retenons les cellules communes. Autrement dit, dans l'équation ci-dessous, nous approximons le terme de gauche par celui-de droite. La surestimation vient alors de l'inclusion stricte.

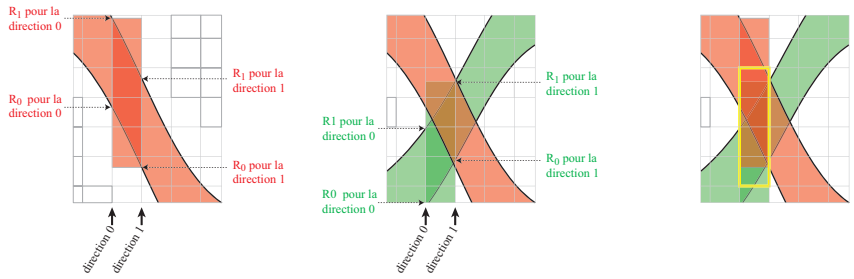
$$Discr(v_\varepsilon(f)) = Discr(\bigcap_{i=1}^{n} v_\varepsilon(S_i)) \subsetneq \bigcap_{i=1}^{n} Discr(v_\varepsilon(S_i)) \tag{3.6}$$

Voyons comment déterminer $Discr(v_\varepsilon(S))$ pour un sommet S. Considérons un ensemble de directions $[\theta_0, \theta_1] \times [\phi_0, \phi_1]$. On définit les quatres directions limites $(d^i)_{i=1...4}$ comme étant (θ_0, ϕ_0),(θ_1, ϕ_0),(θ_1, ϕ_1) et(θ_0, ϕ_1). Pour chacune d'entre elles on peut calculer, grâce à l'équation (3.3), l'intervalle $[\rho_a^i, \rho_b^i]$ des plans valides pour le sommet S. On définit :

$$\rho_a = \min_{i=1...4} \rho_a^i \tag{3.7}$$

$$\rho_b = \min_{i=1...4} \rho_b^i \tag{3.8}$$

On a alors par continuité la propriété suivante : pour chaque $\rho \in [\rho_a, \rho_b]$, il existe une direction $(\theta, \phi) \in [\theta_0, \theta_1] \times [\phi_0, \phi_1]$ telle que le plan (ρ, θ, ϕ) soit valide pour le sommet S. Autrement dit, l'ensemble des cellules $[\theta_0, \theta_1] \times [\phi_0, \phi_1] \times [\rho_0, \rho_1]$ avec



(a) (b) (c)

FIG. 3.10 – Discrétisation simple des plans valides pour un sommet

(a) *Pour un ensemble de directions, le domaine d'un sommet rouge est discrétisé par un intervalle de hauteurs.* (b) *Même chose pour un sommet vert.* (c) *L'ensemble de validité de la face contenant les sommet rouge et vert (le « losange ») est discrétisé en prenant les cellules qui intersectent l'intersection de ces deux intervalles (encadrées en jaune). Seul l'intervalle de direction entre d_0 et d_1 est ici considéré.*

$[\rho_0, \rho_1] \cap [\rho_a, \rho_b] \neq \emptyset$ sont simplement valides pour le sommet S. La Figure 3.10 résume ce calcul. Pour calculer le terme de droite de l'équation (3.6) il suffit alors de prendre l'intersection des intervalles $[\rho_a, \rho_b]$ pour tous les sommets d'une face.

Nous avons donc un procédé permettant de déterminer, pour un échantillon de directions donné dans notre discrétisation, les cellules qui sont valides pour une face. En itérant sur l'ensemble des directions échantillons, on détermine une discrétisation de l'ensemble de validité d'une face. On notera au passage que ces itérations se font en deux dimensions (sur les directions) et non pas sur les trois dimensions comme une implémentation naïve pourrait le faire. Le pseudo-code (3.1) résume ce procédé de discrétisation. La fonction 'distance' se calcule par l'équation (3.3). La discrétisation en θ contient N_θ cellules $\theta[i]$, qui sont des intervalles de valeurs délimités par $\theta[i]_{min}$ et $\theta[i]_{max}$; de même pour ϕ et ρ. La fonction 'k_pour(r)' renvoie k tel que $r \in \rho[k]$.

Comme nous l'avons déjà dit, l'existence d'une cellule valide pour plusieurs faces n'implique pas l'existence d'un plan valide pour ces faces. De la même manière, l'approximation (3.6) signifie que l'existence d'une cellule valide pour une face n'implique pas l'existence d'un plan valide pour la face. En effet, avec cette approximation, une cellule est considérée valide pour une face si pour chaque sommet, il existe un plan dans la cellule qui soit valide au sommet. Ce plan n'est pas forcément le même pour tout les sommets, et il n'y a donc pas nécessairement de plan valide pour la face dans la cellule.

Algorithme 3.1 Algorithme de discrétisation

Fonction *DiscrSimple*
Entrée: une face f définie par ses n sommets S_i
Sortie: un ensemble d'indices de cellules

1. $C \leftarrow \emptyset$
2. **pour** $i, j \leftarrow 0{,}0$ **à** N_θ, N_ϕ
3. **faire** $R_0 \leftarrow 0$
4. $R_1 \leftarrow \infty$
5. **pour** chaque sommet S
6. **faire** $r^0 \leftarrow \text{distance}(S, \theta[i]_{min}, \phi[j]_{min})$
7. $r^1 \leftarrow \text{distance}(S, \theta[i]_{max}, \phi[j]_{min})$
8. $r^2 \leftarrow \text{distance}(S, \theta[i]_{max}, \phi[j]_{max})$
9. $r^3 \leftarrow \text{distance}(S, \theta[i]_{min}, \phi[j]_{max})$
10. $r_0 \leftarrow \min(r^0, r^1, r^2, r^3) - \varepsilon$
11. $r_1 \leftarrow \max(r^0, r^1, r^2, r^3) + \varepsilon$
12. $R_0 \leftarrow \max(R_0, r_0)$
13. $R_1 \leftarrow \min(R_1, r_1)$
14. **pour** $k \leftarrow \text{k_pour}(R_0)$ **à** $\text{k_pour}(R_1)$
15. **faire** $C \leftarrow C \cup \{(i, j, k)\}$
16. **retourner** C

3.5 La fonction de densité

À ce stade de l'exposé, nous avons une discrétisation de l'espace des plans, et un procédé pour discrétiser l'ensemble de validité des faces du modèle. L'objectif de l'algorithme glouton que nous sommes en train de décrire est de déterminer, à chaque itération, un plan valide pour un maximum de face.

Une façon de faire cela consisterait à associer à chaque cellule de la discrétisation la liste des faces pour lequelles elle est valide, puis de choisir la cellule avec la plus grande liste. On rencontre alors un problème algorithmique. Stocker pour chaque cellule une liste de faces entraîne un coût mémoire potentiel de $p \times n$ où p est le nombre de cellules et n le nombre de faces. Ce coût est prohibitif. Au lieu de stocker la liste, nous stockons pour chaque cellule un scalaire, appelé *densité*, qui mesure la « quantité » de face potentiellement approximable par un plan de la cellule. Cette densité dans sa version la plus simple est simplement le cardinal de la liste précédemment associée aux cellules. Dans cette section, nous exhibons une fonction de densité plus évoluée qui conduit à de meilleurs résultats. Cette fonction fait intervenir deux termes, la *contribution* et la *pénalité*, qui sont évalués pour chaque face pour laquelle une cellule est valide, et « accumulés » pour donner la densité de la cellule.

3.5.1 Contribution

Parmi tout les plans valides pour un ensemble de faces donné, il en est certains qui paraissent plus appropriés que d'autres. Dans la Figure 3.11, le plan rouge est valide pour les trois faces représentées. Cependant, le plan vert, qui est lui aussi valide, semble intuitivement plus approprié. Pour favoriser ainsi les plans parallèles aux grandes faces du modèle original, chaque face contribue à la densité d'une cellule à hauteur de son aire projetée sur un plan représentatif de la cellule. Nous choisissons de façon naturelle comme plan représentatif celui dont le dual est au centre de la cellule.

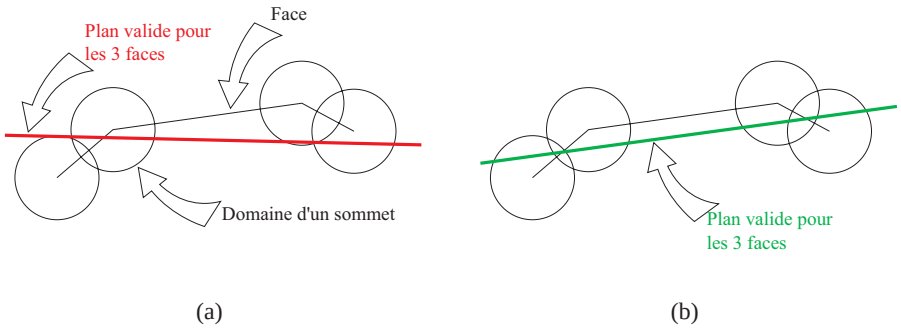


FIG. 3.11 – Contribution des aires projetées

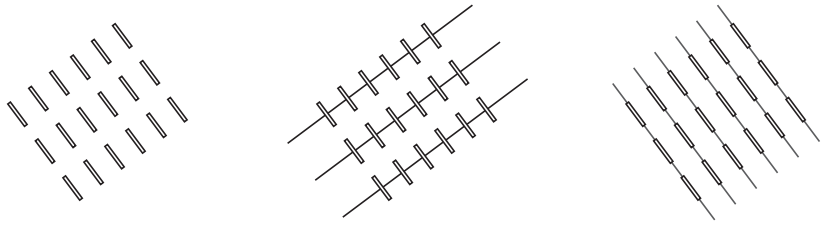
Les plans rouge (a) et vert (b) sont tous les deux valides pour les trois faces indiquées (ils intersectent chaque sphère). Pourtant, le plan vert approxime mieux la forme des trois faces.

La pondération par l'aire projetée permet également de traiter des cas particuliers comme celui des « dominos ». Imaginons un ensemble de dominos posés verticalement sur une table comme illustré sur la Figure 3.12. Les trois plans placés dans le sens de la longueur sont des plans valides mais conduiraient à une très mauvaise approximation des dominos, car les projections de ceux-ci sur les plans sont petites, d'aire très faible. En utilisant l'aire projetée comme mesure de la « quantité de faces » approximables par un plan, on obtiendra les six plans dans la largeur. La simplification comportera plus de plans mais sera de bien meilleure qualité.

3.5.2 Pénalité

Le choix du plan au centre d'une cellule pour évaluer la contribution d'une face à une cellule se justifiera plus loin, dans l'algorithme glouton. Pour l'instant, nous montrons que ce choix n'est pas sans conséquence et qu'il cause un artefact subtil dont nous exposons les détails dans ce paragraphe.

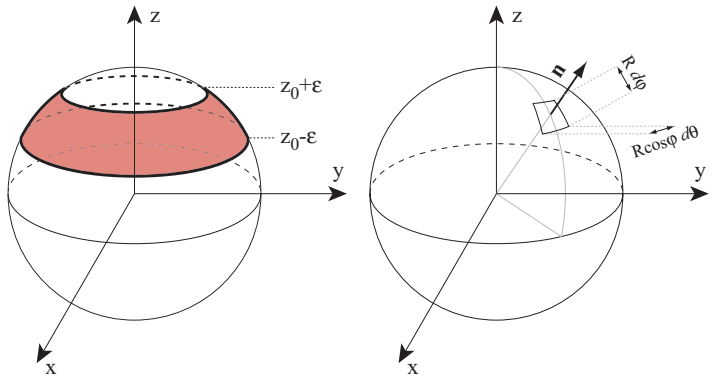
Considérons un modèle d'une sphère de rayon R et un plan quelconque. Par



(a) (b) (c)

FIG. 3.12 – Effet « dominos »

*Pour approximer un ensemble de dominos posés verticalement (*a*) on peut choisir 3 plans (*b*). Les projections sur ces 3 plans des dominos étant d'aire faible, on préfèrera une simplification sous-optimale de 6 plans (*c*).*



(a) (b)

FIG. 3.13 – Contribution cumulée pour la sphère

isotropie, on peut considérer que le plan est perpendiculaire à (Oz) et paramétré par sa distance z_0 à l'origine (Figure 3.13(a)). En considérant la sphère comme infiniment tesselée, calculons la somme des contributions des faces de la sphère pour lesquelles le plan est valide. Si l'erreur autorisée est ε ces faces constituent la couronne sphérique intersectée par les plans $z = z_0 - \varepsilon$ et $z = z_0 + \varepsilon$ (en rouge sur la figure). Considérons un patch élémentaire sur cette couronne (Figure 3.13(b)). Il a pour longueur $R\cos\phi d\theta$, pour largeur $Rd\phi$ et pour normale

$\mathbf{n} = (\cos\theta\cos\phi, \sin\theta\cos\phi, \sin\phi)T$. Son aire projetée sur le plan $z = z_0$ est donc :

$$\begin{aligned} dA &= R^2\cos(\phi)\sin(\phi)\,d\theta\,d\phi \\ &= \frac{R^2}{2}\sin(2\phi)\,d\theta\,d\phi \end{aligned}$$

L'aire de la calotte est donc :

$$\begin{aligned} A_\varepsilon &= \int_{-\pi}^{+\pi}\int_{-\phi_{min}}^{-\phi_{max}} \frac{R^2}{2}\sin(2\phi)\,d\theta\,d\phi \\ &= \pi R^2\left[-\frac{1}{2}\cos(2\phi)\right]_{\phi_{min}}^{\phi_{max}} \\ &= \pi R^2\left[\sin^2(\phi) - \frac{1}{2}\right]_{\phi_{min}}^{\phi_{max}} \\ &= \pi R^2\left[\sin^2(\phi_{max}) - \sin^2(\phi_{min})\right] \\ &= \pi\left[\max(R, z_0+\varepsilon)^2 - (z_0-\varepsilon)^2\right] \end{aligned}$$

Autrement dit :

$$A_\varepsilon = \begin{cases} 4\pi z_0\varepsilon & \text{si } \varepsilon \le z < R-\varepsilon \\ \pi\left(R^2 - (z_0-\varepsilon)^2\right) & \text{si } R-\varepsilon \le z \le R+\varepsilon \end{cases} \tag{3.9}$$

Si nous traçons cette aire en fonction de z_0, nous obtenons la courbe de la Figure 3.14(a). Le maximum est obtenu pour $z = R - \varepsilon$, c'est-à-dire pour le plan coupant la plus grande calotte. Cependant, notre algorithme ne trouvera pas forcément ce plan, et ce pour deux raisons. La première vient du fait que l'on évalue la contribution à l'aide du plan situé au centre de la cellule. Autrement dit, la courbe est en réalité échantillonnée comme indiqué sur la Figure 3.14(b). À cause de cela, il est possible que la cellule de contribution maximale, c'est-à-dire celle qui sera choisie par l'algorithme glouton pour trouver le premier plan approximant, ne soit pas celle qui contienne le maximum réel.

La deuxième raison est liée à l'emploi de la validité simple et la façon dont nous discrétisons les domaines de validité des faces (Section 3.4.3). En effet, les faces qui contribuent à une cellule sont, comme nous l'avons vu, surévaluées. Cela peut-être vu comme une augmentation de l'erreur autorisée ε. La Figure 3.15 illustre ce phénomène. En conséquence, tout se passe comme si nous calculions $A(\varepsilon + \Delta\varepsilon)$ au lieu de $A(\varepsilon)$. La Figure 3.16 montre le profil de contributions obtenu lorsque ε augmente. On voit alors que le maximum est décalé vers l'intérieur de la sphère.

Au final, la cellule de contributions maximale ne contient pas le maximum réel. Comme aucun plan de cette cellule n'est valide pour les faces au voisinage de $z = R$, le plan choisi va nécessairement « trancher » la sphère comme indiqué sur la Figure 3.17. Pour compenser cet effet de la discrétisation et de la validité simple,

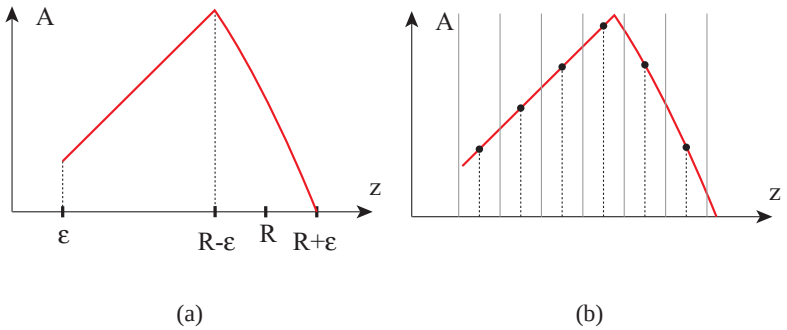


FIG. 3.14 – Contribution cumulée en fonction de z

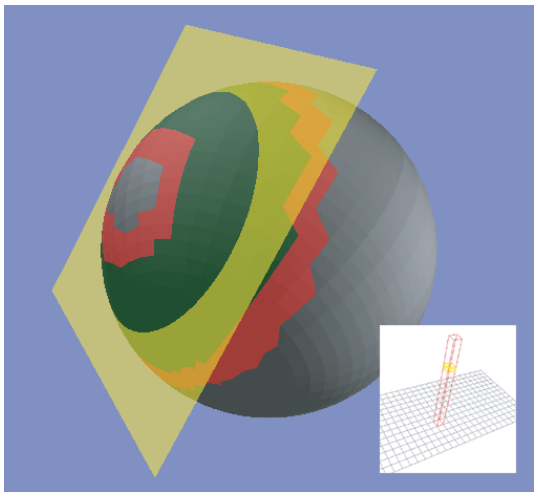


FIG. 3.15 – Surévaluation de la contribution

Les faces en rouge et vert foncé sont celles pour lesquelles la cellule indiqué en jaune dans l'encart est considérée comme valide. Le plan central de la cellule est dessiné en jaune. Les faces pour lesquelles il est effectivement valide sont les faces vertes. La contribution de la cellule est donc surévaluée par rapport à la contribution du plan central.

nous introduisons la notion de *pénalité*. Le but est donc de favoriser les plans « tangents » au modèle initial. Pour cela, nous allons pénaliser les plans qui ratent « de peu » un certain nombre de faces. Chaque face va apporter en quelque sorte une contribution négative aux cellules qui sont presque valides pour elles. Nous avons vu que pour une face et un intervalle de direction donnés, on pouvait calculer l'intervalle $[R_0, R_1]$ de cellules valides (cf. algorithme 3.1). Nous définissons les cellules presque valides comme celles situées dans les intervalles $[R_0 - \delta, R_0]$ et $[R_1, R_1 + \delta]$, où δ est une constante donnée. Autrement dit, une cellule presque va-

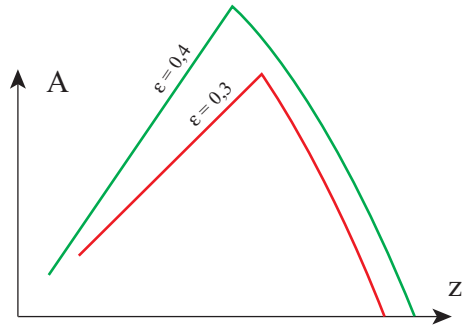


FIG. 3.16 – Courbe réelle et effective des contributions

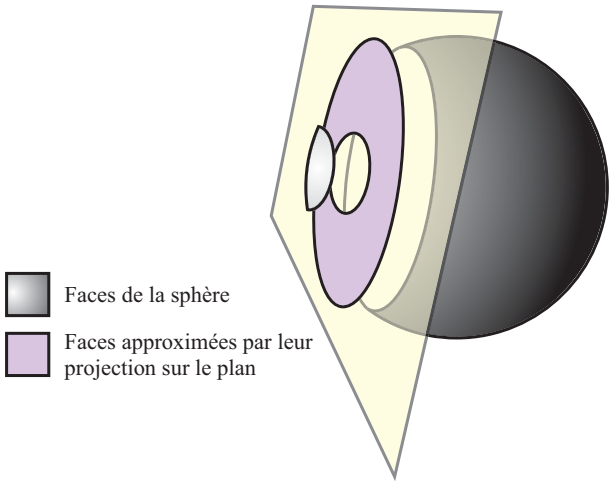


FIG. 3.17 – Approximations d'une sphère

lide est une cellule qui deviendrait valide si elle était décalée de δ vers le bas ou vers le haut.

Dans l'exemple de notre sphère, la pénalité cumulée en z est donc l'aire des faces de la couronne située entre $z + \varepsilon$ et $z + \varepsilon + \delta$. En utilisant l'équation (3.9), il vient :

$$P_{\varepsilon,\delta}(z) \quad = \quad A_{\frac{\delta}{2}}(z + \varepsilon + \frac{\delta}{2}) \tag{3.10}$$

La densité D_ε d'une cellule est définie comme la somme pondérée des contributions et des pénalités apportées par les faces pour lesquelles la cellule est considérée valide. Si l'on note $v_\varepsilon(i, j, k)$ cet ensemble, on a :

$$D_\varepsilon(i, j, k) = \sum_{f \in v_\varepsilon(i,j,k)} (C_\varepsilon(f) + \alpha P_{\varepsilon,\delta}(f)) \tag{3.11}$$

où α est un coefficient de pondération à choisir. Évaluer la densité naïvement en utilisant la formule ci-dessus implique de parcourir chaque cellule de la discretisation, et pour chacune d'entre elles de parcourir chaque face pour laquelle la cellule est valide. Dans notre cas, l'ensemble des faces valides pour une cellule n'est pas explicitement calculé. Comme nous l'avons vu à la Section 3.4.3, nous déterminons plutôt pour chaque face les cellules valides. La contribution et la pénalité de la face sont alors ajoutées à celles stockées pour chacune de ces cellules. En itérant sur l'ensemble des faces, nous évaluons donc bien la densité définie par l'équation (3.11).

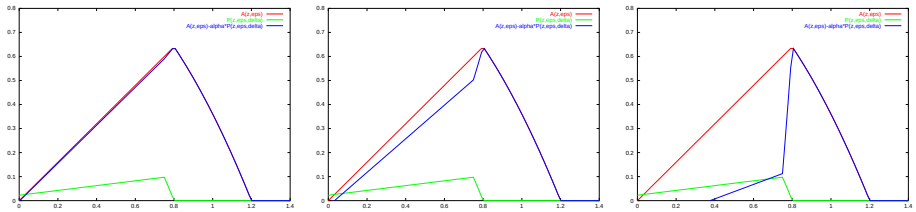
Cette fonction représente donc la « quantité de faces » approximable par un plan. Elle est échantillonnée sur notre discrétisation de l'ensemble des plans, en calculant approximativement son intégrale sur chaque cellule. Il est logique de calculer une intégrale, plutôt que d'évaluer la fonction en des plans échantillons, car comme nous venons de le dire, la densité représente une quantité. Notre but étant de localiser les endroits où la densité est forte, l'utilisation de l'intégrale permet l'indépendance à la résolution de discrétisation. En effet, si l'intégrale est évaluée exactement, la somme des densités de chaque cellule est constante quelle que soit la résolution. Il n'y a donc pas disparition de densité au faibles résolutions. Cette propriété est importante car elle permettra la mise en place d'un algorithme adaptatif comme nous le verrons plus loin.

3.5.3 Choix de δ et de α

La fonction de densité dépend de ε d'une part, et de δ et α d'autre part. Si le premier paramètre, qui correspond à l'erreur autorisée, est forcément spécifié par l'utilisateur, nous allons voir que les deux autres peuvent être fixés indépendamment du modèle et de l'erreur autorisée. Ces deux paramètres sont entièrement liés à la pénalité. Leur but est d'empêcher si possible de choisir des plans qui tranchent le modèle. Reprenons l'étude théorique faite sur la sphère. La Figure 3.18 montre les profils de densités obtenus pour différentes valeurs de δ et α. On voit que, comme escompté, l'utilisation de la pénalité modifie le profil de contribution en marquant d'avantage son maximum. On voit en outre, que le choix de δ n'est pas critique dès que le coefficent α de pondération pénalité/contribution est élevé. En pratique, nous prenons donc $\alpha = 5$ et $\delta = \varepsilon$.

3.6 Algorithme glouton

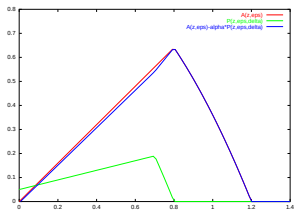
Reprenant le fil de l'algorithme glouton, nous en sommes maintenant à l'étape où, muni de notre fonction de densité discrétisée sur l'espace paramétré des plans, nous allons choisir un plan qui puisse approximer la plus grande quantité de faces. Le point clef est que la fonction de densité ne nous permet pas de trouver directement un plan. Comme nous l'avons vu, elle nous renseigne sur une région de l'espace des plans -une cellule- dans laquelle un tel plan existe *potentiellement*.



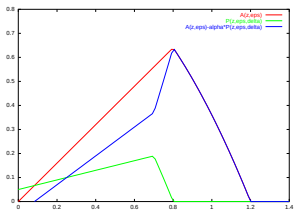
(a) $\delta = \frac{\varepsilon}{4}, \alpha = 0.1$

(b) $\delta = \frac{\varepsilon}{4}, \alpha = 1$

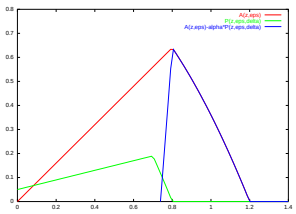
(c) $\delta = \frac{\varepsilon}{4}, \alpha = 5$



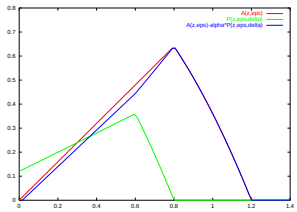
(d) $\delta = \varepsilon/2, \alpha = 0.1$



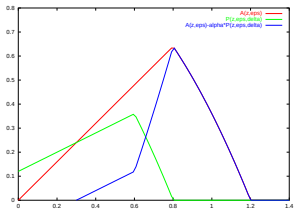
(e) $\delta = \varepsilon/2, \alpha = 1$



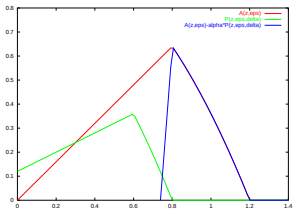
(f) $\delta = \varepsilon/2, \alpha = 5$



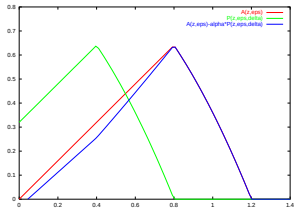
(g) $\delta = \varepsilon, \alpha = 0.1$



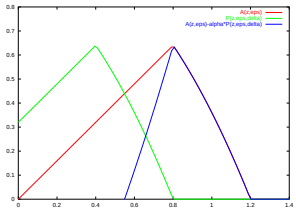
(h) $\delta = \varepsilon, \alpha = 1$



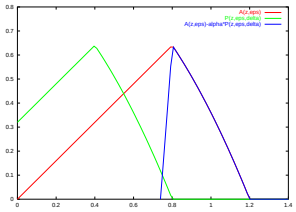
(i) $\delta = \varepsilon, \alpha = 5$



(j) $\delta = 4\varepsilon, \alpha = 0.1$



(k) $\delta = 4\varepsilon, \alpha = 1$



(l) $\delta = 4\varepsilon, \alpha = 5$

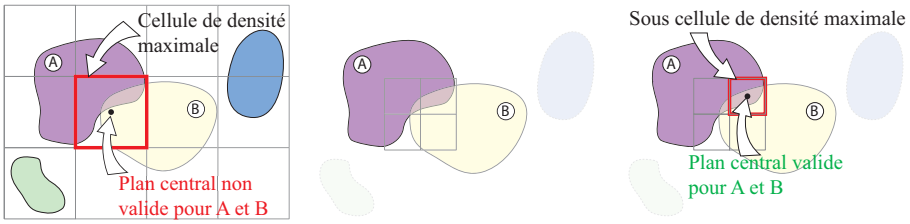
FIG. 3.18 – Densité en fonction de δ et de α

Les courbes ci-dessus représentent pour différentes valeurs de δ et de α les profils de contribution (rouge), de pénalité (vert) et de densité (bleu) correspondants.

En effet, considérons une cellule de forte densité. Tout ce que nous savons c'est qu'un certain nombre de faces (beaucoup de faces, des grandes faces, ou les deux) ont apporté leurs contributions à cette cellule. Mais la cellule n'est que simplement valide pour les faces. C'est-à-dire que pour chaque face, il existe au moins un plan valide dans la cellule. Mais il n'existe pas forcément un plan valide pour toutes faces. Pour déterminer si un tel plan existe, et pour le trouver, nous utilisons une stratégie de raffinement progressif décrit dans la section ci-dessous.

3.6.1 Raffinement adaptatif

La première étape consiste à récupérer la liste des faces f_i pour laquelle la cellule est simplement valide. Nous avons vu en effet qu'il n'était pas possible de stocker cet ensemble pour chaque cellule. Il doit donc être reconstruit une fois la cellule de densité maximale choisie. Une fois cela fait, nous testons si un plan particulier de la cellule -le plan central par exemple- est effectivement valide pour chacune des faces. Si c'est le cas, le problème est résolu. Sinon, nous subdivisons la cellule et calculons la densité pour les sous-cellules, *en n'utilisant que les faces pour laquelle la cellule de départ est valide*. Parmi ces sous-cellules, nous considérons alors celle de densité maximum, et nous ne considérons alors plus *que* les faces dont le dual intersecte cette sous-cellule. Ce processus de raffinement est réitéré jusqu'à ce que le plan central d'une sous-cellule soit valide. La Figure 3.19 illustre le processus. À l'issue du raffinement, nous récupérons donc un plan et un ensemble de face pour lequel il est valide. Pour éviter de raffiner trop longtemps, l'utilisateur fixe un seuil de subdivision maximal. Si ce seuil est atteint et que le plan central n'est pas valide pour toutes les faces associées à la sous-cellule de densité maximale, on s'arrête et on retourne le plan central et l'ensemble des faces pour lequel il est valide.

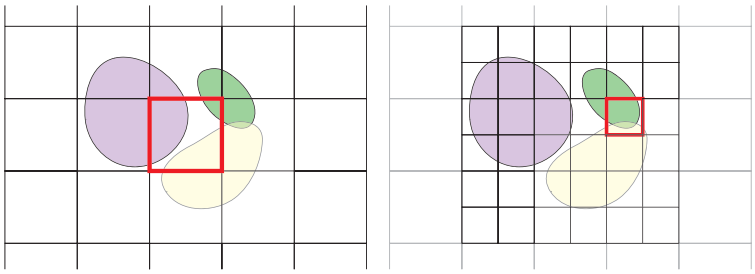


(a) (b) (c)

FIG. 3.19 – Exemple de raffinement adaptatif

Les ensembles de validité de 4 faces sont discrétisés dans une grille (a). *La cellule de densité maximum est valide pour les faces A et B mais pas son plan central. On subdivise la cellule et recalcule les densités en n'utilisant que A et B* (b). *Le plan central de la sous-cellule de densité maximum est valide pour ces 2 faces* (c).

En réalité, nous ne subdivisons pas seulement la cellule de densité maximum mais également les 26 cellules voisines. En effet, une fois encore en raison de la validité simple, le plan que l'on souhaite trouver n'est pas forcément dans la cellule de densité maximale, mais aux abords. La Figure 3.20 montre un cas où cela se produit. L'algorithme glouton et la procédure de raffinement sont synthétisés par les pseudo-codes 3.2 et 3.3.



(a) (b)

FIG. 3.20 – Raffinement des voisins dans l'algorithme glouton adaptatif

À un niveau de discrétisation donné, la cellule de densité maximale ne contient pas forcément de plan valide pour plusieurs faces (a). *En subdivisant également les voisins, on a une chance de récupérer un tel plan au niveau inférieur* (b).

Algorithme 3.2 Algorithme glouton

Fonction *Glouton*
Entrée: les faces du modèle, un seuil ε
1. ensemble de face $\mathcal{F}$ ←faces du modèle
2. nuage de *billboards* BC = $\emptyset$
3. **tant que** ($\mathcal{F} \neq \emptyset$)
4. **faire** Prendre la cellule B de plus grande densité
5. $v_\varepsilon(B)$ ←faces pour lesquelles B est valide
6. $\mathcal{P}_i$ ←**Raffine**(B,$v_\varepsilon(B)$)
7. RafraîchitDensité($v_\varepsilon(\mathcal{P}_i)$)
8. $\mathcal{F} \leftarrow \mathcal{F} - v_\varepsilon(\mathcal{P}_i)$
9. $BC \leftarrow BC \cup \mathcal{P}_i$
10. **retourner**

3.6.2 Construction d'un *billboard*

À l'issue d'une itération nous obtenons un plan et un ensemble de faces. Par construction, le plan est valide pour toutes les faces. Cependant, on peut essayer

Algorithme 3.3 Algorithme de raffinement adaptatif

Fonction *Raffine*
Entrée: une cellule B, un ensemble de faces $\mathcal{F}$
1. plan $P \leftarrow$le plan au centre de B
2. **si** $(v_\varepsilon(P) == \mathcal{F})$ **alors retourner** P
3. cellule $B_{max} \leftarrow$`NULL`
4. **pour** chacun des 27 voisins B_i de B
5. **faire** Subdivise B_i en 8 sous-cellules B_{ij}
6. **pour** chaque B_{ij}
7. **faire** Calcule la densité $d^{\mathcal{F}}(B_{ij})$ liée aux faces de $\mathcal{F}$
8. **si** $(d^{\mathcal{F}}(B_{ij}) > d^{\mathcal{F}}(B_{max}))$
9. **alors** $B_{max} = B_{ij}$
10. **retourner Raffine**$(B_{max}, v_\varepsilon(B_{max}))$
11.

d'améliorer encore ce plan. Car par exemple, si on a un ensemble de faces coplanaires, le plan que l'on va trouvé n'est en effet pas forcément le plan commun de ces faces. Pour traiter ce cas, nous contruisons le plan des moindre carrés des faces et testons si il est valide pour les faces. Si c'est le cas, nous le préférons au plan trouvé par l'algorithme glouton.

Un *billboard* est alors construit pour approximer les faces. Pour cela, chaque sommet est projeté sur le plan. On détermine (à l'aide de la librairie CGAL[3]), le rectangle englobant de ces points projetés, qui constituera le support du *billboard*. On note R un des sommets du rectangle, $\mathbf{e}_x$ et $\mathbf{e}_y$ les vecteurs (non unitaires et orthogonaux) définissant les côtés du rectangle et $\mathbf{n}$ la normale sortante du plan. Pour générer la texture, on effectue un rendu des faces avec OpenGL. On utilise une projection orthogonale sur le plan. On oriente la caméra et on choisit les paramètres de la projection orthogonale de telle sorte que l'image obtenue coïncide avec le rectangle. La caméra est placée « à l'extérieur » du plan. Les plans limites avant et arrière (*near et far planes*) sont placés à une distance ε de chaque côté du plan du *billboard*. On obtient le volume de projection indiqué sur la Figure 3.21. La couleur de fond du buffer couleur OpenGL est choisie noire avec une transparence totale. On désactive également l'éclairage et l'élimination des faces arrières (*backface culling*). La tableau 3.1 résume les appels OpenGL correspondant aux paramètres que nous venons de décrire.

La résolution de la texture n'est pas constante. En effet, nous ne souhaitons pas texturer un petit *billboard* avec autant de pixels qu'un grand. Nous souhaitons au contraire que le nombre de pixels par unité de longueur du modèle soit le même pour chaque *billboard*. Pour cela, nous fixons ce rapport en demandant à l'utilisateur combien de pixels il souhaite allouer pour la plus grande des dimensions de la boîte englobante du modèle. Par une règle de trois avec les dimensions du

[3] `www.cgal.org`

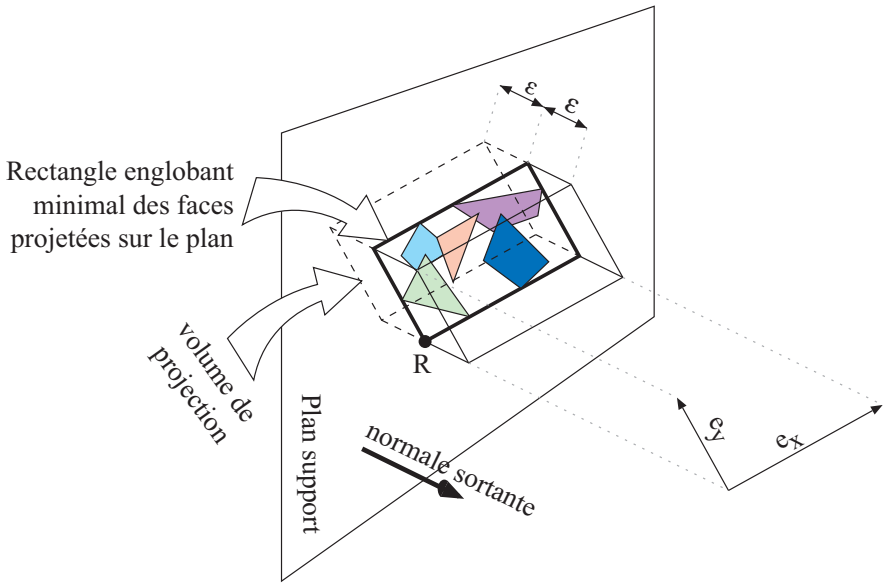


FIG. 3.21 – Projection orthogonale des faces sur un *billboard*

```
glClearColor(0.0f,0.0f,0.0f,0.0f);
glDisable(GL_LIGHTING);
glDisable(GL_CULL_FACE);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0f,w, // w = |ex|
        0.0f,h // h = |ey|
        -e,e); // e = epsilon
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(E[0],E[1],E[2], // E =R
          O[0],O[1],O[2], // O =R-n
          U[0],U[1],U[2]); // U =ey
glViewport(0,0,w,h);
```

TAB. 3.1 – Paramètres OpenGL pour la génération de *billboard*

billboard, on en déduit simplement la résolution de la texture à générer.

La texture obtenue est sauvegardée sur quatre canaux (RGBA), ainsi que les coordonnées dans l'espace des quatre coins du rectangle.

3.6.3 Mise à jour des pénalités

Une fois un *billboard* construit, l'ensemble des faces qu'il approxime doit être retiré du processus glouton. Pour cela, il faut mettre à jour les densités pour enlever

les contributions et les pénalités de ces faces. La procédure est exactement la même que celle qui sert à initialiser les densités, en changeant simplement l'addition en soustraction. L'algorithme glouton est répété jusqu'à ce que toutes les faces aient été projetées sur un plan. À ce moment, les densités restantes dans chaque cellule sont nulles.

3.7 Rendu d'un nuage de *billboards*

Lorsque l'algorithme glouton a terminé, nous obtenons un ensemble de *billboards* enchevêtrés de manière complexe. Cet ensemble représente le modèle initial dans les limites de l'erreur autorisée et peut-être utilisé à sa place dans différents algorithmes, comme nous le verrons à la section 3.9. Pour l'instant, nous décrivons comment générer des images du modèle en utilisant ce nuage, c'est-à-dire l'agorithme de rendu.

Pour rendre un nuage de *billboards*, il suffit de rendre chacun des rectangles texturés. Afin d'éviter les effets d'aliassage et de scintillement dus à l'utilisation de la transparence, il faut activer le *blending* d'OpenGL. Contrairement à ce qui est souvent fait en raison des indications trompeuses données par le manuel OpenGL, les paramètres à utiliser sont :

```
glBlendFunc(GL_ONE,GL_ONE_MINUS_SRC_ALPHA)
```

et non pas

```
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA)
```

comme expliqué par Porter et Duff [PD84]. Pour cela, il faut bien faire attention à pré-multiplier les couleurs par l'alpha, et c'est pourquoi nous utilisons un noir comme couleur de fond pour les textures des *billboards*. En outre, cela permet au *blending* de fonctionner correctement pour le *mip-mapping*.

Comme nous utilisons de la transparence, il nous faut également utiliser le test en alpha. Idéalement, ce test est inutile. En effet, si l'on rend les *billboards* du plus lointain au plus proche (supposons par exemple que l'on utilise un arbre BSP), l'utilisation du *blending* produit l'effet attendu, à savoir que l'on voit à travers les parties transparentes (alpha=0). Cependant, le test alpha ne coûte rien et permet d'éliminer des fragments à afficher. Mais surtout, nous ne souhaitons pas en première approche utiliser de structure de partitionnement pour ordonner les *billboards*. En effet, étant donné l'enchevêtrement de nos *billboards*, le nombre de primitives risque d'augmenter grandement. En outre, nous souhaitons garder une certaine simplicité de la méthode de rendu. Il nous faut donc activer le test en alpha pour rejeter tous les fragments qui ont un alpha de 0 (rappelons que les pixels des textures générées ont un alpha de 0 ou de 1). En pratique, nous rencontrons cependant un problème du à l'utilisation du *mip-mapping*. En effet, cette technique, qui sert à filtrer les textures à différentes résolutions, va transformer les valeurs binaires (0 ou 1) du masque de transparence d'un *billboard* en valeurs moyennes comprise dans l'intervalle $[0, 1]$. En utilisant 0 comme valeur de seuil pour le rejet

des fragments, on « rogne » alors les textures. Idéalement, cette valeur de seuil doit donc être déterminée pour chaque *billboard* en fonction du niveau de *mip-mapping* auquel la texture est visualisée. Ce niveau est déterminé automatiquement par OpenGL en fonction de la distance du *billboard* au point de vue courrant. En pratique, nous utilisons pour l'instant un seuil de 0.5, mais un seuil adaptatif est en cours d'étude.

3.8 Résultats

Dans cette section, nous montrons plusieurs résultats. Tout d'abord, nous montrons les différentes étapes du calcul sur un cas simple. Puis nous montrons quelques exemples complexes illustrant la généralité de la méthode. Enfin nous discutons de différents aspects de l'algorithmes : avantages, limitations, complexité. Les résultats ont été calculés avec notre implémentation de l'algorithme, réalisée en C++. Le code est optimisé, mais certaines parties existent pour des raisons de visualisation des résultats et pourraient être supprimmées, accélérant ainsi les calculs, et réduisant le coût mémoire. Certaines structures de données peuvent également être optimisées [Dow93], et des tests nous laissent penser que nous pouvons obtenir un gain de vitesse de 30%. En outre, certains calculs, comme celui de la densité, sont hautement parallélisables, et nous pensons pouvoir réduire encore les temps de calculs en utilisant une implémentation supportant plusieurs processeurs.

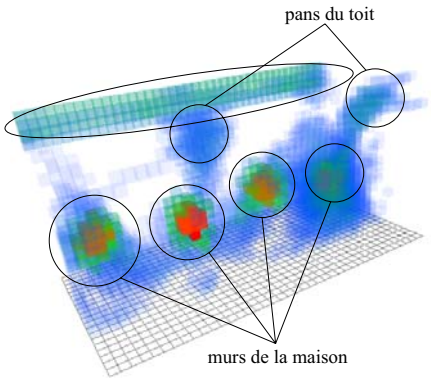
3.8.1 Un exemple simple

La Figure 3.22 montre un cas de base qui nous a servi au test et au développement de l'algorithme. Le modèle simplifié est une maison de 436 triangles et 6 textures (figures *(a)* avec et sans textures pour faire ressortir le relief de la façade avant lié aux polygones). La figure *(b)* montre les densités initialement calculées. La vue est en fausse couleur et avec transparence. Le rouge indique une forte densité, le vert une densité moyenne et le bleu une faible densité. On remarque 7 régions de forte densité qui correspondent aux plans des quatre murs, des deux pans du toit et du sol. Le plan du sol est un plan particulier qui correspond à un pôle de notre représentation : tout les couples $(\theta, \phi = \pi/2)$ représente la même direction quelque soit θ. C'est pourquoi la densité liée au plan du sol est « diluée » le long de la direction θ. Ce qu'il faut noter c'est que le profil de densité obtenu serait quasiment le même si les faces du modèle étaient subdivisées et légérement perturbées (par exemple si le crépi des façades était modélisé géométriquement). La Figure 3.22 page ci-contre montre ensuite les 7 itérations de l'algorithme, en indiquant à chaque fois le *billboard* construit, et la densité mise à jour.

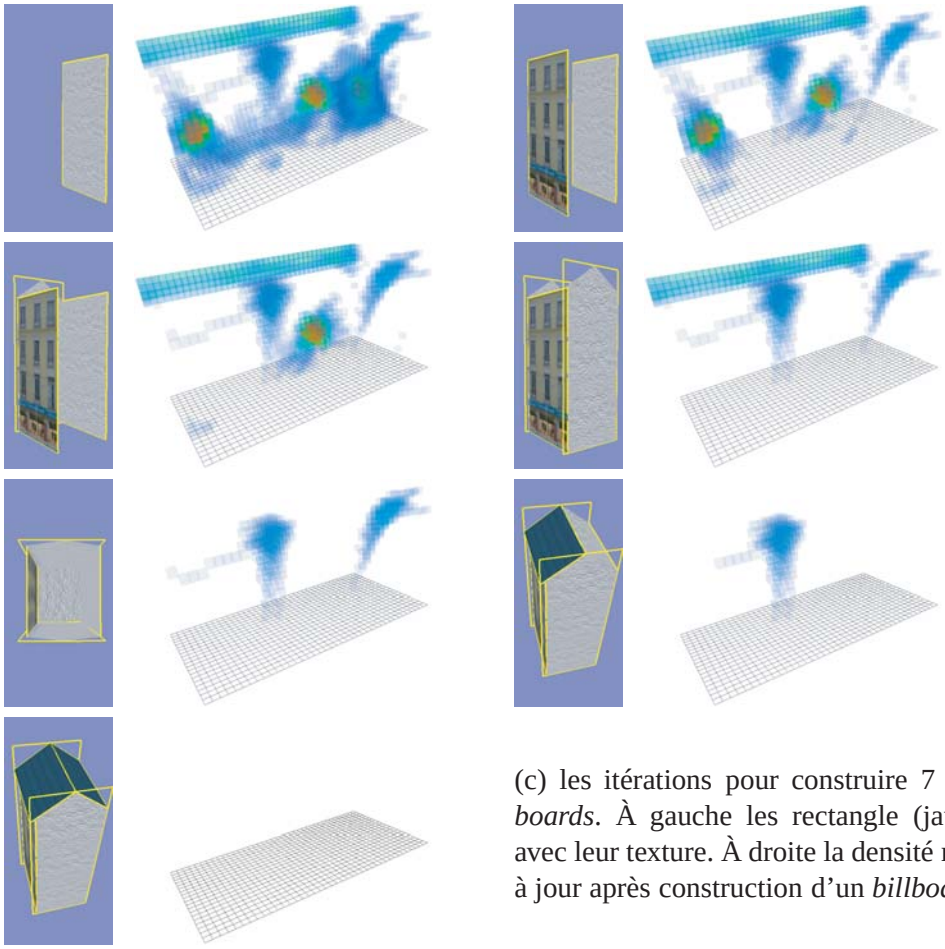
3.8.2 Exemples divers

Nous pensons qu'un des intérêts de l'algorithme que nous avons proposé est qu'il est capable de travailler sur n'importe quel type de modèle. Il prend en entrée

(a) modèle initial



(b) densité initiale



(c) les itérations pour construire 7 *billboards*. À gauche les rectangle (jaune) avec leur texture. À droite la densité mise à jour après construction d'un *billboard*.

FIG. 3.22 – Simplifier une maison en 7 étapes

une soupe de polygones. Les polygones n'ont même pas besoin d'être triangulés car la notion de validité est définie par sommet. Aucune hypothèse topologique n'est faite sur le maillage. Les modèles peuvent être texturés avec un nombre arbitraire de textures. La Figure 3.23 montre les résultats obtenus pour de tels modèles. Il est important de noter que le domaine de simplification dans lequel nous travaillons est celui de la *simplification extrême*, c'est-à-dire que les modèles simplifiés contiennent de l'ordre de la centaine de polygones.

FIG. 3.23 – Résultats pour des maillages complexes, multi-texturés et éventuellement dégénérés. Ligne supérieure : modèle polygonal. Ligne inférieure : nuage de *billboards*.

L'utilisation de textures avec de la transparence permet de restituer des effets visuels complexes, initialement modélisés avec des primitives polygonales. L'exemple de la Tour Eiffel est particulièrement parlant. Le modèle initial com-

porte 13 772 polygones. Notre algorithme construit un nuage de 32 *billboards*. Le résultat obtenu est donc très léger en complexité et coût de rendu, et pourtant, il permet de rendre fidèlement compte du modèle original. Utilisé à la place de celui-ci quand l'objet est à moyenne ou grande distance, il donne une très bonne silhouette et restitue correctement les effets de transparence à travers les piliers de la tour.

FIG. 3.24 – Un exemple avec des effets complexes de parallaxe et de transparence à travers l'objet. À gauche : modèle polygonal. À droite : 32 billboards.

Sur le modèle de la tour Eiffel, on constate également un autre atout de la représentation par *billboards*. En effet, lorsqu'un tel modèle est visualisé en utilisant la description polygonale, il se produit de très désagréables effets d'aliassage. Le modèle « clignote ». Sur le modèle de ferme de la Figure 3.25 on constate aussi des effets de moiré sur le grillage situé sur le devant (et représenté par des polygones), et de très pénibles clignotements sur les arbres en arrière plan. Ces effets disparaissent quand on utilise la représentation par *billboards* car l'utilisation de textures revient à effectuer un pré-filtrage du modèle.

FIG. 3.25 – Simplification d'une scène complète de 174 000 polygones (gauche) en 275 billboards (droite)

Le modèle de la ferme est également intéressant car il montre la capacité de l'algorithme à traiter une scène non structurée. Pour une telle scène, si l'information existait, on aurait certainement intérêt à calculer des nuages de *billboards* par objet. Mais quand cette information n'existe pas, la scène peut être traitée dans son ensemble. Le nuage obtenu pourra avantageusement être utilisé pour afficher une ferme au loin, dans un simulateur de conduite ou de vol par exemple.

Les temps de calculs pour les différents exemples que nous venons de voir sont donnés dans le tableau 3.2. On peut voir que ces temps, si ils sont tout à fait rai-

	# de polys	erreur autorisée	# de plans	# de texels	temps calcul (s)
Château	4,064	1%	106	218 k	8.5
Dinosaure	4,300	3%	86	3,935 k	51.0
Robot	59,855	3%	71	5,884 k	129.0
Tour Eiffel	13,772	4%	32	1,105 k	14.2
Ferme	174,325	1%	275	4,527 k	491.6

TAB. 3.2 – Statistiques pour les simplifications présentées sur les figures 3.23,3.24 et 3.25. Temps mesuré sur un processeur Pentium III 800 MHz. L'erreur autorisée est en pourcentage de la plus grande dimension de la boîte englobante de l'objet.

sonnables pour un pré-calcul, ne permettent pas d'envisager une utilisation dynamique. La complexité de l'algorithme est essentiellement $O(kn)$ où n est le nombre de primitives dans le modèle et k est le nombre de plans trouvés (le calcul initial des densités est $O(n)$ et chaque itération de l'algorithme glouton est approximativement $O(n)$). En ce qui concerne la complexité mémoire, elle est $O(n)$ avec une faible constante. Pour chaque face, il n'y a à stocker qu'un entier indiquant le plan sur lequel elle est au final simplifiée (cet entier indiquant aussi si la face est déjà simplifiée et peut être ignorée lors des différentes itérations de l'algorithme glouton). Dans notre implémentation, nous travaillons avec le modèle complètement chargé en mémoire. Théoriquement, cela n'est pas nécessaire. Pour construire les densités initiales, il suffit de parcourir successivement chaque face par exemple à partir d'un fichier. Une fois que la contribution d'une face est ajoutée aux différentes cellules de la grille de discrétisation, la face peut être « oubliée ». Le modèle n'a donc pas a être chargé entièrement en mémoire principale. À chaque itération, il faut reparcourir les faces pour déterminer lesquelles sont valides pour la cellule de plus forte densité considérée. Là encore, elles peuvent être parcourues séquenciellement.

La figure présente un dernier exemple pour montrer comment notre algorithme se comporte sur un modèle fortement tesselé, prototype du type de modèle sur lesquels sont classiquement testés les algorithmes de simplification de maillage. Le temps de calcul est de 30min pour un erreur autorisée égale à 2% de la plus grande dimension de la boîte englobante de la main. Bien que ne contenant que 23 *billboards*, le modèle simplifié est de bonne qualité et remplace très avantageusement le modèle inititial. C'est ce qui nous fait dire que notre méthode est efficace dans le

domaine de la simplification *extrême*, c'est à dire quand on cherche à simplifier un maillage en dessous de la centaine de polygones. En début de simplification, quand il s'agit par exemple d'éliminer le surplus de triangles du à l'utilisation d'un scanner et d'un échantillonnage régulier, les algorithmes classiques sont évidemment beaucoup plus adaptés. Mais ces algorithmes à base de décimation donneront un résultat dégénéré si l'on simplifie jusqu'à 23 polygones.

modèle polygonal

nuage de *billboards*

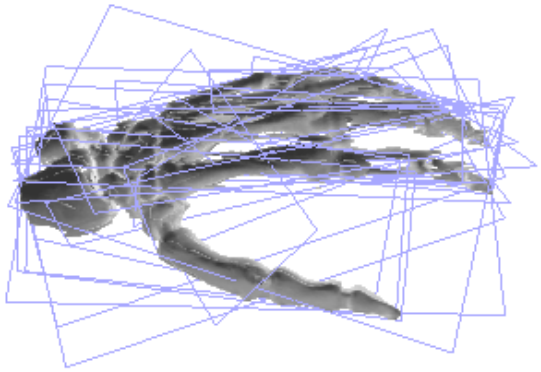


FIG. 3.26 – Un exemple complexe, une main à 654666 faces simplifiée en 23 *billboards*.

3.9 Applications

Nous avons introduit initialement les nuages de *billboards* dans l'optique d'effectuer un rendu accéléré d'un modèle à un niveau de détail donné. Plus généralement, ils fournissent une nouvelle représentation de modèle et une façon d'encoder plus efficacement l'information requise au sujet du modèle pour différents traitements. Un de ces traitements est justement le rendu et nous avonc vu dans la Section 3.8 que nous obtenons dans ce domaine de bons résultats. Mais nous envisageons d'autres applications que nous exposons ici brièvement. Pour certaines d'entre elles, nous présentons les résultats d'investigations préliminaires qui nous semblent convaincants.

3.9.1 Ombre portée au sol

Les concepteurs de jeu savent que l'ombre portée d'un objet sur le sol est un élément important contribuant au réalisme d'un univers 3D. Sans elle, les objets semblent « flotter » au dessus du sol. Si leur absence est de ce fait rapidement remarquée, elles ne sont paradoxalement pas observées consciemment, et le cerveau humain n'y prête que très peu d'attention *du moment qu'elles sont présentes et plausibles*. Pour cette raison, le budget de calcul alloué au rendu de cette ombre est très réduit et sa forme peut être très approximative. C'est pourquoi, dans beaucoup de jeux, l'ombre portée d'un personnage est un simple ovoïde.

Si l'on souhaite cependant une ombre plus précise, reflétant notamment mieux les détails de sillouhette, les nuages de *billboards* offrent d'intéressantes propriétés. En effet, ils contiennent toute l'information nécessaire pour générer l'ombre portée, et cette information est contenue sous une forme exploitable de manière optimale par les cartes graphiques. Les images de la Figure 3.27 montrent les résultats que l'on peut obtenir. L'application dont sont tirées ces images affiche les ombres au sol des différents modèles en temps réel avec la source de lumière (ponctuelle) déplacable en temps réel également. Pour le modèle de la ferme (117 000 polygones) de la Figure 3.30 par exemple, les ombres sont affichées en utilisant simplement 200 *billboards*. Les modèles sont affichés en utilisant leur représentation polygonale pour mettre en évidence le fait que différentes représentations peuvent être utilisées pour différents buts.

Considérons un seul *billboard*. Pour rendre son ombre portée, il suffit de calculer la projection des sommets du rectangle support sur le sol, et de texturer le quadrilatère obtenu avec la texture du *billboard*. Pour obtenir une couleur d'ombre particulière, il suffit de donner à ce quadrilatère la couleur souhaitée et d'utiliser le mode de texture OpenGL suivant :

```
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE).
```

Si le polygone est rendu en surimpression du polygone du sol (comme c'est le cas pour les trois exemples de la Figure 3.27), on pensera à utiliser le *polygone offset* de OpenGL, ou plus simplement à choisir comme fonction de test de profondeur `glDepthDunc(GL_LEQUAL)`.

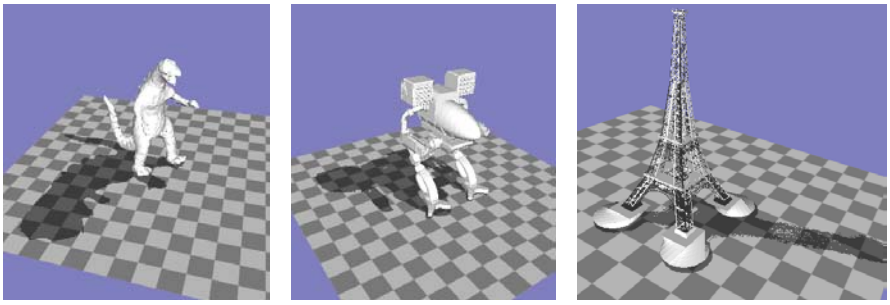
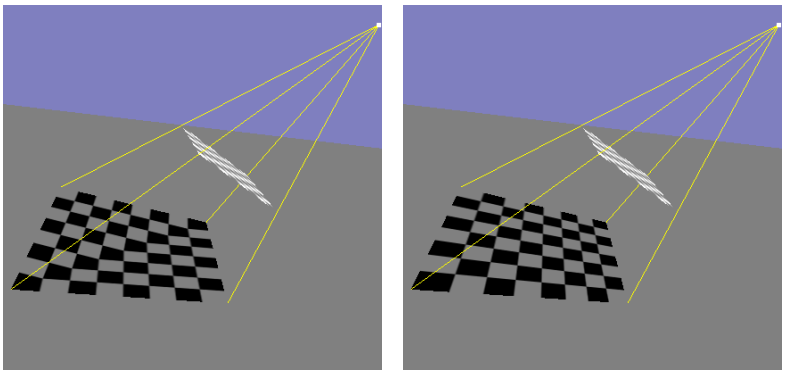


FIG. 3.27 – Exemples d'ombres portées

En outre, pour corriger la projection perspective il faut utiliser la matrice de texture. Si l'on ne fait pas cela, l'interpolation linéaire des coordonnées de texture produit un résultat incorrect comme montré dans l'exemple de la Figure 3.28 pour un *billboard* représentant un damier.



(a) Sans correction (b) Avec correction

FIG. 3.28 – Rôle de la matrice de texture

Comme l'ont montré Heckbert et Herf [HH97], la matrice qu'il faut utiliser est la matrice qui transforme la pyramide d'apex la position de la lumière et de base le rectangle support du *billboard*, en le cube unité. Cette transformation est montrée sur la Figure 3.29. Un point V de coordonnées homogènes (x, y, z) est transformé en un point P de coordonnées (u, v, w) avec $u \in [0, 1]$ et $v \in [0, 1]$ si et seulement si (AV) intersecte le *billboard*. Autrement dit, (u, v) sont les coordonnées de texture du pixel correspondant à l'intersection du rayon (AV) et du *billboard*.

Nous reproduisons ci-dessous la matrice donnée par Heckbert et Herf.

$$M = \begin{pmatrix} \alpha_x \mathbf{n_x}_x & \alpha_x \mathbf{n_x}_y & \alpha_x \mathbf{n_x}_z & -\alpha_x \mathbf{n_x}.\mathbf{b} \\ \alpha_y \mathbf{n_y}_x & \alpha_y \mathbf{n_y}_y & \alpha_y \mathbf{n_y}_z & -\alpha_y \mathbf{n_y}.\mathbf{b} \\ 0 & 0 & 0 & 1 \\ \alpha_w \mathbf{n_w}_x & \alpha_w \mathbf{n_y}_y & \alpha_w \mathbf{n_w}_z & -\alpha_w \mathbf{n_w}.\mathbf{a} \end{pmatrix} \tag{3.12}$$

avec $\mathbf{a} = \overrightarrow{OB}$, $\mathbf{b} = \overrightarrow{OA}$ et :

$$\begin{array}{rclcrcl} \mathbf{n_x} & = & \mathbf{e_w} \wedge \mathbf{e_y} & & \alpha_x & = & 1/\mathbf{e_x}.\mathbf{n_x} \\ \mathbf{n_y} & = & \mathbf{e_x} \wedge \mathbf{e_w} & \text{et} & \alpha_y & = & 1/\mathbf{e_y}.\mathbf{n_y} \\ \mathbf{n_w} & = & \mathbf{e_y} \wedge \mathbf{e_x} & & \alpha_w & = & 1/\mathbf{e_w}.\mathbf{n_w} \end{array}$$

Il faut faire attention à ce que la matrice est celle donnée par les auteurs qui sont

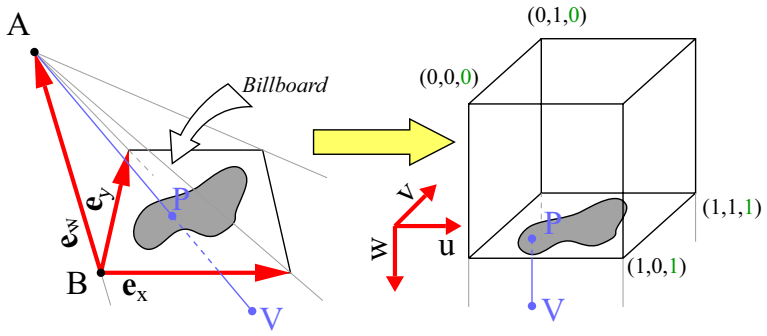


FIG. 3.29 – Matrice d'ombre

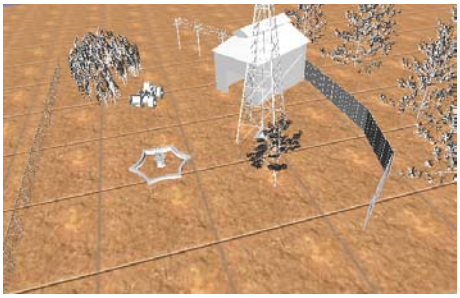
anglo-saxons, et que donc la multiplication se fait par la gauche (vecteurs lignes) et non par la droite (vecteurs colonnes) à la française. Autrement dit, on a $u = U/T$, $v = V/T$ avec $(U, V, W, T) = (x, y, z, 1) \times M$. Pour utiliser cette matrice de texture, il suffit de la charger dans OpenGL et de prendre comme coordonnées de texture des quatre coins du *billboard* leurs coordonnées dans l'espace. Le tableau 3.3 montre le code qui effectue le rendu d'une ombre d'un *billboard*.

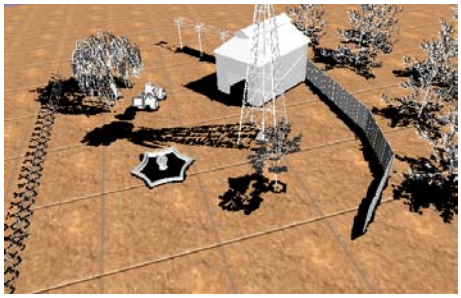
Ombre « transparente »

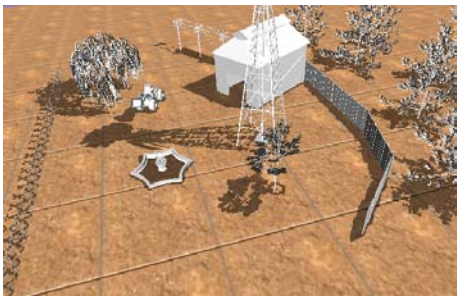
Sur les trois images de la Figure 3.27 on voit que les ombres sont « transparentes » : le damier du sol est visible à travers l'ombre. La méthode que nous venons de décrire ne permet pas cela car elle rend en surimpression sur le sol des polygones texturés d'une couleur fixée. Pour obtenir la transparence, nous utilisons d'abord le *stencil buffer* pour indiquer les pixels qui sont dans l'ombre. Nous rendons les ombres comme précédemment mais nous désactivons la mise à jour du *frame buffer* (l'écriture de pixels), et nous activons le test en alpha, et les opérations sur le *stencil buffer*. Les fragments dont l'alpha est supérieur à 0 passent le test, et

```
glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);
glBindTexture(GL_TEXTURE_2D,texture);

glMatrixMode(GL_TEXTURE);
glPushMatrix();
glLoadMatrix(M);

glBegin(GL_QUAD);
glTexCoord3fv(B[0]);glVertex3fv(B[0]);
glTexCoord3fv(B[1]);glVertex3fv(B[1]);
glTexCoord3fv(B[2]);glVertex3fv(B[2]);
glTexCoord3fv(B[3]);glVertex3fv(B[3]);
glEnd();

glPopMatrix();
```

TAB. 3.3 – Paramètres OpenGL pour le rendu d'ombre de *billboards*







Cette série montre trois vues d'une même scène sans ombrage (les objets semblent flotter sur le sol), avec un ombrage opaque et avec un ombrage transparent (qui laisse voir les détails du sol à travers l'ombre).

FIG. 3.30 – Différentes ombres

la valeur du stencil pour les pixels correspondants est mise à 1. Puis nous réactivons l'écriture dans le *frame buffer* et nous rendons un large polygone englobant tout les polygones d'ombres, en activant le test sur le stencil (seul les pixels dont la valeur de stencil est à 1 passent le test) et en activant le *blending*. Le tableau 3.4 montre le code OpenGL correspondant.

```
// Première passe : mise à jour du stencil
glDepthMask(false);
glColorMask(false,false,false,false);
glClear(GL_STENCIL_BUFFER_BIT);
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS,1,~0);
glStencilOp(GL_REPLACE,GL_REPLACE,GL_REPLACE);

glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER,0.0f);

renderShadowBillboards(); // Comme prédémment

// Deuxième passe : affichage d'un polygone
glDepthMask(true);
glColorMask(true,true,true,true);

glDisable(GL_TEXTURE_2D);
glDisable(GL_ALPHA_TEST);
glEnable(GL_BLEND);
glBlendFunc(GL_ONE,GL_ONE_MINUS_SRC_ALPHA);

glStencilFunc(GL_EQUAL,1,~0);
glStencilOp(GL_KEEP,GL_KEEP,GL_KEEP);

const float blackIntensity = 0.5f;
glColor4f(0.0f,0.0f,0.0f,blackIntensity);
glBegin(GL_QUADS);
glVertex3fv(sol[0]);
glVertex3fv(sol[1]);
glVertex3fv(sol[2]);
glVertex3fv(sol[3]);
glEnd();
```

TAB. 3.4 – Paramètres OpenGL pour le rendu d'ombre semi-transparentes

3.9.2 Intersection avec un rayon

La représentation d'un modèle sous forme de nuages de *billboard* peut être utilisée pour effectuer rapidement et de manière approximative des calculs d'intersection avec un rayon. Pour cela, il suffit de calculer l'intersection du rayon avec le plan support d'un *billboard* et de regarder si le pixel correspondant est opaque ou non. Nous avons implémenté une version simpliste de ce calcul pour simuler un jeu de tir, et déterminer si un objet se trouve sur la trajectoire d'une balle. Quand une balle n'est pas bloquée, elle laisse un impact sur un mur situe derrière l'objet comme le montre la Figure 3.31.

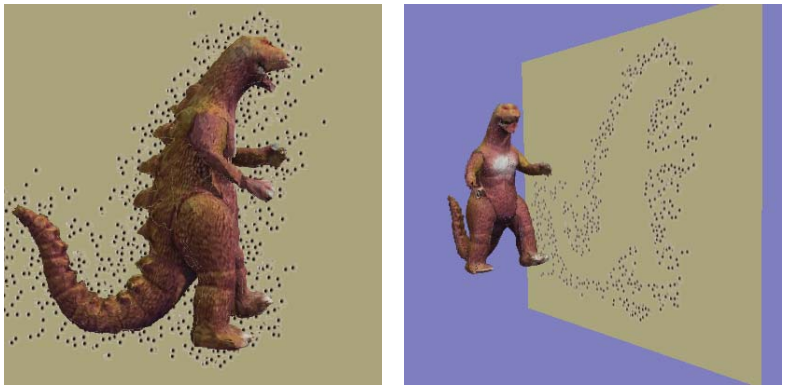


FIG. 3.31 – Intersection rayon/nuage de *billboards*

3.9.3 Application de l'algorithme de recherche de plans

Nous pensons en outre que notre méthode originale de recherche de plans principaux dans un ensemble 3D de primitives peut s'avérer intéressante pour d'autres applications que celles envisagées ici. Inspirée des travaux de Hough pour la recherche de lignes dans des images, elle pourrait être utilisée en vision pour de la reconnaissance de formes ou de modèles. Elle pourrait aussi servir pour des algorithmes de clusterisation, notamment pour des calculs de radiosité hiérarchique [Tur02]. La méthode est également applicable à des ensembles non structurés de points et nous pouvons envisager de l'utiliser pour guider des méthodes de reconstruction requises par exemple par les digitaliseurs 3D ou en vision [FHP83].

3.10 Extension au cas dépendant du point de vue

Dans cette section, nous adaptons l'algorithme de construction d'un nuage de *billboards* au cas dépendant du point de vue, pour lequel nous minimisons l'erreur par rapport à une cellule volumétrique. Les accélérations que permettent le rendu à base d'image dépendantes du point de vue ont été mises en évidence dans une grande variétés de situations [SDB97, DSSD99, Ali96, SLS+96]. Dans la lignée de ces approches, nous utilisons un *point de vue de référence* dans la cellule pour calculer les textures des *billboards*, et pour définir la notion de validité. Nous choississons de construire le nuage de *billboards* de telle sorte que ce nuage soit exact vu du point de référence, et que l'erreur commise en tout autre point soit bornée. Les nuages de *billboards* sont l'aboutissement d'une réflexion menée depuis longtemps sur la simplification à base d'image. Les premiers travaux que nous avions menés avaient conduit aux imposteurs multi-couches [DSSD99] dont la Figure 3.32 montre un exemple. Déjà, il s'agissait de représenter une partie distante du modèle par un ou plusieurs polygones texturés placés de manière à minimiser l'erreur pour tout point de vue dans une région donnée. La méthode était limitée à des cellules rectilignes (portions de rue), et n'utilisait que partiellement la

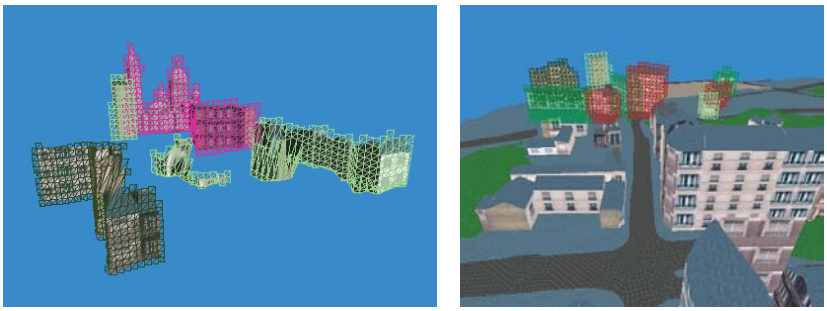


FIG. 3.32 – Imposteurs multi-couches

transparence sur la bordure de chaque couche d'imposteur. Le relief était restitué par l'utilisation de panneaux 2D et demi, là où les nuages de *billboard* utilisent un enchevêtrement et la transparence. Les imposteurs multi-couches peuvent être cependant vus comme un cas particulier de nuage de *billboards*, où chaque patch rectangulaire du maillage de l'imposteur est un *billboard*. Récemment, Wimmer *et al* ont justement proposé (non publié pour l'instant) une méthode originale qui construit un imposteur très détaillé, puis lui applique un algorithme de simplification de maillage, c'est-à-dire d'une certaine façon, cherche les ensembles de facettes quasi-coplanaires. La méthode que nous proposons utilise le formalisme introduit dans cette thèse pour construire directement une ensemble adéquat de plans.

Pour cela, nous dérivons d'abord une formule pour les domaines de validité (section 3.10.1). Nous discutons ensuite le choix du point de vue de référence. Nous présentons notamment une analyse montrant comment ce point doit être choisi, et pourquoi. Enfin nous montrons quelques résultats.

3.10.1 Domaine de validité

Nous considèrons donc une cellule, et un point de vue de référence V dans cette cellule. Dans le cas indépendant du point de vue, nous autorisions les sommets à se déplacer dans une sphère autour de leurs positions originales. C'est ce que nous avons appelée le domaine de validité du sommet. Ici, nous imposons que la vue d'un objet simplifiée soit équivalente à celle de l'objet non simplifié pour le point de vue de référence. Plus précisément, un sommet M donné du modèle original et son équivalent dans la version simplifiée doivent se projeter sur le même pixel. Pour cela, le sommet M doit être simplifié le long de (VM). Si M est simplifié en P, l'erreur de reprojection pour un point de vue T dans la cellule est définie comme l'angle solide sous lequel $[MP]$ est vu de T (Figure 3.33). Cette erreur est maximale quand (MT) est tangent à la cellule et MT maximum. Le segment $[P_-P_+]$ dans lequel M peut être déplacé, c'est-à-dire le domaine de validité de M,

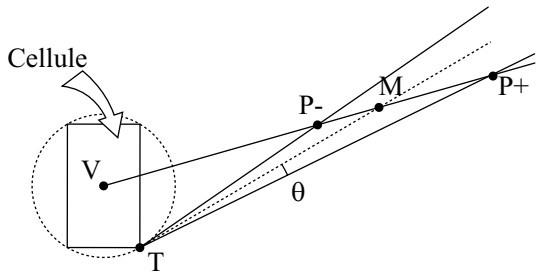


FIG. 3.33 – Erreur de reprojection

pour une erreur de reprojection inférieure à θ_{max} est donné par :

$$VP_{\pm} = \frac{VM}{a \pm \tan\theta \sqrt{\frac{VM^2}{VT^2} - 1}} \tag{3.13}$$

Pour convertir θ_{max} en une distance en pixels dans l'espace image, nous utilisons une linéarisation de l'angle solide θ en fonction de la longueur en pixels :

$$\tan\theta = \frac{1}{2n} \times \frac{w}{W} \times e \tag{3.14}$$

où n est la distance du plan de la caméra au centre de projection (*near plane*), w la largeur de l'écran en unité du monde et W sa largeur en pixels.

La contribution d'une face est définie comme l'aire projetée dans la vue de référence. Cela favorise les faces qui sont le plus visibles dans la vue de référence. Aucune pénalité n'est pour l'instant prise en compte. Le domaine de validité et la contribution étant définie, la recherche des plans est effectuée comme précédemment avec un algorithme glouton guidé par une fonction de densité.

La création des textures est un un peu différente. Au lieu d'utiliser une projection orthogonale, nous utilisons une projection perspective avec pour centre le point de référence de la cellule. La résolution de texture dépend de la valeur de *viewport* choisie ; cette dernière est spécifiée par l'utilisateur et doit idéalement être supérieure à la valeur de *viewport* utilisée lors de la visualisation. La matrice correspondant à cette projection doit également être stockée et utilisée comme matrice de texture lors du rendu des *billboards* pour corriger la distorsion perspective.

3.10.2 Choix du point de référence

Nous avons choisi le centre de la cellule comme point de vue de référence. Ce choix est celui couramment fait dans les techniques de rendu à base d'images. Nous allons maintenant montrer que ce point est optimal dans le cas général quand les sommets simplifiés sont suffisamment nombreux et relativement loins de la cellule. Nous montrerons ensuite que dans la plupart des cas, lorsque la partie du modèle simplifiée est située « du même côté de la cellule », il existe un meilleur choix.

Rappelons notre problème. Étant donné un point M, nous cherchons le « meilleur » plan sur lequel simplifier M. Contrairement au cas indépendant du point de vue, la notion de « meilleur » fait intervenir ici le plan *et* le point de vue de référence, qui définit comment M est simplifié sur un plan donné et qui entre en compte dans le calcul de l'erreur de reprojection. Nous allons étudier l'influence de ces deux paramètres. Nous allons notamment montrer que dans les cas qui nous intéressent (modèle éloigné de la cellule) ces deux paramètres sont décorrélés : on peut choisir un même point de référence pour tous les sommets à simplifier, et ensuite effectuer la recherche des plans sur lesquels simplifier ces sommets.

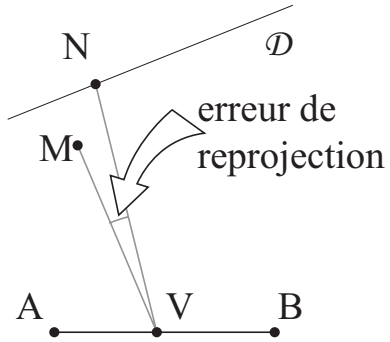


FIG. 3.34 – Erreur de reprojection

Nous considèrons d'abord une cellule qui est un segment. Soit donc un segment $[AB]$, un point M et une droite $\mathcal{D}$ du plan. Nous voulons remplacer M par un point N de $\mathcal{D}$ de manière à minimiser l'erreur de reprojection commise en tout point V de $[AB]$. Cette erreur est l'angle $\widehat{MVN}$ comme indiqué sur la Figure 3.34. Pour un N choisi, elle est clairement maximale pour $V = A$ ou $V = B$. On a le lemme suivant :

Lemme 1 *Une condition nécessaire et suffisante pour qu'un point N minimise l'erreur de reprojection est que l'on ait l'égalité angulaire* $\widehat{MAN} = \widehat{MBN}$.

►**preuve:** On munit la droite $\mathcal{D}$ d'une abscisse x. Soit A' l'intersection de $\mathcal{D}$ et de (AM). Traçons la courbe de la fonction f_A qui donne la mesure de l'angle $\widehat{MAN}$ fonction de x_N. Elle a le profil de la Figure 3.35. Cette fonction est continue, monotone sur $]-\infty, x_{A'}]$ $[x_{A'}, +\infty[$ et vaut 0 en $x_{A'}$. De même on construit B' et f_B. On suppose sans perte de généralité que l'on a $x_{A'} < x_{B'}$. Par continuité, il existe $x_N \in]x_{A'}, x_{B'}[$ tel que $f_A(x_N) = f_B(x_N)$. Étant donné les monotonies de f_A et de f_B, pour tout $x \neq x_N$ on a soit $f_A(x) > f_A(x_N)$ soit $f_B(x) > f_B(x_N)$. Soit N le point d'abscisse x_N, il minimise donc l'erreur de reprojection maximale, et les angles $\widehat{MAN}$ et $\widehat{MBN}$ sont égaux. CQFD. ◄

Nous cherchons donc à construire le lieu des points P tels que $\widehat{MAP} = \widehat{MBP}$. Il s'agit de l'hyperbole de diamètre $[AB]$ passant par M dans le cas général (voir théorème n°515, p334 dans [LH63]), et la médiatrice de $[AB]$ dans le cas particulier

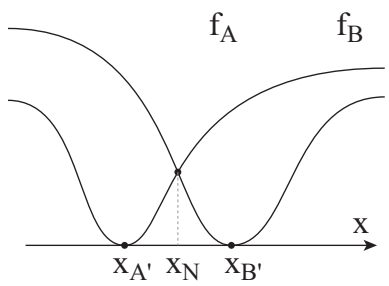


FIG. 3.35 – Profils des angles

où M est sur cette médiatrice. Le point N qui minimise l'erreur de reprojection se trouve à l'intersection de cette hyperbole et de la droite $\mathcal{D}$. Pour déterminer ce point, nous cherchons l'équation de l'hyperbole dans un repère (O, x, y) centré sur le milieu de $[AB]$.

Lemme 2 *L'équation dans un repère orthonormé (O, x, y) d'une hyperbole équilatère admettant O pour centre de symmétrie est de la forme :*

$$\sin 2\theta\,(x^2 - y^2) - 2xy\,\cos 2\theta + 2A = 0$$

où θ et A sont deux valeurs à déterminer.

►**preuve:** Comme l'hyperbole est équilatère et symmétrique autour de O, il existe un repère (O, X, Y) tourné de θ par rapport à (O, x, y) dans lequel l'hyperbole a pour équation $XY = A$. Le résultat est immédiat en effectuant dans cette équation le changement de coordonnées :

$$\begin{aligned} X &= \cos\theta\, x + \sin\theta\, y \\ Y &= -\sin\theta\, x + \cos\theta\, y \end{aligned}$$

et en utilisant les formules trigonométriques classiques. ◄

Pour trouver l'équation d'une hyperbole équilatère passant par deux points P et P', il suffit de résoudre en $(u, v, w) = (\sin 2\theta, \cos 2\theta, 2A)$ le système :

$$\left\{ \begin{array}{rcrcrcl} (x^2 - y^2)u & - & 2xy\,v & + & w & = & 0 \\ (x'^2 - y'^2)u & - & 2x'y'\,v & + & w & = & 0 \\ & & & & u^2 + v^2 & = & 1 \end{array} \right.$$

En faisant la différence des deux premières équations et en injectant le résultat dans la troisième, il vient :

$$\sin 2\theta = -\frac{b' - b}{\Delta},\ \cos 2\theta = +\frac{a' - a}{\Delta} \text{ et } 2A = \frac{\begin{vmatrix} a & a' \\ b & b' \end{vmatrix}}{\Delta}$$

avec :

$$\begin{array}{rclcrcl} a & = & x^2 - y^2 & \text{et} & a' & = & x'^2 - y'^2 \\ b & = & -2xy & \text{et} & b' & = & -2x'y' \\ \Delta & = & \sqrt{(a'-a)^2 + (b'-b)^2} & & & & \end{array}$$

Un calcul élémentaire montre que $\Delta = 0$ si et seulement si P et P' sont symétriques. L'hyperbole que l'on cherche est donc entièrement déterminée par la donnée de M et de A (ou de B).

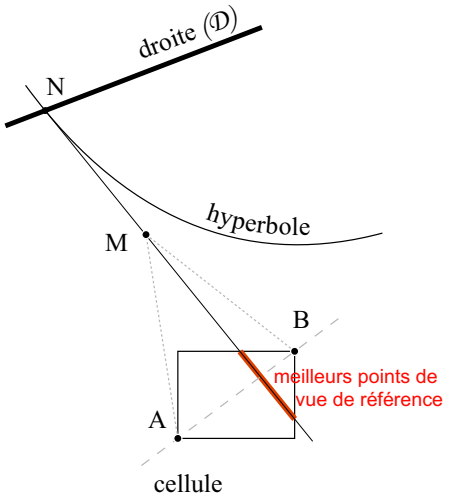


FIG. 3.36 – Meilleurs points de vue de référence

Nous considérons maintenant une cellule de forme quelconque, par exemple un rectangle. Le point M « voit » cette cellule sous un certain segment $[AB]$ (voir Figure 3.36), ce qui permet de nous ramener au cas que nous venons d'étudier. Il est en effet clair que c'est en ces deux points que l'erreur de reprojection est maximale. On peut déterminer, par intersection de l'hyperbole équilatère passant par A, B et M et de la droite $\mathcal{D}$ le point N où il convient le mieux de déplacer M. Choisir de simplifier M sur ce point N, c'est imposer au point de vue de référence de se trouver dans l'intersection de la droite (MN) et de la cellule (en rouge sur la Figure 3.36).

Le meilleur point de vue de référence dépend donc *a priori* du plan sur lequel un point doit être simplifié. Cependant, lorsque le point M et le plan en question sont assez éloignés de la cellule, la droite (MN) se confond avec une des asymptotes de l'hyperbole. L'intersection de (MN) et de la cellule ne dépend alors plus du choix du plan (qui ne fait que déplacer N). Ce résultat justifie que l'on choisisse un point de vue de référence, et que l'on effectue ensuite l'optimisation gloutonne pour déterminer les plans de simplification.

En outre, le meilleur point de vue de référence est *a priori* défini pour chaque sommet du modèle. Nous avons vu que c'était un segment, différent pour chaque sommet. Or lorsque les sommets sont éloignés de la cellule, on constate sur la

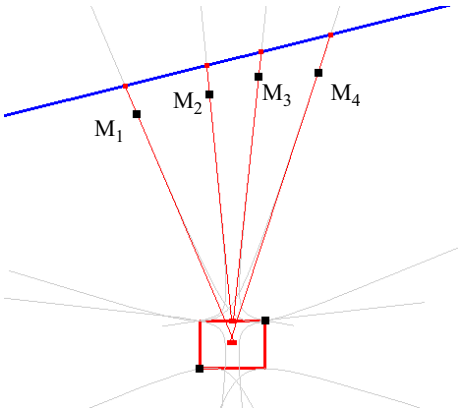


FIG. 3.37 – Choix du centre de la cellule

Figure 3.37) que ces segments passent tous approximativement par le centre de la cellule. Ce qui justifie le choix de ce point comme point de référence commun à tout les sommets. On constate aussi que si tout les point sont situés au loin sur le « devant » de la cellule, un autre bon choix est le milieu du côté avant (le plus près des points) de la cellule. Ceci tend à montrer que dans une méthode à base d'image, les images devraient être prises à partir du point de la cellule qui est le plus proche du modèle qu'elle vont remplacer. Ceci correspond à l'intuition : il vaut mieux faire une photo de près, donc comportant beaucoup de détails, et l'utiliser de loin, que le contraire. Dans les imposteurs de Sillion *et al* [SDB97] par exemple, ce n'est pas ce qui est fait. Les cellules sont des lignes, et la texture qui habille un imposteur est une image prise *en entrant* dans la cellule. La conséquence est notamment un manque de résolution lorsque l'on se rapproche de l'image, une vue exacte quand on est loin et distordue quand on est près (alors que l'on voudrait le contraire) et des effets d'élongation (*rubber sheet triangles*). Notre analyse suggère qu'en calculant l'imposteur à partir du *point de sortie* de la cellule, ces artefacts seraient réduits.

3.10.3 Résultats

Nous avons implémenté une version indépendante du point de vue. Ces travaux sont préliminaires : notre but était de montrer la versatilité de la représentation par nuage de *billboards* et de vérifier la validité de la fonction de densité dans le cas dépendant du point de vue. La Figure 3.38 montre que l'algorithme se comporte bien comme nous le souhaitions : plus la cellule est éloignée de l'objet, moins il faut de *billboards* pour le représenter.

Cependant, il reste encore de nombreuses choses à explorer pour pouvoir utiliser ces résultats dans une application. La construction d'un nuage par cellule pose le problème du coût de stockage : il faut stocker une représentation de l'objet par cellule. Il y a aussi le problème de la transition entre les représentations de deux cellules voisines. Cependant, la piste dépendante du point de vue nous paraît très

(a) 20 billboards (b) 3 billboards (c) 1 billboard

FIG. 3.38 – Simplification dépendante du point de vue

La rangée du haut montre une vue côte-à-côte du modèle polygonal (à droite) et du nuage de billboards (montré en dessous de près et sous un angle différent), comme vu de la cellule, avec une cellule placée de plus en plus loin de l'objet.

intéressante car dans le cas où l'objet est placé au loin, nous obtenons une simplification extrême (un seul *billboard*) qui a pourtant une qualité visuelle très bonne. En outre, il existe un point, le point de vue de référence, pour lequel les *billboards* sont équivalents au modèle. Les problèmes de stokage et de transition que nous venons d'indiquer sont principalement liés au fait que les cellules sont choisies indépendamment des objets (il peut par exemple s'agir des cellules utilisées pour un pré-calcul de visibilité comme expliquée au chapitre 1). D'une part il peut y en avoir un grand nombre, et d'autre part pour un objet donné, beaucoup de cellules sont redondantes. Imaginez par exemple plusieurs petites cellules adjacentes situées loin de l'objet. Il paraît inutile de calculer un (probablement identique) nuage pour chacune d'entre elles. Nous aimerions donc explorer la piste de cellules calculées par objet, en fonction par exemple d'une orientation et d'une distance, comme indiquée sur la Figure 3.39.

3.11 Travaux futurs envisagés

Afin d'améliorer la qualité des nuages de *billboards*, d'optimiser leur fabrication et leur usage, nous souhaitons poursuivre nos investigations sur cette nouvelle représentation. Voici plusieurs pistes que nous envisageons d'explorer.

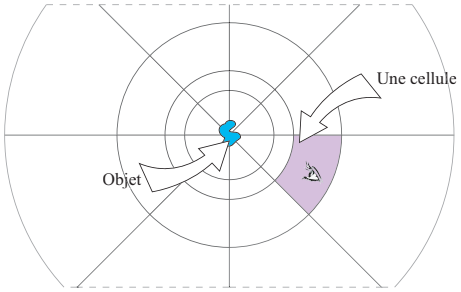


FIG. 3.39 – Cellules basées sur les objets

Relaxations pour optimiser l'usage des textures

Dans l'exposé qui vient d'être fait, nous avons considéré que le coût d'un nuage de *billboards* étaient uniquement lié au nombre de plans. Or, comme nous l'avions dit en introduction, il faut aussi tenir compte de la taille des textures. Certaines d'entre elles peuvent notamment être assez large mais ne contenir que peut de pixels opaques. C'est le cas par exemple si l'on simplifie une scène composée de deux bâtiments alignés et placés à une certaine distance. Un plan va certainement être crée pour approximer les deux facades. La texture résultante contiendra deux composantes correspondant aux deux bâtiments, avec un grand espace transparent entre les deux, espace d'autant plus grand que les bâtiments sont éloignés. Dans un tel cas, nous souhaiterions scinder le *billboard* en deux *billboards* coplanaires mais associés à des textures beaucoup plus optimales. Cela peut-être fait en post-traitement à l'issue de l'algorithme que nous avons présenté, mais nous aimerions envisager la prise en compte de ce paramètre dans l'algorithme glouton qui répartit les faces sur différents plans de simplification.

Densités stochastiques

Nous avons vu que la densité mesure la quantité de faces qui sont approximables par un plan donné. Cette densité est utilisée pour guider un algorithme glouton de regroupement de faces sur un *billboard*. Elle est évaluée en accumulant les contributions de chaque face du modèle. Supposons qu'on calcule la fonction de densité pour un modèle donné, et pour le même modèle où chaque face a été artificiellement subdivisée. Les fonctions de densité seront théoriquement les mêmes. Partant de cette observation, nous voudrions essayer un calcul de la densité *stochastique*, qui ne prendrait en compte que les contributions de faces choisies aléatoirement. Si le modèle est finement tesselé, la densité obtenue devrait être assez proche et conduire l'algorithme glouton aux mêmes résultats, tout en étant plus rapide. L'algorithme glouton (recherche explicite d'un plan valide) et la construction des textures utiliseraient toujours l'ensemble des faces. Seul le calcul initial de la densité utiliserait un échantillon de faces. On pourrait cependant imaginer que l'algorithme glouton n'utilise lui aussi qu'un échantillon aléatoire des faces. À la fin,

une fois tous les plans trouvés, on essaye de placer les faces restantes sur les plan trouvés pour les faces échantillons. La probabilité est certainement grande qu'il soit possible de placer ainsi toutes les faces sans pour autant avoir à les prendre en compte dans l'algorithme glouton.

Patchwork et compression des textures

Pour l'instant, les nuages de *billboards* que nous fabriquons sont stockés sous forme d'une liste de quadrilatères et de textures associées. Lors du rendu, il y a donc de nombreux changements de contexte liés aux changement de texture. Nous voudrions essayer différents algorithmes permettant de regrouper toutes les textures en une seule grande texture, à la manière d'un *patchwork*, comme le faisaient Cignoni *et al* [CMSR98]. Pour cela, il faut une méthode pour regrouper de la manière la plus compacte les textures, et modifier les coordonnées textures de manière adéquate. En raison de la transparence, il faut aussi s'assurer que le *mip-mapping* ne produira pas d'artefacts à la jonction entre deux textures.

L'utilisation de *billboards* augmente le nombre de textures utilisées pour une scène. Or la mémoire texture disponible sur les cartes graphiques est limitée. Les nouvelles cartes graphiques offrent la possibilité de travailler avec des textures compressées et d'augmenter ainsi le nombre de textures disponibles simultanément. Nous souhaiterions évaluer le nombre d'objets représentés par des *billboards* que cela nous permettrait de placer dans une scène.

Réeclairage de nuage de *billboards*

Une des limitations immédiatement visible des *billboards*, commune à toutes les méthodes de rendu à base d'image type « imposteurs », est l'impossibilité de rééclairer la représentation obtenue. En effet, l'éclairage tel qu'il est typiquement fait en OpenGL nécessite une information de normale par sommet. Cette information, et la notion même de sommet, disparaît dans la représentation que nous avons proposée. Dans l'exemple de la Figure 3.26 par exemple, les textures ont été générées en activant l'éclairage et donnent donc un sentiment de relief. Mais les ombres correspondent aux conditions d'éclairage lors de la génération des textures. Si l'on change l'éclairage dans l'application utilisant les *billboards*, ces ombres ne seront plus du tout correctes. Cependant, les nouvelles cartes graphiques offrent une fonctionnalité tout à fait intéressante qui permettrait de pallier à cet inconvénient. Il s'agit des *pixels shaders*, c'est-à-dire de la possibilité de spécifier des fonctions par pixel lors du rendu. Nous envisageons alors d'augmenter nos *billboards* avec une carte de normales (les normales sont des vecteurs dont les 3 composantes peuvent être codées sur les canaux RVB d'une image) et d'utiliser un procédé de *bump-mapping* pour restituer *dynamiquement* des effets d'éclairage sur des textures qui ont été calculées *statiquement*. Des essais montrent qu'on peut ainsi restituer les effets diffus et même spéculaires de façon très efficace.

Nuages de *billboards* temporels

Puisqu'on peut texturer des polygones avec des textures, l'idée vient assez naturellement de les texturer avec des séquences vidéos. Certaines API comme Performer [RH94] le proposent déjà. Il serait intéressant de pouvoir générer automatiquement des nuages de *billboards* dont certaines textures sont animées. Imaginons par exemple un avion dont les hélices sont en rotation. Pour cela, l'algorithme glouton devra tenir compte des parties qui bougent pour les regrouper ensemble et limiter le nombre de textures qui devront être animées.

La vie est faite d'illusions. Parmi ces illusions, certaines réussissent. Ce sont celles qui constituent la réalité.

Jacques Audiberti

Conclusion

Cette thèse a travaillé sur le problème de la visualisation de grosses bases de donnée représentant des environnements virtuels. Nous avons étudié les possibilités de pré-calculer des résultats qui pourraient être utilisées pour accélérer ensuite la visualisation et proposons maintenant un résumé de nos contributions.

Nous avons tout d'abord proposé une nouvelle méthode de pré-calcul de visibilité. Un tel pré-calcul consiste à déterminer, pour un ensemble fini de régions recouvrant les points de vues possibles, les objets potentiellement visibles de chaque région. Toute la difficulté réside dans le calcul *pour une région* : si l'on sait désormais très bien déterminer ce qui est visible en un point, il est beaucoup plus dur de déterminer ce qui est visible d'une région. Notre méthode s'appuie sur un résultat théorique, première contribution de cette thèse, qui permet d'effectuer un calcul de visibilité en un point et d'obtenir une réponse valable pour une région autour de ce point. Ce résultat s'appuie sur le théorème de la réduction simultanée des bloqueurs et des bloqués par un élément structurant convexe, que nous avons démontré et qui généralise une propriété initialement exhibée par Wonka et Wimmer [WWS00]. Le théorème dit en substance que pour qu'un objet soit caché d'une région par un ensemble d'objets, il suffit que sa *réduction* soit cachée par les *réductions* des objets en un point fixé de la région ; la réduction étant définie comme la différence de Minkowski d'un objet et de la région. Le théorème étend la propriété montrée par Wonka et Wimmer de deux façons. D'une part en généralisant la réduction par une boule, à la réduction par un élément convexe. Cela permet de choisir des régions de visibilité adaptées. Dans le cadre d'une ville, par exemple, où l'élévation du point de vue est quasiment constante (hauteur des yeux), le choix de cellules sphériques entraîne une réduction inutile dans la direction verticale. Le choix de cellules parallélépipédiques plus larges sera plus adapté. D'autre part, le théorème montre que l'on peut réduire non seulement les bloqueurs comme le font Wonka et Wimmer, mais aussi les éléments dont on teste si ils sont cachés par les bloqueurs. Les occlusions détectées sont ainsi plus fines, notamment lorsque l'on utilise une région de grande taille, c'est-à-dire une forte réduction.

Pour pouvoir utiliser le théorème ci-dessus, il faut être capable de calculer les réductions des objets. La deuxième contribution de cette thèse est un algorithme robuste et efficace qui permet de calculer une approximation de ces réductions. L'algorithme procède en voxelisant les objets, reprenant là l'idée des travaux de Schaufler *et al* [SDDS00] selon laquelle c'est le volume décrit par un modèle po-

lygonal qui est intéressant. Puis il calcule de manière robuste la différence de Minkowski de l'ensemble de voxels obtenu et d'un cube de côté quelconque. Il est important de noter que la taille du cube est *indépendante* de la taille des voxels. Dans l'application au calcul de visibilité, cela permet de décorréler le choix du niveau de voxelisation du choix des cellules.

Muni du théorème de réduction et d'un algorithme d'approximation, nous avons proposé un méthode de pré-calcul de visibilité adaptée à de larges environnements. La méthode permet théoriquement de traiter des modèles d'étendue spatiale quelconque car elle ne nécessite pas de structure de calcul portant sur le modèle *en entier*. Au contraire, elle travaille *objet par objet*, en utilisant l'espace image pour calculer de manière implicite les occlusions et réaliser de façon générale la fusion des pénombres. L'implémentation de la méthode est relativement simple, très robuste, et tire profit des fonctionnalités des nouvelles cartes graphiques. Notre implémentation nous permet ainsi de pré-calculer dans un temps très raisonnable -quelques heures- la visibilité pour un large modèle. En outre la méthode présente l'avantage de pouvoir être accélérée de multiples façons en utilisant des méthodes de calcul de visibilité en un point.

Après avoir déterminé ce qui est potentiellement visible, il reste à l'afficher le plus rapidement possible. La troisième contribution de cette thèse est une nouvelle représentation, baptisée nuage de *billboards*, qui permet de représenter assez fidèlement mais de façon simplifiée un objet ou une collection d'objets. Il s'agit d'un ensemble de rectangles texturés avec de la transparence et enchevêtrés de manière complexe pour rendre à la fois la forme et l'apparence de l'objet. Si un tel ensemble peut être construit à la main, nous décrivons un algorithme permettant de placer un nombre optimal de tels rectangles pour approximer un modèle polygonal quelconque. L'algorithme fournit un cadre pour la résolution du problème d'optimisation sous-jacent : trouver un nombre minimal de plans approximant un modèle donné tout en respectant une borne d'erreur et en minimisant une « distance » entre le modèle original et le modèle simplifié. Nous montrons deux spécialisations de cet algorithme, une indépendante du point de vue et une dépendante du point de vue ; mais d'autres peuvent certainement être mises au point en intégrant à la fonction de densité des paramètres spécifiques à certaines classes de modèles, et certaines conditions de visualisation.

L'idée que nous souhaitons faire ressortir de cette thèse est qu'il faut disposer de différentes représentations des données à traiter, d'algorithmes capables d'extraire une représentation d'une autre, et choisir la représentation la plus adaptée à là tâche effectuée. Nous montrons ainsi qu'une représentation par voxel du volume des objets permet un calcul efficace de ce qui est caché par cet objet. L'intérêt de la méthode proposée est sa robustesse et la possibilité de travailler à différentes résolutions de voxels. Nous avons également introduit une nouvelle représentation, le nuage de *billboards*, qui nous paraît intéressante pour plusieurs raisons. Tout

d'abord elle travaille dans un intervalle de simplification habituellement non géré par les méthodes classiques, le domaine de la simplification « extrême ». Partant d'un modèle pouvant contenir un grand nombre de primitives, elle introduit une représentation contenant très peu de primitives (de l'ordre de la centaine) et pourtant capable de restituer des effets complexes de parallaxe ou de silhouette. Dans les cas le pire, elle produit même une seule primitive dont la qualité est cependant très bonne pour un grand nombre de points de vue. Ensuite, les nuages de *billboards* permettent de travailler de manière unifiée sur la forme et sur l'apparence, et réalisent *in fine* un filtrage du modèle à une résolution choisie. Les nuages de *billboards* sont aussi utiles pour d'autres applications que le rendu d'un objet. Nous avons vu par exemple qu'ils permettaient de rendre efficacement des ombres de bonne qualité. Les nuages de *billboards* offrent enfin un certain nombre de perspective de recherche. L'algorithme de construction peut être optimisé et prendre en compte de nouveaux paramètres, notamment le coût des textures. Le cas dépendant du point de vue soulève différentes questions : combien d'objets peut-on représenter, comment se fait la transition entre deux nuages, entre un nuage et une représentation polygonale, comment peut-on compresser efficacement un nuage de *billboards*, le transmettre progressivement ?

Le problème de la représentation des données est une question beaucoup discutée aujourd'hui : si la représentation polygonale s'est imposée de par sa simplicité et l'existence de cartes dédiées, un certain nombre d'autres représentations -à base d'images, à base de points- ont récemment vu le jour, ou redeviennent d'actualité en raison des nouvelles capacités et de la puissance des machines désormais disponibles. Si ces méthodes sont en compétition, il nous paraît clair qu'aucune d'elles n'est fondamentalement « meilleure » que les autres. Avec le développement croissant des applications de l'informatique en général et de l'image de synthèse en particulier, les besoins deviennent très spécifiques. Il est alors important de bien comprendre quels sont les avantages et inconvénients des différentes représentations et méthodes de rendu disponibles. Or ces méthodes sont de plus en plus complexes, avec de nombreuses variations et raffinements mais rarement des implémentations commerciales. Nous pensons donc que le domaine arrive à une période de maturité et que pour préciser les directions d'évolution, il faudrait intégrer les différentes techniques existantes dans un système permettant l'utilisation, l'évaluation et la comparaison de ces techniques.

Bibliographie

[AB91] Edward H. Adelson et James R. Bergen. The Plenoptic Function and the Elements of Early Vision. Dans M. Landy et J. A. Movshon, éditeurs, *Computation Models of Visual Processing*, pages 3–20. MIT Press, Cambridge, 1991.

[AL97] Daniel G. Aliaga et Anselmo A. Lastra. « Architectural Walkthroughs Using Portal Textures ». Dans *Proceedings of the conference on Visualization '97*, pages 355–362. IEEE-CS press, octobre 1997.

[Ali96] Daniel G. Aliaga. « Visualization of Complex Models Using Dynamic Texture-based Simplification ». Dans *Proceedings of the conference on Visualization '96*, pages 101–106. IEEE-CS press, octobre 1996.

[AM00] Ulf Assarsson et Tomas Möller. « Optimized View Frustum Culling Algorithms for Bounding Boxes ». *Journal of Graphics Tools : JGT*, 5(1) :9–22, 2000.

[BBM+01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, et Michael Cohen. « Unstructured lumigraph rendering ». Dans *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432. ACM Press, 2001.

[BK97] Gill Barequet et Subodh Kumar. « Repairing CAD models ». Dans *Proceedings of the conference on Visualization '97*, pages 363–370. IEEE-CS Press, octobre 1997.

[Bli78] James F. Blinn. « Simulation of wrinkled surfaces ». *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3) :286–292, août 1978.

[BS95] Gill Barequet et Micha Sharir. « Filling gaps in the boundary of a polyhedron ». *Computer Aided Geometric Design*, 12(2) :207–229, 1995.

[BW00] Dmitry Brodsky et Benjamin Watson. « Model Simplification Through Refinement ». Dans *Graphics Interface*, pages 221–228, May 2000.

[BWW01] Jiří Bittner, Peter Wonka, et Michael Wimmer. « Visibility Preprocessing for Urban Scenes using Line Space Subdivision ». Dans

Proceedings of Pacific Graphics (PG'01), pages 276–284, Tokyo, Japan, 2001.

[Cat74] Edwin E. Catmull. « *A Subdivision Algorithm for Computer Display of Curved Surfaces* ». PhD thesis, Dept. of CS, U. of Utah, décembre 1974.

[CBL99] Chun-Fa Chang, Gary Bishop, et Anselmo Lastra. « LDI tree : a hierarchical representation for image-based rendering ». Dans *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 291–298. ACM Press/Addison-Wesley Publishing Co., 1999.

[CCST00] Jin-Xiang Chai, Shing-Chow Chan, Heung-Yeung Shum, et Xin Tong. « Plenoptic sampling ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 307–318. ACM Press/Addison-Wesley Publishing Co., 2000.

[Che95] Shenchang Eric Chen. « Quicktime VR - An Image-Based Approach to Virtual Environment Navigation ». Dans Robert Cook, éditeur, *SIGGRAPH 95 Conference Proceedings*, pages 29–38. Addison Wesley, aug 1995. held in Los Angeles, California, 06-11 August 1995.

[Cla76] James H. Clark. « Hierarchical geometric models for visible surface algorithms ». *Communications of the ACM*, 19(10) :547–554, 1976.

[CMO97] Jonathan Cohen, Dinesh Manocha, et Marc Olano. « Simplifying polygonal models using successive mappings ». Dans *Proceedings of the conference on Visualization '97*, pages 395–402. IEEE-CS Press, octobre 1997.

[CMSR98] P. Cignoni, C. Montani, R. Scopigno, et C. Rocchini. « A general method for preserving attribute values on simplified meshes ». Dans *Proceedings of the conference on Visualization '98*, pages 59–66. IEEE-CS Press, octobre 1998.

[COM98] Jonathan Cohen, Marc Olano, et Dinesh Manocha. « Appearance-preserving simplification ». Dans *Proceedings of SIGGRAPH '98, Computer Graphics* Proceedings, Annual Conference Series, pages 115–122. ACM SIGGRAPH, août 1998.

[Coo84] Robert L. Cook. « Shade trees ». Dans *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231, 1984.

[CPD+96] Andrew Certain, Jovan Popović, Tom Duchamp, David Salesin, Werner Stuetzle, et Tony DeRose. « Interactive Multi-Resolution Surface Viewing ». Dans Holly Rushmeier, éditeur, *Proceedings of*

the 23rd annual conference on Computer graphics and interactive techniques, pages 91–97. ACM Press, août 1996.

[CS81] M. Cosman et R. Schumacker. « System Strategies to Optimize CIG Image Content ». Dans *Proceedings of the Image II Conference*, Scottsdale, Arizona, 10-12 June 1981.

[CVM+96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, et William Wright. « Simplification envelopes ». Dans *Proceedings of SIGGRAPH '96, Computer Graphics* Proceedings, Annual Conference Series, pages 119–128. ACM SIGGRAPH, août 1996.

[CW93] Shenchang Eric Chen et Lance Williams. « View Interpolation for Image Synthesis ». Dans James T. Kajiya, éditeur, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, AUG 1993.

[CZ98] Gadi Fibich Dan Halperin Cohen-Or, Daniel et Eyal Zadicerio.. « Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes ». *Computer Graphics Forum*, 17(3) :243–255, aug 1998. Proceedings of Eurographics '98. ISSN 0167-7055.

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, et Otfried Schwarzkopf. *Computational Geometry : Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd édition, 2000.

[DDTP00] Frédo Durand, George Drettakis, Joëlle Thollot, et Claude Puech. « Conservative Visibility Preprocessing Using Extended Projections ». Dans Kurt Akeley, éditeur, *Computer Graphics Proceedings*, Annual Conference Series, pages 239–248. ACM Press / ACM SIGGRAPH, 2000. Annual Conference Series, SIGGRAPH'00.

[DH72] Richard O. Duda et Peter E. Hart. « Use of the Hough transformation to detect lines and curves in pictures ». *Communications of the ACM*, 15(1) :11–15, 1972.

[DHT+00] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, et Mark Sagar. « Acquiring the reflectance field of a human face ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 145–156. ACM Press/Addison-Wesley Publishing Co., 2000.

[DJZ91] Michael J. DeHaemer Jr. et Michael J. Zyda. « Simplification of objects rendered by polygonal approximations ». *Computers and Graphics*, 15(2) :175–184, 1991.

[DMS01] Laura Downs, Tomas Möller, et Carlo H. Séquin. « Occlusion Horizons for Driving through Urban Scenery ». Dans *Symposium on Interactive 3D Graphics*, pages 121–124, March 2001.

[Dow93] Kevin Dowd. *High performance computing*. O'Reilly & Associates, Inc., 1993.

[DS02] Xavier Décoret et François Sillion. « Street Generation for City Modelling ». Dans *Architectural and Urban Ambient Environment*, 2002.

[DSSD99] Xavier Décoret, François Sillion, Gernot Schaufler, et Julie Dorsey. « Multi-layered impostors for accelerated rendering ». *Computer Graphics Forum*, 18(3) :61–73, septembre 1999. ISSN 1067-7055.

[DTM96] Paul E. Debevec, Camillo J. Taylor, et Jitendra Malik. « Modeling and rendering architecture from photographs : a hybrid geometry-and image-based approach ». Dans *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM Press, 1996.

[Dur99] Frédo Durand. « *Visibilité tridimensionnelle : Etude analytique et applications* ». PhD thesis, Université Joseph Fourier (Grenoble), Jul 1999.

[EDD+95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, et Werner Stuetzle. « Multiresolution analysis of arbitrary meshes ». Dans *Proceedings of SIGGRAPH '95, Computer Graphics* Proceedings, Annual Conference Series, pages 173–182. ACM SIGGRAPH, 1995.

[EM99] Carl Erikson et Dinesh Manocha. « GAPS : general and automatic polygonal simplification ». Dans *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 79–88. ACM Press, 1999.

[Eri96] Carl Erikson. « Polygonal Simplification : An Overview ». Rapport Technique TR96-016, 16, 1996.

[FHP83] Olivier Faugeras, M. Hebert, et E. Pauchon. « Segmentation of range data into planar and quadratic patches ». Dans *Proceedings of IEEE Conf. on Computer Vision and Pattern Recognition*, 1983.

[FST92] Thomas A. Funkhouser, Carlo H. Sequin, et Seth J. Teller. « Management of large amounts of data in interactive building walkthroughs ». *Computer Graphics (Proc. Symposium on Interactive 3D Graphics'92)*, 25(2) :11–20, mars 1992. held in Boston ; 29 March - 1 April 1992.

[FvDFH90] James D. Foley, Andries van Dam, Steven Feiner, et John Hughes. *Computer Graphics : Principles and Practice*. Addison-Wesley, Reading, Mass., 1990.

[GBW90] B. Garlick, D. Baum, et J. Winget. « Interactive viewing of large geometric data bases using multiprocessor graphics workstations ». Dans *Parallel Algorithms and Architectures for 3D Image Generation*, 1990. Siggraph '90 Course Notes.

[GCS91] Ziv Gigus, John Canny, et Raimund Seidel. « Efficiently computing and representing aspect graphs of polyhedral objects ». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6) :542–551, 1991.

[GDCV98] Jonas Gomes, Lucia Darsa, Bruno Costa, et Luiz Velho. *Warping and morphing of graphical objects*. Morgan Kaufmann Publishers Inc., 1998.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, et Michael F. Cohen. « The Lumigraph ». Dans Holly Rushmeier, éditeur, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, août 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[GH95] Daniel Green et Don Hatch. « Fast Polygon-cube Intersection Testing ». Dans *Graphics Gems V*, pages 375–379. Paul Heckbert, 1995.

[GH97] Michael Garland et Paul S. Heckbert. « Surface simplification using quadric error metrics ». Dans *Proceedings of SIGGRAPH '97, Computer Graphics* Proceedings, Annual Conference Series, pages 209–216. ACM SIGGRAPH, août 1997.

[GH98] Michael Garland et Paul S. Heckbert. « Simplifying surfaces with color and texture using quadric error metrics ». Dans *Proceedings of the conference on Visualization '98*, pages 263–269. IEEE-CS Press, octobre 1998.

[GJ79] Michael R. Garey et David S. Johnson. *Computers and Intractability : A Guide to the Theory of Np-Completeness*. W. H. Freeman & Co, juin 1979.

[GKM93] Ned Greene, Michael Kass, et Gavin Miller. « Hierarchical Z-buffer visibility ». Dans *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238. ACM Press, 1993.

[Gra] Silicon Graphics. « Programming with OpenGL : Advanced Rendering ». `http://www.sgi.com/software/opengl/advanced97/notes/node31.html`.

[Gre86] Ned Greene. « Applications of world projections ». Dans *Proceedings on Graphics Interface '86/Vision Interface '86*, pages 108–114. Canadian Information Processing Society, 1986.

[GWH01] Michael Garland, Andrew Willmott, et Paul S. Heckbert. « Hierarchical face clustering on polygonal surfaces ». Dans *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58. ACM Press, 2001.

[HAA97] Youichi Horry, Ken-Ichi Anjyo, et Kiyoshi Arai. « Tour into the picture : using a spidery mesh interface to make animation from a single image ». Dans *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 225–232. ACM Press/Addison-Wesley Publishing Co., 1997.

[Han89] Samet Hanan. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, août 1989.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, et Werner Stuetzle. « Surface reconstruction from unorganized points ». Dans *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78. ACM Press, 1992.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, et Werner Stuetzle. « Mesh optimization ». Dans *Proceedings of SIGGRAPH '93, Computer Graphics* Proceedings, Annual Conference Series, pages 19–26. ACM SIGGRAPH, août 1993.

[Hec89] Paul S. Heckbert. « Fundamentals of Texture Mapping and Image Warping ». Master's thesis, University of California, Berkeley, 1989.

[HG94] Paul Heckbert et Michael Garland. « Multiresolution modelling for fast rendering ». Dans *Proc. Graphics Interface 94*, pages 43–50, May 1994.

[HG97] Paul S. Heckbert et Michael Garland. « Survey of Polygonal Surface Simplification Algorithms ». Rapport Technique, 97.

[HH93] Paul Hinker et Charles Hansen. « Geometric Optimization ». Dans *Proceedings of the conference on Visualization '93*, pages 189–195, octobre 1993.

[HH97] Paul S. Heckbert et Michael Herf. « Simulating Soft Shadows with Graphics Hardware ». Rapport Technique CMU-CS-97-104, janvier 1997.

[HHK+95] Taosong He, Lichan Hong, Arie E. Kaufman, Amitabh Varshney, et Sidney W. Wang. « Voxel Based Object Simplification ». Dans *Proceedings of the conference on Visualization '95*, pages 296–303, octobre 1995.

[HKS98] W. Heidrich, J. Kautz, et Ph. Slusallek. « Canned Lightsources ». Dans Nelson Max George Drettakis, éditeur, *Rendering Techniques '98 (Proceedings of EG Rendering Workshop '98)*. Eurographics, 1998.

[Hoc96] Dorit Hochbaum, éditeur. *Approximations for NP-hard Problems*. PWS Publishing, Boston, MA, 1996.

[Hop96] Hugues Hoppe. « Progressive meshes ». Dans *Proceedings of SIGGRAPH '96, Computer Graphics* Proceedings, Annual Conference Series, pages 99–108. ACM SIGGRAPH, août 1996.

[Hop97] Hugues Hoppe. « View-dependent refinement of progressive meshes ». Dans *Proceedings of SIGGRAPH '97, Computer Graphics* Proceedings, Annual Conference Series, pages 189–198. ACM SIGGRAPH, août 1997.

[Hop99] Hugues Hoppe. « New quadric metric for simplifiying meshes with appearance attributes ». Dans *Proceedings of the conference on Visualization '99 : Celebrating ten years*, pages 59–66. IEEE-CS Press, octobre 1999.

[IMG00] Aaron Isaksen, Leonard McMillan, et Steven J. Gortler. « Dynamically reparameterized light fields ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 297–306. ACM Press/Addison-Wesley Publishing Co., 2000.

[KBGT96] Mike Krus, Patrick Bourdot, Franc̃coise Guisnel, et Guillaume Thibault. « Niveaux de Détails et Simplification Polygonale : un tour d'horizon ». Dans *AFIG'96, Quatrièmes Journées de l'Association Franc̃aise d'Informatique Graphique*, nov 1996.

[KBGT97] Mike Krus, Patrick Bourdot, Franc̃coise Guisnel, et Guillaume Thibault. « Levels of Detail and Polygonal Simplification ». *ACM's Crossroads*, 3.4, 1997.

[KCS98] Leif Kobbelt, Swen Campagna, et Hans-Peter Seidel. « A General Framework for Mesh Decimation ». Dans *Graphics Interface*, pages 43–50, 1998.

[KT96] A.D. Kalvin et R.H. Taylor. « Superfaces : Polygonal Mesh Simplification with Bounded Error ». *IEEE Computer Graphics and Applications*, 16(3) :64–77, May 1996.

[LC87] William E. Lorensen et Harvey E. Cline. « Marching cubes : A high resolution 3D surface construction algorithm ». Dans *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[LE97] David Luebke et Carl Erikson. « View-dependent simplification of arbitrary polygonal environments ». Dans Turner Whitted, éditeur, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 199–208. ACM Press/Addison-Wesley Publishing Co., août 1997. Held in Los Angeles, California.

[LH63] Camille Lebossé et Corentin Hémery. *GÉOMETRIE : Classe de Mathématiques moderne*. Fernand Nathan, Paris, 1963.

[LH96] Marc Levoy et Pat Hanrahan. « Light Field Rendering ». *Computer Graphics*, 30(Annual Conference Series) :31–42, 1996.

[Lin00] Peter Lindstrom. « Out-of-core simplification of large polygonal models ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 259–262. ACM Press/Addison-Wesley Publishing Co., 2000.

[Lou93] Michael Lounsbery. « *Multiresolution Analysis for Surfaces of Arbitrary Topological Type* ». Ph.D. thesis, Department of Computer Science and Engineering. Univerisity of Washington, Seattle, WA 98195-2350, 1993.

[LPC+00a] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, et Duane Fulk. « The digital Michelangelo project : 3D scanning of large statues ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144. ACM Press/Addison-Wesley Publishing Co., 2000.

[LPC+00b] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, et Duane Fulk. « The digital Michelangelo project : 3D scanning of large statues ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144. ACM Press/Addison-Wesley Publishing Co., 2000.

[LRC+02] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, et R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, first édition, july 2002.

[LS97] Jed Lengyel et John Snyder. « Rendering with Coherent Layers ». Dans Turner Whitted, éditeur, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 233–242. ACM SIGGRAPH, Addison Wesley, août 1997. ISBN 0-89791-896-7.

[LS01] P. Lindstrom et C. Silva. « A memory insensitive technique for large model simplification », 2001.

[LT97] Kok-Lim Low et Tiow-Seng Tan. « Model simplification using vertex-clustering ». Dans *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 75–ff. ACM Press, 1997.

[LT98] Peter Lindstrom et Greg Turk. « Fast and memory efficient polygonal simplification ». Dans *Proceedings of the conference on Visualization '98*, pages 279–286. IEEE-CS Press, octobre 1998.

[LT00] Peter Lindstrom et Greg Turk. « Image-driven simplification ». *ACM Transactions on Graphics*, 19(3) :204–241, 2000.

[Lue97] David Luebke. « A Survey of Polygonal Simplification Algorithms ». Rapport Technique TR97-045, Department of Computer Science, University of North Carolina, Chapel Hill, North Carolina, 1997.

[Mar95] Makoto Maruya. « Generating a texture map from object-surface texture data ». *Computer Graphics Forum (Proc. Eurographics'95)*, 14(3) :397–405,506–507, 1995.

[MB95] Leonard McMillan et Gary Bishop. « Plenoptic modeling : an image-based rendering system ». Dans *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46. ACM Press, 1995.

[Min03] H. Minkowski. « Volumen und Oberfläche ». *Mathematische Annalen*, 57 :447–495, 1903.

[MMB97] William R. Mark, Leonard McMillan, et Gary Bishop. « Post-rendering 3D warping ». Dans *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff. ACM Press, 1997.

[MN98] Alexandre Meyer et Fabrice Neyret. « Interactive Volumetric Textures ». Dans George Drettakis et Nelson Max, éditeurs, *Eurographics Rendering Workshop 1998*, pages 157–168, New York City, NY, Jul 1998. Eurographics, Springer Wein. ISBN.

[MS95] Paulo W. C. Maciel et Peter Shirley. « Visual Navigation of Large Environments Using Textured Clusters ». Dans *Symposium on Interactive 3D Graphics*, pages 95–102, 211, 1995.

[Mur83] F. Murtagh. « A survey of recent advances in hierarchical clustering algorithms ». *computer Journal*, 26(4) :354–359, 1983.

[OBM00] Manuel M. Oliveira, Gary Bishop, et David McAllister. « Relief texture mapping ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368. ACM Press/Addison-Wesley Publishing Co., 2000.

[OCDD01] Byong Mok Oh, Max Chen, Julie Dorsey, et Frédo Durand. « Image-based modeling and photo editing ». Dans *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 433–442. ACM Press, 2001.

[PD84] Thomas Porter et Tom Duff. « Compositing digital images ». Dans *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, janvier 1984.

[PD90] Harry Plantinga et Charles R. Dyer. « Visibility, occlusion, and the aspect graph ». *International Journal of Computer Vision*, 5(2) :137–160, 1990.

[PH97] Jovan Popović et Hugues Hoppe. « Progressive simplicial complexes ». Dans Turner Whitted, éditeur, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 217–224. ACM Press/Addison-Wesley Publishing Co., août 1997. Held in Los Angeles, California.

[PMS+99] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, et Charles Hansen. « Interactive ray tracing ». Dans *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 119–126. ACM Press, 1999.

[RB93] J. R. Rossignac et P. Borrel. « Multi-Resolution 3D Approximations for Rendering Complex Scenes ». Dans B. Falcidieno et T. L. Kunii, éditeurs, *Geometric Modeling in Computer Graphics*, pages 455–465, Genova, Italy, 1993. Springer-Verlag. Also published as technical report RC 17697 (#77951), IBM Research Division, T. J. Watson Research Center, 1992.

[RE00] D. Rose et T. Ertl. « Rendering Details on Simplified Meshes by Texture Based Shading ». Dans *Workshop on Vision, Modelling, and Visualization VMV '00*, pages 239–245. infix, 2000.

[RH94] John Rohlf et James Helman. « IRIS performer : a high performance multiprocessing toolkit for real-time 3D graphics ». Dans *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394. ACM Press, 1994.

[SBhD86] Francis J M Schmitt, Brian A. Barsky, et Wen hui Du. « An adaptive subdivision method for surface-fitting from sampled data ». Dans *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 179–188. ACM Press, 1986.

[SCG97] Peter-Pike J. Sloan, Michael F. Cohen, et Steven J. Gortler. « Time Critical Lumigraph Rendering ». Dans *Symposium on Interactive 3D Graphics*, pages 17–24, 181, 1997.

[Sch97a] Dieter Schmalstieg. « A Survey of Advanced Interactive 3-D Graphics Techniques ». Research paper, Institute of Computer Graphics, Vienna University of Technology, http ://visinfo.zib.de/EVlib/Show ?EVL-1997-17, 1997.

[Sch97b] William J. Schroeder. « A topology modifying progressive decimation algorithm ». Dans *Proceedings of the conference on Visualization '97*, pages 205–ff. IEEE-CS Press, octobre 1997.

[Sch98] Gernot Schaufler. « Per-Object Image Warping with Layered Impostors ». Dans George Drettakis andd Nelson Max, éditeur, *Eurographics Rendering Workshop 1998*, pages 145–156, Wien, juin 1998. Eurographics, Springer. ISBN 3-211-83001-4.

[SDB97] Franĉois Sillion, George Drettakis, et Benoit Bodelet. « Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery ». Dans D. Fellner et L. Szirmay-Kalos, éditeurs, *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16, Budapest, Hungary, septembre 1997.

[SDDS00] Gernot Schaufler, Julie Dorsey, Xavier Décoret, et François Sillion. « Conservative Volumetric Visibility with Occluder Fusion ». Dans Kurt Akeley, éditeur, *Computer Graphics Proceedings*, Annual Conference Series, pages 229–238. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. Annual Conference Series, SIGGRAPH'00.

[Sew97] Jonathan M. Sewell. « Managing complex models for computer graphics ». Rapport Technique TR420, Cambridge Computer Laboratory, 1997.

[SGG+00] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, et John Snyder. « Silhouette clipping ». Dans *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 327–334. ACM Press/Addison-Wesley Publishing Co., 2000.

[SGHS98] Jonathan W. Shade, Steven J. Gortler, Li-wei He, et Richard Szeliski. « Layered Depth Images ». Dans Michael Cohen, éditeur, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, jul 1998. ISBN 0-89791-999-8.

[SGR96] Marc Soucy, Guy Godin, et Marc Rioux. « A texture-mapping approach for the compression of colored 3D triangulations ». *Visual Computer*, 12 :503–514, 1996.

[She00] A. Sheffer. « Model simplification for meshing using face clustering ». *Computer-Aided Design*, 2000.

[SHS00] Hartmut Schirmacher, Wolfgang Heidrich, et Hans-Peter Seidel. « High-Quality Interactive Lumigraph Rendering Through Warping ». Dans *Graphics Interface*, pages 87–94, May 2000.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, et Paul Haeberli. « Fast shadows and lighting effects using texture mapping ». Dans *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252. ACM Press, 1992.

[SL98] John Snyder et Jed Lengyel. « Visibility Sorting and Compositing Without Splitting for Image Layer Decomposition ». Dans Michael Cohen, éditeur, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 219–230. ACM SIGGRAPH, Addison Wesley, juillet 1998. ISBN 0-89791-999-8.

[SLS+96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, et John Snyder. « Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments ». Dans Holly Rushmeier, éditeur, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, aug 1996.

[SM93] Michel Schmitt et Juliette Mattioli. *Morphologie Mathématique*. Masson, Paris, 1993.

[SMB01] Richard Southern, Patrick Marais, et Edwin Blake. « Generic memoryless polygonal simplification ». Dans *Proceedings of the 1st conference on Computer graphics, virtual reality and visualisation*, pages 7–15. ACM Press, 2001.

[SS95] G. Schaufler et W. Stürzlinger. « Generating multiple levels of detail from polygonal geometry models ». Dans *Selected papers of the Eurographics workshops on Virtual environments '95*, pages 33–41. Springer-Verlag, 1995.

[SS96] Gernot Schaufler et Wolfgang Sturzlinger. « A Three-Dimensional Image Cache for Virtual Reality ». *Computer Graphics Forum*, 15(3) :C227–C235, C471–C472, septembre 1996.

[SZL92] W. J. Schroeder, J. A. Zarge, et W. E. Lorensen. « Decimation of Triangle Meshes ». Dans *SIGGRAPH '92*, volume 26, pages 65–70, juillet 1992.

[Tel92] Seth J. Teller. « Computing the antipenumbra of an area light source ». Dans *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 139–148. ACM Press, 1992.

[TK96] Jay Torborg et James T. Kajiya. « Talisman : commodity real-time 3D graphics for the PC ». Dans *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 353–363. ACM Press, 1996.

[TS91] Seth J. Teller et Carlo Séquin. « Visibility Preprocessing for Interactive Walkthroughs ». Dans *Proc. Computer Graphics (SIGGRAPH'91)*, volume 25, pages 61–69. ACM Press, 1991.

[Tur92] Greg Turk. « Re-tiling polygonal surfaces ». Dans *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 55–64. ACM Press, jul 1992.

[Tur02] Jérémie Turbet. « *De l'analyse à la conception d'algorithmes pour une radiosité hiérarchique efficace* ». PhD thesis, Université Joseph Fourier, Grenoble, Mars 2002.

[UST] UST. « Urban Simulation Team ».

[Vel01] Luiz Velho. « Mesh Simplification using Four-Face Clusters ». Dans *Proceedings of SMI 2001*. Instituto per la Matematica Applicata - CNR, IEEE Computer Society, May 2001. International Conference on Shape Modeling and Applications.

[VL96] Philippe Véron et Jean-Claude Léon. « Synthèse et Analyse des différentes approches de simplification de polyèdres ». Dans *Congrès Numérisation 3D : Design et Digitalisation, Création Industrielle et Artistique*, mai 1996.

[Wil83] Lance Williams. « Pyramidal Parametrics ». Dans *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17(3), pages 1–11, juillet 1983.

[WS99] Peter Wonka et Dieter Schmalstieg. « Occluder Shadows for Fast Walkthroughs of Urban Environments ». Dans P. Brunet et R. Scopigno, éditeurs, *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 51–60. The Eurographics Association and Blackwell Publishers, 1999.

[WWS00] Peter Wonka, Michael Wimmer, et Dieter Schmalstieg. « Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs ». Dans *Eurographics Workshop on Rendering*, 2000.

[WWS01] Peter Wonka, Michael Wimmer, et François Sillion. « Instant Visibility ». Dans *EuroGraphics*. Eurographics, A. Chalmers and T.-M. Rhyne, 2001.

[XV96] Julie C. Xia et Amitabh Varshney. « Dynamic view-dependent simplification for polygonal models ». Dans *Proceedings of the conference on Visualization '96*, pages 327–334. IEEE-CS Press, octobre 1996.

[ZG02] Steve Zelinka et Michael Garland. « Permission grids : practical, error-bounded simplification ». *ACM Transactions on Graphics (TOG)*, 21(2) :207–229, 2002.

[ZMHKEH97] Hansong Zhang, Dinesh Manocha, Tom Hudson, et III Kenneth E. Hoff. « Visibility culling using hierarchical occlusion maps ». Dans *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 77–88. ACM Press/Addison-Wesley Publishing Co., 1997.

Table des figures

1.1 Pyramide de rendu 13
1.2 Ombre et pénombre 15
1.3 Fusion des pénombres 16
1.4 Visibilité dans l'espace des droites 19
1.5 Un exemple de maillage dégénéré mais étanche 21
1.6 Convex Vertical Prisms 22
1.7 Importance de l'ordre de parcours 23
1.8 Importance de l'ordre de parcours (suite) 23
1.9 Fusion incomplète des pénombres 24
1.10 Perte d'ombre 25
1.11 Variations d'une boîte englobante avec les petits détails 26
1.12 Modèle étendu 27
1.13 Réduction de deux cubes adjacents 29
1.14 *d*-réduction d'un objet 29
1.15 Réduction de bloqueur, démonstration 30
1.16 Différence entre érosion et différence de Minkowski 31
1.17 Réduction bloqueur et bloqués, démonstration 32
1.18 Rendu des six faces d'un cube dans une seule vue 33
1.19 Calcul simple *versus* calcul conservatif de la visibilité 34
1.20 Types de cellule à l'interface intérieure/frontière 38
1.21 Structure de données pour représenter l'intérieur 38
1.22 Décompositin d'un parallèlépipède 39
1.23 Érosion d'un ensemble de voxels par un segment 40
1.24 Grille creuse 41
1.25 Réduction et érosion morphologique 41
1.26 Réduction continue 2D 42
1.27 Matière à créer pour une arête concave 42
1.28 Les 13 types de sommets 44
1.29 Matière à créer pour les différents types de sommets 45
1.30 Dinosaure, voxelisations interne et externe 46
1.31 Dinosaure, réduction continue 46
1.32 Voxelisation des objets 47
1.33 Boston et Vienne 48
1.34 Un exemple de précalcul 49

3.1 Un *billboard* d'arbre 75
3.2 Un nuage de *billboards* 75
3.3 Couverture géométrique 79
3.4 Paramétrisation d'un plan 80
3.5 Représentation duale de l'ensemble des plans 81
3.6 Dual d'un point 82
3.7 Dual d'une sphère 83
3.8 Échantillonnage de l'espace des plans 84
3.9 Discrétisation à différentes résolutions 84
3.10 Discrétisation simple des plans valides pour un sommet 86
3.11 Contribution des aires projetées 88
3.12 Effet « dominos » 89
3.13 Contribution cumulée pour la sphère 89
3.14 Contribution cumulée en fonction de z 91
3.15 Surévaluation de la contribution 91
3.16 Courbe réelle et effective des contributions 92
3.17 Approximations d'une sphère 92
3.18 Densité en fonction de δ et de α 94
3.19 Exemple de raffinement adaptatif 95
3.20 Raffinement des voisins dans l'algorithme glouton adaptatif 96
3.21 Projection orthogonale des faces sur un *billboard* 98
3.22 Simplifier une maison en 7 étapes 101
3.23 Résultats pour des maillages complexes 102
3.24 Effets complexes de parallaxe et de transparence 103
3.25 Simplification d'une scène complète de 174 000 polygones 103
3.26 Un exemple complexe, une main à 654666 faces 105
3.27 Exemples d'ombres portées 107
3.28 Rôle de la matrice de texture 107
3.29 Matrice d'ombre 108
3.30 Différentes ombres 109
3.31 Intersection rayon/nuage de *billboards* 111
3.32 Imposteurs multi-couches 112
3.33 Erreur de reprojection 113
3.34 Erreur de reprojection 114
3.35 Profils des angles 115
3.36 Meilleurs points de vue de référence 116
3.37 Choix du centre de la cellule 117
3.38 Simplification dépendante du point de vue 118
3.39 Cellules basées sur les objets 119

Liste des tableaux

1.1 Résultats du précalcul 48

3.1 Paramètres OpenGL pour la génération de *billboard* 98
3.2 Statistiques pour les simplifications 104
3.3 Paramètres OpenGL pour le rendu d'ombre de *billboards* 109
3.4 Paramètres OpenGL pour le rendu d'ombre semi-transparentes . . 110

Algorithmes

1.1 Discrétisation d'une face dans un octree 36
3.1 Discrétisation de l'ensemble de validité d'une face 87
3.2 Recherche gloutonne de plans 96
3.3 Raffinement adaptatif 97

Table des matières

Introduction **3**

Notations **9**

1 Calcul de visibilité **11**
1.1 La problématique 11
1.1.1 Calcul dynamique 12
1.1.2 Calcul statique 13
1.2 Analyse préliminaire 14
1.2.1 Analogie lumineuse 14
1.2.2 Fusion des pénombres 15
1.2.3 Contraintes algorithmiques 16
1.2.4 Conditions analytiques 17
1.3 Visibilité volumétrique 18
1.3.1 Avantages 20
1.3.2 Limitations 22
1.4 La réduction de bloqueurs 27
1.5 Réduction volumique de bloqueurs voxelisés 29
1.5.1 Réduction de bloqueurs et de récepteurs 29
1.5.2 Réduction approximative 35
1.5.3 Réduction continue de voxels 36
1.6 Résultats 46
1.6.1 Réduction 46
1.6.2 Pré-calcul de la visibilité 47
1.7 Analyse et perspective 49

2 Travaux antérieurs **53**
2.1 Simplification de maillage 54
2.1.1 Simplification de maillage, quand et pourquoi faire ? . . . 54
2.1.2 Clusterisation de sommets 57
2.1.3 Décimation de primitives 59
2.1.4 Autres approches 64
2.1.5 La gestion des attributs 65
2.1.6 Contrôle de l'erreur 67
2.2 Rendu à base d'image 67

2.2.1 Méthodes purement image 67
2.2.2 Méthodes hybrides 69

3 Simplification de modèles polygonaux 73
3.1 Les nuages de *Billboards* 73
3.2 Motivations 76
3.3 Algorithme de construction 76
3.3.1 Vue d'ensemble 77
3.3.2 Fonction d'erreur et fonction de coût 77
3.3.3 Validité 78
3.3.4 Couverture géométrique 78
3.4 Espace des plans 80
3.4.1 Dualisation et représentation 80
3.4.2 Discrétisation 82
3.4.3 Détermination de la validité simple 85
3.5 La fonction de densité 87
3.5.1 Contribution 88
3.5.2 Pénalité 88
3.5.3 Choix de δ et de α 93
3.6 Algorithme glouton 93
3.6.1 Raffinement adaptatif 95
3.6.2 Construction d'un *billboard* 96
3.6.3 Mise à jour des pénalités 98
3.7 Rendu d'un nuage de *billboards* 99
3.8 Résultats 100
3.8.1 Un exemple simple 100
3.8.2 Exemples divers 100
3.9 Applications 106
3.9.1 Ombre portée au sol 106
3.9.2 Intersection avec un rayon 110
3.9.3 Application de l'algorithme de recherche de plans 111
3.10 Extension au cas dépendant du point de vue 111
3.10.1 Domaine de validité 112
3.10.2 Choix du point de référence 113
3.10.3 Résultats 117
3.11 Travaux futurs envisagés 118

Conclusion 123