Université Paul Sabatier – Toulouse 3 École Doctorale Informatique et Télécommunications

Algorithmes pour le rendu temps-réel de haute qualité des géométries basées points

MÉMOIRE

présenté et soutenu publiquement le 9 décembre 2005

pour l'obtention du

Doctorat de l'Université Paul Sabatier

(Spécialité Informatique)

par

Gaël GUENNEBAUD

Composition du jury

Président :	René Caubet (Professeur Université Toulouse III)
Rapporteurs :	Markus Gross (Professeur ETH Zurich) George Drettakis (Directeur de recherche INRIA Sophia-Antipolis)
Examinateurs :	Christophe Schlick (Professeur Université Bordeaux) Fabrice Neyret (CR CNRS GRAVIR-IMAG-INRIA Grenoble) Mathias Paulin (MDC HDR Université Toulouse III, Directeur de thèse)
Invité :	Loïc Barthe (MDC Université Toulouse III)

Institut de Recherche en Informatique de Toulouse — UMR CNRS 5505

Résumé

Les scènes virtuelles 3D deviennent de plus en plus couramment utilisées comme vecteur d'informations numériques. A l'instar des sons, des images ou de la vidéo, cette nouvelle forme de média nécessite des outils d'acquisition, de stockage, d'édition, de transmission et de restitution de géométries 3D. Dans de nombreux domaines d'applications, il existe une forte demande pour des outils performants permettant de manipuler des scènes toujours plus complexes.

En synthèse d'images, les objets manipulés sont généralement visualisés par des maillages polygonaux. Principalement à cause de leur forte connectivité, ceux-ci montrent de nombreuses limites dès que la complexité des objets dépasse un certain seuil. Parmis les représentations alternatives récemment proposées, les géométries à base de points sont celles offrant le plus de flexibilité. Ici, un objet quelconque est simplement représenté par un ensemble de points répartis sur la surface.

Dans cette thèse nous proposons un *pipe-line* complet de rendu de géométries basées points. Notre *pipe-line* de rendu a la particularité de maintenir une très haute qualité de la visualisation que ce soit en cas de fort sur-échantillonnage, ou au contraire en cas de fort sous-échantillonnage tout en garantissant le temps-réel. Pour cela, nous présentons dans un premier temps une adaptation de l'algorithme de *splatting* de surface aux cartes graphiques actuelles afin de tirer profit de leurs performances accrues. Offrant de plus en plus de flexibilité, nous montrons que ces dernières permettent maintenant un rendu par points avec une haute qualité du filtrage et du calcul de l'ombrage.

Cependant, les calculs coûteux requis par le filtrage des points ne sont pas supportés nativement. Lors du passage à l'échelle, c'est à dire dans le cadre de la visualisation de scènes complexes, il est alors parfois nécessaire de se passer de ce filtrage. Aussi, dans un deuxième temps nous proposons une méthode appelée *deferred splatting* dont le principe est de retarder au maximum les opérations de filtrage afin de ne réaliser celles-ci que sur les points réellement visibles. Avec notre méthode les coûts du filtrage deviennent donc indépendant de la complexité de la scène.

D'une manière générale, la précision d'une représentation basée points dépend fortement de la densité des échantillons. En effet, dès que cette densité n'est plus suffisante, lors de zoom par exemple, la qualité des images rendues par splatting se dégrade rapidement. Aussi, d'une manière similaire aux surfaces de subdivisions pour les maillages polygonaux, nous proposons un algorithme complet de raffinement itératif des géométries basées points. Basé sur des heuristiques locales et un calcul de voisinage précis, notre algorithme est capable de reconstruire dynamiquement une surface lisse à partir d'un nuage de points irrégulier et/ou sous-échantillonné.

Mots-clés: Synthèse d'images, Rendu temps-réel, Rendu à base de points, Raffinement

Abstract

In computer graphics, manipulated objects are usually visualized by a polygonal meshes. Owing to their connectivity, meshes exhibit several limitations as soon as the geometric complexity reached a given threshold. Among all the alternative representations that have been recently proposed, point-based geometries appear to be the most flexible of them. With point-based geometries, an object is simply represented by a set of points spread on its surface.

In this thesis we propose a complete rendering pipeline of point-based geometries which has the particularity to maintain a very high rendering quality both in case of under-sampled or over-sampled models while maintaining real-time performances. To reach these goals, in a first time we present an adaptation of the surface splatting algorithm to modern graphics cards in order to get benefit from their high performances. Thanks to their new flexibilities, we show how to modern graphics cards can now support the rendering of point clouds with very high filtering and shading quality.

However, the expensive computations required by the filtering of points are not natively supported yet. Then, in the context of the visualisation of complex scenes, it is often necessary to remove this filtering to maintain interactive frame rate. So, in a second time, we propose a new technique called *deferred splatting* which defers filtering operations in order to perform them on visible points only. Hence, with our method, the expensive filtering cost doesn't depend on the scene complexity.

From a geometric point of view, the accuracy of a point-based representation mainly depends on the sampling density. When the point set is not dense enough, e.g. when zooming in, images rendered by a splatting technique exhibit several artefacts. So, in similar fashion to subdivision surfaces for polygonal meshes, we proposed an interpolatory refinement framework for point-based geometries. Based on local heuristics and an accurate neighborhood computation, we show that our algorithm is able to dynamically reconstruct a smooth surface from a scattered and/or under-sampled models.

Keywords: Computer graphics, Real-time rendering, Point-based rendering, Refinement

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Contexte de l'étude et motivations

Les scènes virtuelles 3D deviennent de plus en plus couramment utilisées comme vecteurs d'informations numériques. A l'instar des sons, des images ou de la vidéo, cette nouvelle forme de média nécessite des outils d'acquisition, de stockage, d'édition, de transmission et de restitution de géométries 3D. L'objectif final de la synthèse d'image est, par définition, l'obtention d'une image à partir d'une représentation informatique de notre scène 3D. Cependant, les attentes que nous pouvons avoir envers cette étape de visualisation peuvent varier d'un extrême à l'autre en fonction du domaine d'application et de l'usage courant. Par exemple, dans le cadre de la création de films d'animation, nous devons à la fois disposer d'algorithmes de rendu interactif lors de la phase de modélisation et de la mise en place des animations, alors que la génération finale de la séquence d'images animées peut être réalisée de manière différée. Les différents critères que nous pouvons choisir de mettre en avant sont :

- **Le réalisme physique de l'éclairage.**
  L'objectif de ce critère est de garantir que l'image obtenue soit bien en accord avec les lois de la physique. Ce critère est par exemple très important en aménagement spatial ou design automobile.

- **La qualité visuelle.**
  Ce critère, où seul l'observateur est juge, peut être atteint de différentes manières. L'objectif peut être la quête d'un photo-réalisme suffisant pour tromper l'observateur ou, au contraire, s'éloigner du photo-réalisme afin d'atteindre la sensibilité artistique des observateurs, ou encore de rendre une scène *lisible* (illustration technique par exemple).

- **L'interactivité.**
  C'est-à-dire être capable de fournir l'image à l'observateur dans un temps raisonnable ou maîtrisé. Selon les applications, des temps de rendu acceptables peuvent aller de quelques secondes à quelques millisecondes. Lorsque les temps de calcul sont suffisement courts et maîtrisés pour garantir un taux de rafraîchissement supérieur à 24 images par seconde nous pouvons alors parler de rendu temps-réel. Pour une certaine gamme d'applications, le temps-réel est sans doute le critère le plus important.

- **La flexibilité**
  En fait, un algorithme de visualisation peut être qualifé de *flexible* pour énormément de raisons différentes. Dans ces travaux, notre critère de flexibilité le plus important est la capacité à visualiser directement, c'est-à-dire sans précalculs, une scène 3D. Ce critère est particulièrement important pour toutes les applications où l'utilisateur intéragit avec la géométrie.

Les deux premiers critères ont une influence forte sur les temps de calculs, et il est souvent nécessaire de réaliser des compromis, soit sur la qualité, soit sur l'interactivité. Alors que les méthodes de rendu de haute qualité, telles que les techniques à base de lancer de rayons ou de radiosité, sont souvent utilisées pour la production en différé d'une séquence d'images animées ; les algorithmes de rendu par projection et discrétisation (*rastérisation*) sont quant à eux majoritairement utilisés pour les applications interactives ou temps-réel. Les performances de ces algorithmes sont cruciaux dès qu'il y a interaction entre l'utilisateur et la scène, que ce soit lors de l'étape de modélisation ou bien au sein d'un simulateur virtuel. Le tracé de polygones étant grandement optimisé par les cartes graphiques, la plupart des algorithmes de rendu interactif s'appuient sur une représentation polygonale.

Le point : primitive universelle de rendu ? Cependant de nombreux autres types de représentations géométriques que le polygone sont couramment utilisés par les étapes d'acquisition, d'édition et de modélisation. En modélisation, nous pouvons citer les surfaces paramétriques telles que les splines ou NURBS, les surfaces de subdivisions ou encore les surfaces implicites. La visualisation de ces géométries, non directement *rastérisables*, est donc couramment réalisée par une phase d'échantillonnage de la surface sous la forme d'un maillage triangulaire suivie d'une rastérisation de celui-ci. Échantillonner une surface par un maillage de triangles revient à positionner des sommets sur cette surface puis à connecter ces points. En plus des éventuels problèmes de topologie, la gestion de la connectivité entre les sommets du maillage est conceptuellement inefficace : pourquoi ne pas *oublier* les informations topologiques pour, simplement, placer des points sur la surface ?

De nombreux travaux ont déjà montrés la viabilité d'une telle approche. Par exemple, les surfaces implicites, définies de manière discrète ou non, sont généralement converties en triangles en utilisant l'algorithme des *marching cubes* [LC87] dont une alternative par projection de points sur la surface est l'iso-splatting [CHJ03]. Lorsque la surface implicite est définie de manière continue, celle-ci peut efficacement être échantillonnée par des particules aléatoires [] ou contrôlées par des forces d'attraction et de répulsion [WH94]. Les surfaces paramétriques peuvent également avantageusement être échantillonnées par un nuage de points [CK03]. Dans le cadre des surfaces de subdivisions, comme l'a suggéré Catmull [Cat74], le concept de rendu par polygones perd tout son sens au profit d'un rendu par points dès que la taille des polygones a atteint la taille d'un pixel de l'image.

Si positionner des points sur une surface semble plus facile que positionner un maillage, encore faut-il être capable de visualiser efficacement et avec une bonne qualité les nuages de points ainsi obtenus.

Les points pour la gestion de la complexité ? Dans de nombreux domaines d'applications, de la simulation de phénomènes physiques au divertissement, il existe une forte demande pour des outils performants permettant de manipuler et visualiser des scènes toujours plus complexes.

Par scène complexe, nous entendons une scène nécessitant plusieurs millions de primitives géométriques, généralement des triangles, pour être fidèlement représentée. Lorsque l'on souhaite manipuler et visualiser en temps-réel de telles scènes, de nombreuses questions se posent. Comment gérer ces millions de polygones efficacement ? Comment éviter de réaliser des traitements parfois coûteux sur des régions invisibles ? Comment assurer un temps de calcul le plus faible possible, tout en conservant une perception correcte du monde virtuel par un utilisateur ? Bien sûr, ces questions ont déjà été énormément étudiées lors de ces 20 dernières années et de grandes avancées, matérielles et logicielles, ont été faites, permettant l'essor du rendu temps-réel et de ses applications dans le divertissement.

Dans ce contexte, une des principales clés de la réussite est de disposer d'une représentation multi-résolution efficace de notre scène capable de fournir à tout instant le niveau de simplification le plus adapté au traitement effectué. Bien que de nombreuses méthodes de simplification de maillage [LRC+02] aient vu le jour, aucune ne donnent de résultats satisfaisants dès lors que l'on dépasse un seuil de complexité trop important. Cette difficulté est en fait intrinsèque aux maillages polygonaux qui ont comme désavantages de présenter une forte connectivité et une décorrélation partielle de la géométrie et de la texture de la surface. Aussi, nous pouvons nous demander s'il n'existe pas une primitive plus simple que le polygone et offrant plus de flexibilité ?

Les points semblent donc à nouveaux une alternative viable, puisqu'une fois la paramétrisation et connectivité oubliées, il n'y a plus de limites pour la simplification de la géométrie. Basiquement, il suffit de moyenner les attributs des points les plus proches pour réduire la complexité du modèle. Encore une fois se pose le besoin d'algorithmes de rendu efficace et de qualité des nuages de points.

Le point : primitive universelle ? Si, comme nous le verrons dans le chapitre 2, l'utilisation des points comme primitives de rendu n'est pas un concept nouveau, très récemment les géométries à base de points sont également apparues comme représentations efficaces pour la modélisation et l'édition d'objets 3D. Pour de nombreuses opérations d'édition et de modélisation, les représentations à base de points, caractérisées par une absence de connectivité et de paramétrisation, ont montré un net avantage sur les autres types de géométrie. Par exemple, les points sont particulièrement bien adaptés aux opérations de sculpture et de peinture de la surface [ZPKG02, AWD+04], à la déformation de formes libres [PKKG03, BK05] ou encore à la modélisation d'objets élastiques [MKN+04], sans oublier les traditionnelles opérations booléennes [PKKG03, AD03].

De plus, aujourd'hui les périphériques d'acquisition, tels que les scanners 3D se sont démocratisés. Ceux-ci permettent l'acquisition d'objets du monde réel et fournissent en sortie un nuage de points. Il y a encore quelques années, ces nuages de points étaient au préalable triangulés, avant d'être traités par des algorithmes connus travaillant sur la représentation polygonale. Récemment, de nombreux travaux [FCOAS03, PGK02, WPK+04] ont montré que toutes les opérations de débruitage, reconstruction des trous, simplification et compression, pouvaient être avantageusement réalisées directement sur le nuage de points. Finalement, les nuages de points apparaissent comme une représentation viable d'un bout à l'autre d'une chaîne de création de contenu 3D typique : acquisition, traitement, modélisation/édition et enfin visualisation. En ce sens, le point devient donc une primitive universelle.

De nos jour, de plus en plus de périphériques accessibles sont mis en réseau, rendant l'accès aux média numérique de plus en plus aisé. La transmission des sons et vidéos est déjà largement démocratisée. Il est donc important d'ajouter dans cette chaîne de traitement et de visualisation une étape de transmission. Contrairement aux polygones, pour lesquels sommets, topologie et textures 2D ont à être transférés, dans le cadre de la transmission de scènes 3D, les représentations à base de points sont particulièrement bien adaptées puisque nous avons qu'un seul type de flux, c'est-à-dire des points [RL01, ZZY04, GM04]. Quelque soit la quantité de points reçu par le client, à tout instant celui-ci est capable d'effectuer le rendu, même partiel ou avec une qualité réduite, de la scène. De ce fait, une solution par points est beaucoup moins sensible à la perte de paquets ou aux éventuelles erreurs de transmissions.

Motivations

Comme nous venons de le voir, il existe une forte demande pour des algorithmes de rendu de points rapides et offrant une bonne qualité d'image. Dans cette thèse, nous nous sommes donc plus particulièrement interessés à la visualisation des nuages de points. Afin de couvrir un large spectre d'applications, nos principaux critères pour le développement d'un algorithme de rendu sont : rapidité, qualité et flexibilité. Ces deux premiers critères sont bien sûr antagonistes puisque généralement une plus grande qualité d'image requiert également des calculs plus coûteux. Le critère de flexibilité signifie principalement que notre *pipe-line* de rendu doit être capable de satisfaire au mieux les deux premiers critères quelque soient les données en entrée. Nous devons notamment être capables de tracer directement un nuage de points non organisé, indispensable lorsque l'utilisateur intéragi avec la géométrie, tout en offrant la possibilité d'utilisation de structure de données multi-résolution indispensable au rendu de scènes complexes.

1.2 Contribution et organisation du mémoire

Dans ce mémoire, nous proposons un *pipe-line* de rendu de géométries basées points complet, résumé figure 1.1. Notre *pipe-line* de rendu a la particularité de maintenir une très haute qualité de la visualisation, que ce soit en cas de fort sur-échantillonnage ou au contraire en cas de fort sous-échantillonnage, tout en assurant des performances optimales et une forte flexibilité. Les trois étages de ce *pipe-line* de rendu correspondent respectivement, de bas en haut, aux chapitres 3, 4, et 5 de ce mémoire :

Chapitre 3 : étage "splatting de haute qualité". Nous présentons dans un premier temps une adaptation de l'algorithme de *splatting* de surface aux cartes graphiques actuelles afin de tirer profit de leurs performances accrues. Offrant de plus en plus de flexibilité, nous montrons que ces dernières permettent maintenant un rendu par points avec une haute qualité du filtrage et du calcul de l'ombrage. Notre implantation supporte les objets semi-transparents ainsi que le rendu hybride points-polygones avec lissage des transitions.

Chapitre 4 : étage "deferred splatting". Cependant, les calculs coûteux requis par le filtrage des points ne sont pas supportés nativement par le matériel graphique. Lors du passage à l'échelle, c'est-à-dire dans le cadre de la visualisation de scènes complexes, il est alors parfois nécessaire de se passer de tout type de filtrage pour maintenir l'intéractivité. Aussi, dans un deuxième temps, nous proposons une méthode appelée *deferred splatting* dont le principe est de retarder au maximum les opérations de filtrage afin de ne réaliser celles-ci que sur les points réellement visibles. Avec notre méthode, les coûts du filtrage

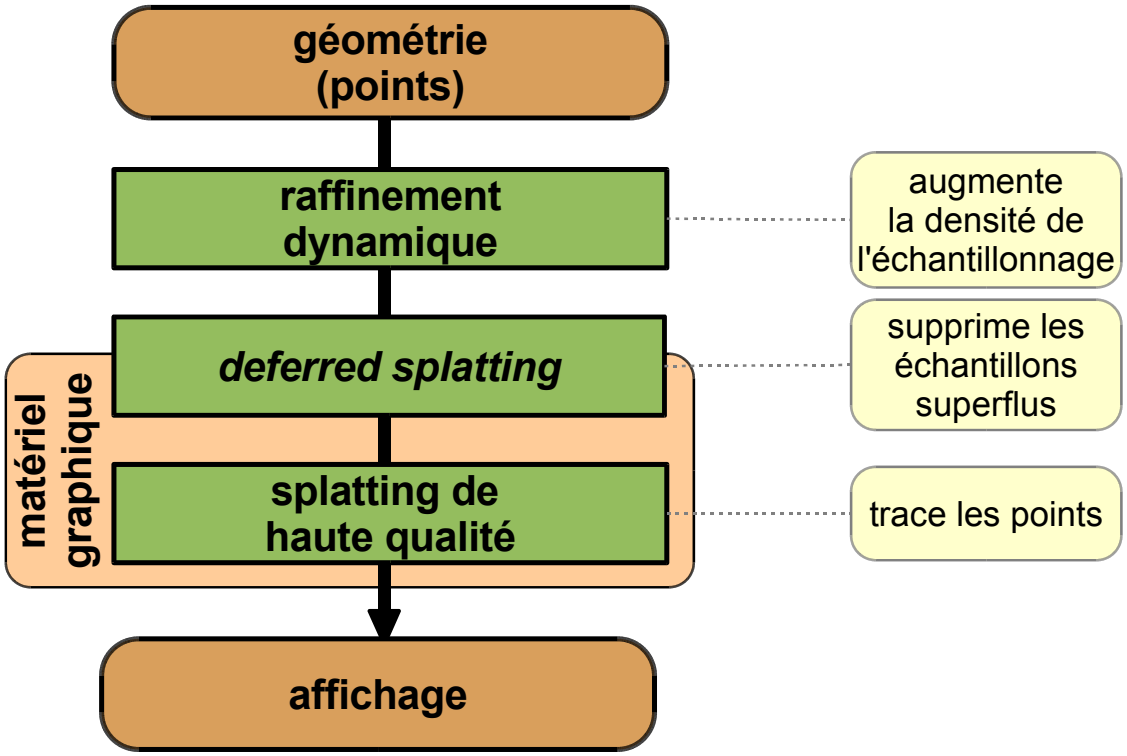


FIG. 1.1 – Aperçu de notre pipe-line de rendu par points.

deviennent donc indépendants de la complexité de la scène.

Chapitre 5 : étage "raffinement dynamique". D'une manière générale, la précision d'une représentation basée points dépend fortement de la densité des échantillons. En effet, dès que cette densité n'est plus suffisante, lors d'un zoom par exemple, la qualité des images rendues par splatting se dégrade rapidement : les disques représentant les points deviennent visibles, notamment au niveau de la silhouette des objets. Aussi, d'une manière similaire aux surfaces de subdivision pour les maillages polygonaux, nous proposons un algorithme complet de raffinement itératif des géométries basées points. Basé sur des heuristiques locales et sur un calcul de voisinage précis, notre algorithme est capable de reconstruire dynamiquement une surface lisse à partir d'un nuage de points irrégulier et/ou sous-échantillonné. À la fin de ce chapitre nous montrons comment notre algorithme de raffinement peut être efficacement intégré dans notre *pipe-line* de rendu temps-réel.

Avant de présenter notre *pipe-line* de rendu nous commencerons, au **chapitre 2**, par faire le tour des repréentations géométriques à base de points existantes et des algorithmes de rendu associés à ces représentations. Nous finirons ce tour d'horizon par la présentation de quelques outils et méthodes pour le rendu à base de points de scènes complexes. Dans cet état de l'art nous discuteront des limites des méthodes de rendu à base de points actuelles tout en justifiant les orientations de recherches prisent dans le cadre de cette thèse.

Chapitre 2

Le rendu à base de points : état de l'art

Notre principal objectif dans cet thèse étant d'établir un *pipe-line* de rendu à base de points flexible, rapide et calculant des images de qualité, nous allons, dans de ce chapitre, présenter et discuter des avantages et limitations des méthodes de visualisation des nuages de points existantes (section 2.2). Cependant, dans un premier temps, nous préciseront la notion même de représentation basée points en faisant un tour d'horizon des différents types de géométries pouvant être considérées comme étant "à base points" (section 2.1). Le rendu par points de scènes très complexes faisant parti des applications visées pour notre *pipe-line*, dans une troisième section (numéro 2.3) nous présenteront des sructures de données multi-résolution et algorithmes permetant d'accélérer le rendu en ne sélectionnant qu'un minimum de points à tracer.

Émergence du concept de point en tant que primitive universelle de visualisation

Modéliser et visualiser des objets représentés sous la forme d'une collection de points est loin d'être une idée récente. Dans les années 70, les premiers jeux vidéos représentaient déjà des vaisseaux spatiaux explosant grâce à de petits points lumineux remplissant l'écran. En fait, les points ont dans un premier temps principalement été utilisés sous la forme de particules pour modéliser des objets dit "flous", c'est-à-dire dont la surface n'est pas vraiment définie et donc difficilement représentable par les modèles géométriques classiques. Une particule n'est rien d'autre qu'un point 3D associé à un certain nombre d'attributs tels que couleur, taille, densité, forme, accélération, etc. En 1979, Charles Csuri et al. utilisent des particules statiques pour modéliser la fumée sortant d'une cheminée [CHP+79]. En 1982, Jim Blinn anime les particules pour représenter des nuages [Bli82]. Dans le même temps, Reeves présente son fameux système de particules [Ree83], plus générique que le précédent, permettant de simuler feux, explosions et herbes. Ce système de particules a notamment été utilisé pour générer un mur de feux dans le film Star Trek II : The Wrath of Khan [RLE+82]. Peu après, diverses améliorations ont été proposées permettant de représenter, par exemple, chutes d'eaux [Sim90] et autres fluides [MP89]. Cependant, toutes ces méthodes utilisent des particules dites isotropes, c'est-à-dire sans orientation, afin de représenter des objets volumiques.

En 1985, Levoy et Whitted furent les premiers à envisager l'utilisation du point comme primitive universelle de modélisation et de rendu de surface [LW85]. Ils proposent d'échantillonner la surface de tous types d'objets sous la forme d'un ensemble de points 3D suffisamment dense

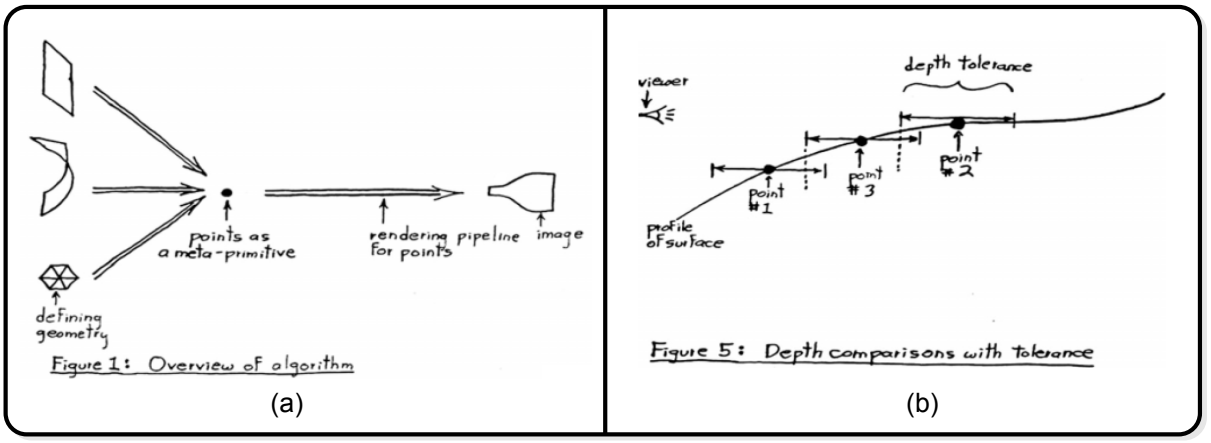


FIG. 2.1 – Le rendu par point selon Levoy et Whitted. (sources [LW85])
(a) Le point : primitive universelle de rendu ? (b) z-buffer avec tolérance.

pour donner l'impression d'un objet solide (figure 2.1-a). Dans leur proposition, un point a une position, une normale, une couleur et un coefficient de transparence. Dans leur rapport, les principaux problèmes posés par le rendu par points y sont abordés et discutés :

- Aliasing en cas de réduction. Lorsque plusieurs points se projettent sur un seul pixel il est nécessaire de mettre en place un mécanisme de filtrage pour lisser les hautes fréquences, c'est-à-dire moyenner entre eux les points visibles.
- Trou en cas d'agrandissement. Au contraire, lorsque tous les pixels correspondant à une surface ne sont pas atteint par la projection d'un point, l'arrière plan, c'est-à-dire les objets non visibles, devient visible au travers du premier plan. Il est donc nécessaire de mettre en place un mécanisme pour interpoler les points visibles entre eux.

L'algorithme de rendu proposé par Levoy et Whitted est déjà très évolué :

- Interpolation par accumulation de filtres de reconstruction Gaussien dans l'espace image avec anti-aliasing.
- Utilisation d'un z-buffer avec tolérance pour assurer le mélange de toutes les contributions réellement visibles (figure 2.1-b).
- Transparence indépendante de l'ordre. Permet de ne pas avoir à trier les primitives par profondeur à chaque rendu d'une image.
- Anti-crénelage des silhouettes.

Ces deux dernière fonctionnalités sont gérées par une sorte de A-buffer (anti-aliasing buffer) [Car84].

Il est intéressant de remarquer que, comme nous le verrons par la suite, vingt ans plus tard ce sont toujours les mêmes mécanismes qui sont utilisés par les algorithmes de rendu les plus évolués. En revanche, les GPU[1]les plus modernes, 60000 fois plus puissants que le VAX 11-780 utilisé par Levoy et Whitted en 1985, avec environ 4000 fois plus de transistors et 5000 fois plus de mémoire vive, ne permettent toujours pas de réaliser des algorithmes aussi simples qu'un test de profondeur avec tolérance ou qu'un rendu multi-couches. À l'heure actuelle, ces mécanismes doivent donc être simulés via des algorithmes de rendu multi-passes.

[1]GPU signifie *Graphic Processor Unit* et désigne le processeur de la carte graphique [SA04].

Le concept de point a été repris par la suite par Max pour le rendu d'arbre [MO95] et par Szelisky et Tonnesen en 1992 dans [ST92] où ils présentent un outil de modélisation basé sur un système de particules orientées qui définissent la surface de l'objet.

Cependant, à l'époque de ces méthodes pionnières dans ce domaine, les ordinateurs disponibles n'étaient pas suffisamment puissants pour afficher en temps interactif un grand nombre de points. Aussi, l'engouement pour les méthodes basées points n'a vraiment commencé qu'en 1998 avec la technique des *layered depth images* de Shade et al. [SGwHS98] et au système de rendu interactif par points de Grossman et Dally [GD98, Gro98] présentés simultanément.

Depuis, des techniques nombreuses et variées ont vu le jour. Dans un premier temps, nous allons récapituler les différents types de représentations à base de points existantes, section 2.1. Section 2.2, nous verrons les méthodes permettant de visualiser directement un nuage de points désorganisés en classant les méthodes basées sur un lancer de rayon d'un côté et les méthodes par rastérisation d'un autre côté. Enfin, dans une troisième section 2.3 nous nous intéresserons aux techniques permettant de sélectionner efficacement les points à tracer.

2.1 Les représentations à base de points

Nous pouvons considérer comme "à base de points", tout type de géométrie où la surface d'un objet est représentée par un ensemble d'échantillons ponctuels $\mathbf{p}_i \in \mathbb{R}^3$ non connectés. Optionnellement, chaque point $\mathbf{p}_i$ peut être associé à un ensemble d'attributs de la surface tels que la normale, la couleur ou toute autre propriété de matériaux. Durant ces dernières années, une large variété de type de géométries basées points ont été proposées. Nous classons ces géométries en deux catégories principales : les nuages de points désorganisés d'un coté, que nous définissons par opposition à la seconde catégorie correspondant aux nuages de points organisés sous forme d'images ou de grilles 3D.

2.1.1 Les nuages de points désorganisés

Nous allons commencer par nous intéresser au cas général des nuages de points non structurés. Les types de géométries présentées ici sont basées sur une simple liste de points attribués dans laquelle les coordonnées des positions des points sont stockées explicitement. Différents types de représentations peuvent être distingués en fonction des attributs relatifs à la géométrie de la surface stockés en chaque point.

2.1.1.1 Les points purs

Pour commencer, nous pouvons considérer les nuages de points où aucune information sur la géométrie locale de la surface n'est associée aux points. En pratique, cette classe de représentation n'est jamais utilisé directement puisque l'étape de visualisation nécessite au minimum une normale. Comme nous le verrons dans la suite, il est généralement possible de passer d'une telle représentation aux représentations suivantes. Citons tout de même les travaux de Xu et al. [HXC04] où une technique de rendu non photo réaliste permet de visualiser un nuage de points sans information de normales.

2.1.1.2 Les points orientés

Ensuite, nous pouvons considérer les nuages de points orientés où chaque point stocke en plus la normale de la surface en ce point. Nous pouvons noter une faible perte en généralité et

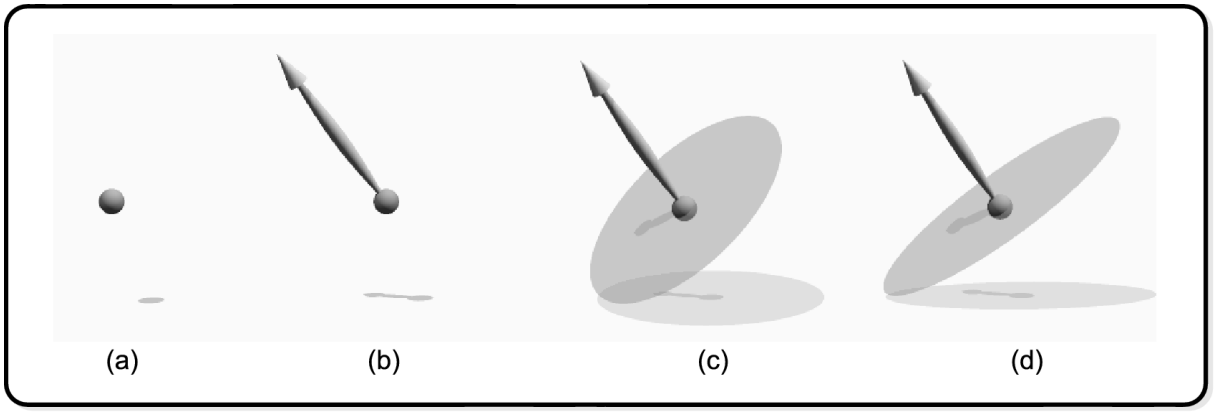


FIG. 2.2 – (a) un point pur. (b) un point orienté. (c) un splat. (d) un splat elliptique.

en flexibilité puisqu'il est maintenant nécessaire de maintenir une consistance des normales lors de toute manipulation géométrique. En pratique, ces pertes sont négligeables. Au contriare, en plus de permettre le calcul de l'éclairage lors du rendu, cette information de normale augmente nettement la précision de la représentation [ABK98]. De plus, il est généralement possible de passer d'un nuage de points purs à un nuage de points orientés.

En effet, si la densité du nuage de points est suffisante il est alors possible d'estimer la normale d'un point en analysant localement son voisinage. Comme aucune information de connectivité n'est disponible, ce voisinnage est habituellement construit en utilisant la méthode des k plus proches voisins. Soit $\mathbf{p}_0$ un point du nuage de points et $\{\mathbf{p}_1, ..., \mathbf{p}_k\}$ ses k plus proches voisins. La normale du point $\mathbf{p}_0$ est alors estimée comme étant la normale du plan passant au plus près du point $\mathbf{p}_0$ et de ses voisins au sens des moindres carrés. Cette normale est obtenue de manière équivalente par une analyse en composantes principales (ACP). Soit la matrice de covariance suivante :

$$\sum_{i=0}^{k} \|\mathbf{p}_i - \mathbf{c}\|^2 \in \mathbb{R}^{3\times 3} \tag{2.1}$$

où $\mathbf{c} = \frac{1}{k+1}\sum_{i=0}^{k} \mathbf{p}_i$. Au signe près, la direction de la normale du point $\mathbf{p}_0$ correspond alors au vecteur propre associé à la plus petite valeur propre de cette matrice (qui est symétrique par construction). Une orientation consistante des normales pour l'ensemble des points du nuage peut être obtenue par une propagation des orientations (intérieur/extérieur) à travers un arbre de chevauchement minimal [HDD+92].

Cependant, l'information de normale n'est généralement pas suffisante à la visualisation "sans trou" de la surface. Il est en effet nécessaire de connaître une estimation de l'espacement entre les points voisins. Cette information peut être donnée de manière implicite dans le cas d'échantillonnages réguliers. Une telle approche a comme inconvénient de rendre très difficile toute manipulation géométrique locale et de nécessiter un nuage de points très dense puisque les zones plates doivent être échantillonnées avec la même résolution que les zones de fortes courbures.

De cette manière, les *splats* semblent constituer un bien meilleur compromis entre la simplicité des primitives géométriques et le nombre de primitives qui doivent être utilisés.

2.1.1.3 Les splats

Les surfaces à base de splats ont été proposées en premier par Zwicker et al. en 2001 [ZPvBG01]. Un *splat* n'est rien d'autre qu'un point associé à une normale et à un rayon de telle sorte qu'il peut être vu comme un disque orienté posé sur la surface de l'objet. Les rayons associés aux points doivent être choisis de manière à boucher l'espace entre les points. Le rayon d'un point correspond donc à l'espacement entre ce point et ses voisins. Ce type de représentation fourni une approximation C^{-1} linéaire par morceau de la surface.

Comparés aux simples points ou points orientés, les splats ont l'avantage d'inclure la densité locale du nuage de points. Ainsi, il est possible d'ajuster la densité des points localement en fonction de la courbure locale tout en conservant la même erreur d'approximation. Cette erreur étant relative à la distance entre les splats et la surface sous-jacente, une représentation optimale en terme du nombre de points aura peu d'échantillons pour représenter les zones plates alors que les zones de fortes courbures seront échantillonnées par de nombreux mais petits splats. Plusieurs techniques ont déjà été proposées pour optimiser le nombre des splats en fonction d'une erreur d'approximation donnée. Citons celles de Pauly et al. en 2002 [PGK02] et Wu et Kobbelt en 2004 [WK04].

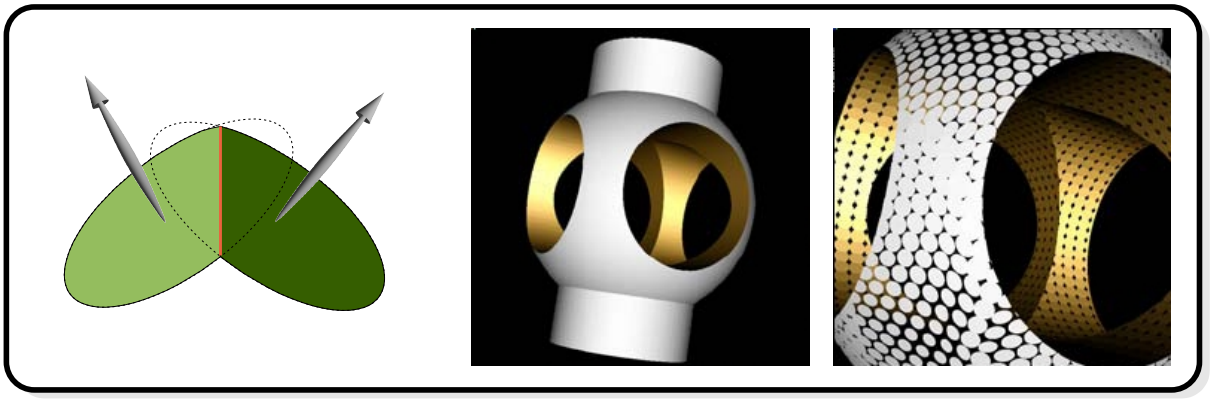


FIG. 2.3 – Représentation des arêtes franches. Les images de droites sont issues de [ZRB+04].

Un problème que nous n'avons pas encore abordé conçerne la représentation des arêtes franches et autres discontinuités (bords et coins). Théoriquement, avec une représentation par points, une arête franche pourrait être représentée par une densité infinie de points. Cependant, avec une représentation par splats, une arête franche peut efficacement être représenté en utilisant des splats découpés [PKKG03, ZRB+04]. À chaque splat intersectant un bord ou une arête franche est associé une ligne de découpe définie dans le plan tangent du splat. De la même manière, un coin peut être représenté en utilisant deux lignes de découpe par splat.

2.1.1.4 Les points différentiels

L'isotropie des splats tels que nous venons de les voir ne permet pas de prendre en compte l'anisotropie locale de la courbure de la surface. Une adaptation optimale à la courbure locale de la surface est possible en utilisant des splats elliptiques. Un tel splat est défini par la position $\mathbf{p}_i$, deux axes tangentiels $\mathbf{u}_i$ et $\mathbf{v}_i$ et leur rayon respectif. La normale $\mathbf{n}_i$ peut alors être déduite du

produit vectoriel des deux vecteurs tangents. Une telle approximation est localement optimale si les deux axes sont alignés avec les directions de courbures principales de la surface et que les rayons sont inversement proportionnels aux rayons de courbures maximal et minimal.

Le plan tangent τ_i du point $\mathbf{p}_i$ peut être défini paramétriquement par :

$$\tau_i(\mathbf{y}_i) = \mathbf{p}_i + y_{i,u}\mathbf{u}_i + y_{i,v}\mathbf{v}_i \tag{2.2}$$

où $\mathbf{y}_i = (y_{i,u}, y_{i,v})$ sont les coordonnées paramétriques d'un point dans le plan tangent.

En mettant à l'échelle les vecteurs tangents de sorte que leur norme soit égale aux rayons de l'ellipse, l'ellipse représentant le splat du point $\mathbf{p}_i$ est alors définie par l'ensemble des points de coordonnées paramétriques $\mathbf{y}_i = (y_{i,u}, y_{i,v})$ tel que $y_{i,u}^2 + y_{i,v}^2 \leq 1$.

Ce type de représentation a été proposé en premier lieu par Kalaiah et Varshney en 2001 [KV01, KV03] pour échantillonner des surfaces NURBS. En 2004, Wu et Kobbelt présentent une méthode de simplification de tels nuages de points [WK04]. Côté rendu, les points différentiels permettent de dériver une normale localement permettant un éclairage par pixel sans avoir à interpoler les normales entre les points [KV01, KV03, BK04] (voir aussi section 2.2.2.3).

En comparaison aux splats isotropes, les points différentiels ont comme avantage une plus grande puissance d'approximation puisque la même erreur d'approximation peut être obtenue avec un nombre moindre de primitives. Cependant, cet avantage devient obsolète dès que la fréquence spatiale des attributs de la surface devient supérieure à la fréquence de la surface elle même, ce qui est généralement le cas des objets texturés. De plus, ceux-ci sont moins compacts et offrent nettement moins de flexibilité puisqu'il est nécessaire de calculer et de maintenir à jour les informations de courbures, taches qui peuvent être particulièrement difficiles.

Les maillages de triangles et splats elliptiques fournissent tous deux une approximation d'ordre quadratique. Les splats elliptiques sont pourtant supérieurs aux triangles. En effet, en accord avec la géométrie différentielle, l'ellipse est localement le meilleur approximant d'une surface lisse [KB04]. De plus, puisque les splats ne sont pas connectés mais sont seulement C^{-1} continus, ils fournissent la même flexibilité topologique que les nuages de points purs.

2.1.1.5 Les nuages de splats paramétriques

Afin d'obtenir une approximation continue de la surface (et des attributs) à partir d'une représentation par splats circulaires, Zwicker [Zwi03] propose d'associer à chaque point un noyau de reconstruction local défini dans le plan tangent. Ce noyau doit être une fonction à base radiale décroissante, d'où le terme *splat* qui correspond à la forme d'une boule de neige écrasée (splatted). Généralement des noyaux Gaussiens sont utilisés. Soit $\mathbf{u}_i$, $\mathbf{v}_i$ deux vecteurs unitaires orthogonaux définis dans le plan tangent du point $\mathbf{p}_i$ de normale $\mathbf{n}_i$, ($\mathbf{u}_i$, $\mathbf{v}_i$, $\mathbf{n}_i$) forment donc une base orthonormée de notre espace 3D et une paramétrisation du plan tangent τ_i du point $\mathbf{p}_i$ est définie par : $\tau_i(\mathbf{y}_i) = \mathbf{p}_i + y_{i,u}\mathbf{u}_i + y_{i,v}\mathbf{v}_i$, où $\mathbf{y}_i = (y_{i,u}, y_{i,v})$ sont les coordonnées d'un point dans cette paramétrisation locale.

Rappelons que ce plan tangent définit également un approximant linéaire local de la surface autour du point $\mathbf{p}_i$ tandis que les attributs de la surface, tels que la couleur c_i au point $\mathbf{p}_i$, définissent un approximant local constant. Pour généraliser cela, appelons $\mathcal{P}_i^A(\mathbf{y}_i) : \mathbb{R}^2 \rightarrow \mathbb{R}^{dim(A)}$ la fonction approximant localement l'attribut générique A (position, couleur, normale, ...) au point $\mathbf{p}_i$. D'une manière générale, il est possible de calculer les paramètres des fonctions $\mathcal{P}_i^A$ par

approximation, au sens des moindres carrés par exemple, des attributs des plus proches voisins. De même, il est possible de prendre des fonctions plus évoluées que des polynômes linéaires pour les positions ou constants pour les autres attributs. Il est par exemple tout à fait possible d'utiliser des splats elliptiques (ou points différentiels) qui fournissent un polynôme d'approximation local quadratique de la position de la surface.

L'idée principale de Zwicker est alors de mélanger ces approximations locales via des fonctions de mélange lisses et locales combinées entre elles grâce à une paramétrisation globale du nuage de points.

Soit $\phi_i(\mathbf{y}_i) : \mathbb{R}^2 \rightarrow [0,1]$ le noyau de reconstruction locale associé au point $\mathbf{p}_i$ et défini dans le plan de référence locale τ_i. Intuitivement, ces noyaux définissent le niveau de confiance que l'on peut avoir en l'approximation locale de la surface et des attributs. Soit $\mathcal{M}_i : \mathbf{y}_i \rightarrow \mathbf{x}$ un morphisme inversible 2D/2D transformant les coordonnées locales $\mathbf{y}_i$ en coordonnées globales $\mathbf{x}$.

$$\phi_i'(\mathbf{x}) = \phi_i(\mathcal{M}_i^{-1}(\mathbf{x})) \qquad \text{et} \qquad \mathcal{P}_i^{A\prime}(\mathbf{x}) = \mathcal{P}_i^A(\mathcal{M}_i^{-1}(\mathbf{x})) \tag{2.3}$$

La reconstruction globale $S_A(\mathbf{x})$ d'un attribut A de la surface, paramétrisée par les coordonnées globales $\mathbf{x}$ est alors définie par la somme pondérée et normée suivante :

$$S_A(\mathbf{x}) = \frac{\sum_i \phi_i'(\mathbf{x}) \cdot \mathcal{P}_i^{A\prime}(\mathbf{x})}{\sum_i \phi_i'(\mathbf{x})} \tag{2.4}$$

Un scénario typique est d'utiliser des morphismes $\mathcal{M}_i$ linéaires, des fonctions d'approximation $\mathcal{P}_i^A$ constantes ou linéaires et des noyaux ϕ_i Gaussiens qui ont tous la particularité d'être C^∞ continus. Dans ce cas, la fonction $S_A(\mathbf{x})$ est également C^∞ continue.

Bien sûr, cette reconstruction n'est pas interpolante, c'est-à-dire que $S_A(\mathcal{M}_i((0,0))) \neq \mathcal{P}_i^A((0,0))$. Mais le principal inconvénient de cette définition de surface est qu'elle nécessite une paramétrisation globale du nuage de points. Trouver un morphisme inversible est également problématique mais dans une moindre mesure si l'on considère un noyau de reconstruction à support compact. Une projection orthogonale peut alors être suffisante. De plus, la reconstruction dépend de la paramétrisation globale et du morphisme.

En pratique, il est généralement nécessaire de se restreindre à une petite portion de l'objet pour laquelle il est possible de définir une paramétrisation globale du morceau de surface considéré via une simple projection sur un plan de référence. Côté applications, cela permet de :

- Définir des outils d'éditions locaux tels que des opérateurs de peinture, de sculpture, de ré-échantillonnage, de filtrage, ... [ZPKG02, Zwi03],
- Appliquer une texture 2D à un nuage de points en demandant à l'utilisateur de définir sa propre paramétrisation globale [ZPKG02, Zwi03],
- Reconstruire une image continue lors du rendu via l'algorithme de *surface splatting* [ZPvBG01, Zwi03]. Une paramétrisation globale est alors définie par projection des points sur le plan de la caméra.

2.1.1.6 Les "Moving Least Squares Surfaces"

La technique des *moving least squares* (MLS) est une autre méthode de reconstruction ou d'approximation de surface continue à partir d'un nuage de point non uniforme. Tout comme le mélange paramétrique des splats vue à la section précédente, la principale motivation des MLS est de définir une surface continue à partir d'une représentation purement discrète qu'est

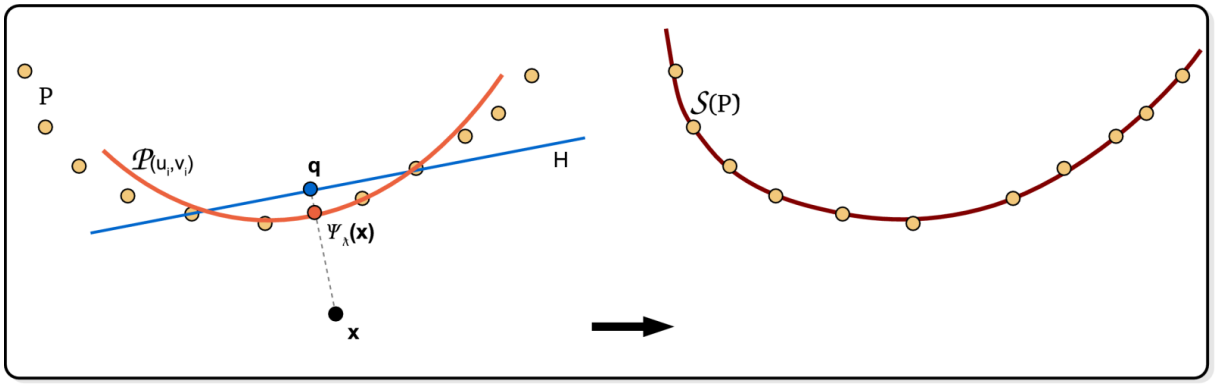


FIG. 2.4 – Illustration de la procédure de projection par MLS.

le nuage de points. Les MLS ont été introduits dans un premier temps pour la reconstruction de fonctions par Lancaster et Salkauskas [LS81] puis étendus à la reconstruction de surface manifold par Levin [Lev01]. Les premières applications à l'informatique graphique ont été proposées par Alexa et al. [ABCO+01].

Soit $P = \{\mathbf{p}_i\}$ un nuage de points représentant une surface lisse et manifold $\mathcal{S}$. La surface $\mathcal{S}(P)$ reconstruite par la méthode des MLS est définie par l'ensemble des points stationnaires d'un opérateur de projection Ψ_P, c'est-à-dire

$$\mathcal{S}(P) = \{\mathbf{x} \in \mathbb{R}^3 \mid \Psi_P(\mathbf{x}) = \mathbf{x}\}. \tag{2.5}$$

L'opérateur de projection Ψ_P, illustré figure 2.4, projette un point quelconque de l'espace sur la surface et la surface reconstruite corrspond donc à l'ensemble des points se projetant sur eux même. Le calcul de la projection $\Psi_P(\mathbf{x})$ est réalisé en trois étapes :

- Un plan de référence local H défini par la normale $\mathbf{n}$ et le scalaire d

  $$H = \{\mathbf{y} \in \mathbb{R}^3 \mid \mathbf{y} \cdot \mathbf{n} - d = 0\} \tag{2.6}$$

  est calculé de manière à minimiser la somme pondérée des distances au carré des points par rapport au plan :

  $$\sum_i (\mathbf{p}_i \cdot \mathbf{n} - d)^2 \phi(\|\mathbf{p}_i - \mathbf{q}\|) \tag{2.7}$$

  où $\mathbf{q}$ est la projection orthogonale de $\mathbf{x}$ sur le plan H et $\phi : \mathbb{R}^+ \to \mathbb{R}$ est une fonction de poids lisse, définie positive, monotone et décroissante. Remarquons que dans ce problème d'optimisation non linéaire, la seule inconnue est en fait la variable $\mathbf{q}$ puisqu'elle permet de déterminer $\mathbf{n}$ et d via les relations suivantes :

  $$\mathbf{n} = \frac{\mathbf{x} - \mathbf{q}}{\|\mathbf{x} - \mathbf{q}\|} \quad \text{et} \quad d = \mathbf{q} \cdot \mathbf{n}$$

- Ce plan de référence H définit un système de coordonnées locales où $\mathbf{q}$ est l'origine. Soient (u_i, v_i, e_i) les coordonnées du point $\mathbf{p}_i$ dans ce système de coordonnées, c'est-à-dire, (u_i, v_i) sont les coordonnées paramétriques dans H et e_i est l'élévation de $\mathbf{p}_i$ par rapport à H. Ensuite, une approximation locale par un polynôme bivarié $\mathcal{P} : H \to \mathbb{R}^3$ est calculée de manière à minimiser la somme pondérée suivante :

  $$\sum_i (\mathcal{P}(u_i, v_i) - e_i)^2 \phi(\|\mathbf{p}_i - \mathbf{q}\|) \tag{2.8}$$

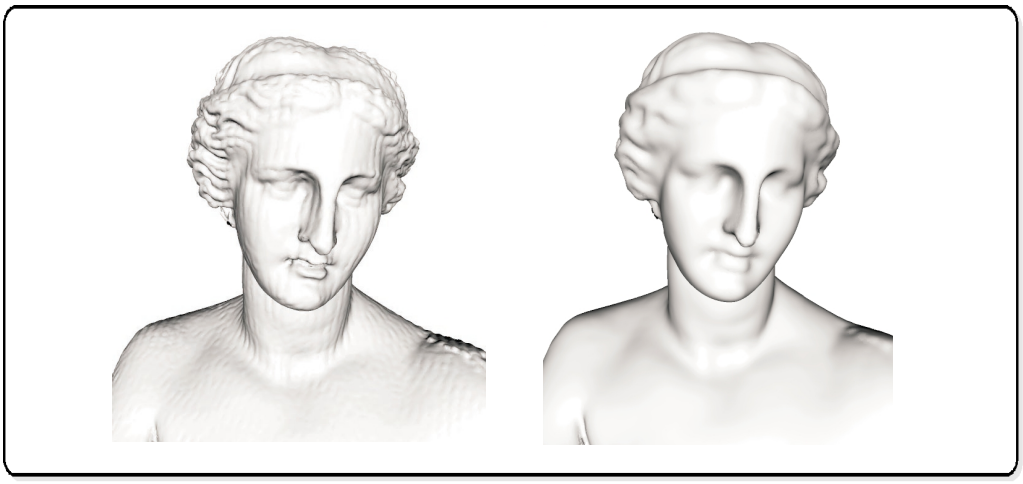


FIG. 2.5 – Reconstruction par MLS pour différente largeur h du noyau. (source [ABCO+01])

- Finalement, la projection de $\mathbf{x}$ sur la surface $S(P)$ est définie par

$$\Psi_P(\mathbf{x}) = \mathbf{q} + \mathcal{P}(0,0)\mathbf{n} \tag{2.9}$$

  Bien que souvent considérée comme telle, remarquons que la normale de la surface en ce point n'est pas la normale $\mathbf{n}$ du plan de référence calculé mais doit être déduite des dérivées du polynôme $\mathcal{P}$ trouvé à l'étape précédente.

Cette projection est illustrée figure 2.4. Un choix typique pour la fonction de poids ϕ est de prendre la Gaussienne

$$\phi(x) = e^{\frac{x^2}{h^2}} \tag{2.10}$$

où h est un facteur d'échelle global qui détermine la largeur du noyau. En général, celui-ci correspond à l'espacement moyen entre les points et permet de contrôler le degré de lissage, comme l'illustre la figure 2.5. Pour les nuages de points non globalement réguliers, Pauly et al. [PKKG03, Pau03] proposent de choisir la valeur de h dynamiquement afin de s'adapter à la densité locale.

La principale difficulté est le calcul du plan de référence H qui nécessite un processus d'optimisation non linéaire. Pour cela, plusieurs approches sont possibles. Dans [ABCO+01] Alexa et al. utilisent des itérations de Powell pour calculer les paramètres $\mathbf{n}$ et d alors que Pauly et al. [PGK02] utilisent des itérations de Newton pour estimer $\mathbf{q}$ directement, cette dernière étant moins rigoureuse mais plus simple et plus rapide à calculer.

Variante des surfaces MLS Une alternative considérablement plus simple à cette procédure de projection a été proposée par Adamson et Alexa dans [AA03a, AA04b] où une estimation correcte de la normale est utilisée afin de définir une surface implicite à partir du nuage de points. Leur procédure de projection, illustrée figure 2.6, consiste à projeter itérativement un point $\mathbf{x}$ sur une suite de plans de références locaux passant par les points $\mathbf{a}(\mathbf{x})$ et de normales

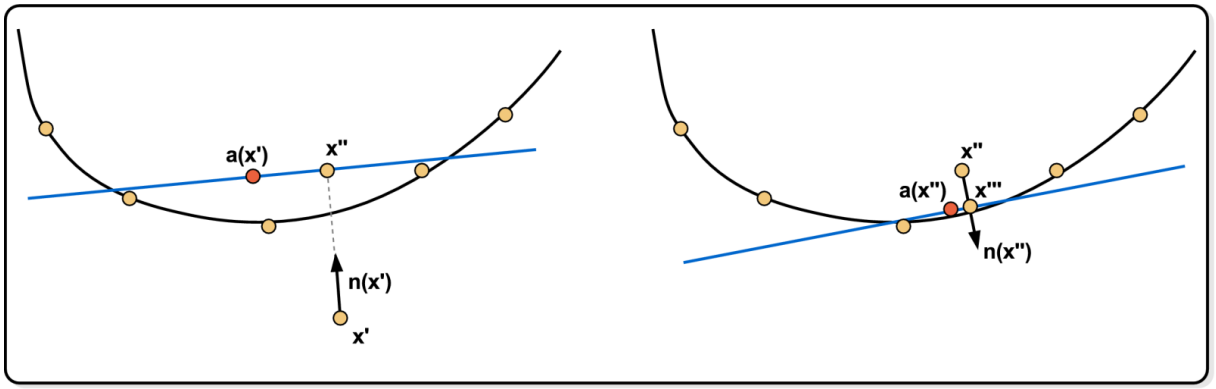


FIG. 2.6 – Illustration de la procédure de projection simplifiée. À chaque pas, l'approximation courante $\mathbf{x}'$ est mise à jour par une projection orthogonale sur le plan de référence déterminé par $\mathbf{a}(\mathbf{x}')$ et $\mathbf{n}(\mathbf{x}')$.

$\mathbf{n}(\mathbf{x})$ définis par les sommes pondérées suivantes :

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_i \phi(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{p}_i}{\sum_i \phi(\|\mathbf{x} - \mathbf{p}_i\|)} \tag{2.11}$$

et

$$\mathbf{n}(\mathbf{x}) = \frac{\sum_i \phi(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{n}_i}{\|\sum_i \phi(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{n}_i\|} \tag{2.12}$$

où $\mathbf{n}_i$ est la normale du point $\mathbf{p}_i$. Si cette normale n'est pas disponible alors $\mathbf{n}(\mathbf{x})$ peut être estimé par la minimisation suivante :

$$\mathbf{n}(\mathbf{x}) = argmin \sum_i |\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}_i)|^2 \phi(\|\mathbf{x} - \mathbf{p}_i\|) \tag{2.13}$$

Une procédure de projection itérative possible est alors :

1. Initialiser $\mathbf{x}' \leftarrow \mathbf{a}(\mathbf{x})$.
2. Calculer $\mathbf{n} \leftarrow \mathbf{n}(\mathbf{x}')$ et $\mathbf{a} \leftarrow \mathbf{a}(\mathbf{x}')$.
3. Si $|\mathbf{n} \cdot (\mathbf{a} - \mathbf{x}')| < \epsilon$ alors retourner $\mathbf{x}'$.
4. Sinon, projeter $\mathbf{x}' \leftarrow \mathbf{x}' + \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}_i)\mathbf{n}$ et recommencer à partir de l'étape 2.

Le résultat de cette procédure est le point $\mathbf{x}'$, projection du point $\mathbf{x}$ sur la surface MLS. Cependant, cette projection simplifiée n'est pas orthogonale bien que cela puisse être amélioré par de légères variantes proposées par Alexa et Adamson [AA04b].

Une définition implicite de la surface définie par cet opérateur de projection est donnée par la fonction de potentiel $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ suivante :

$$f(\mathbf{x}) = (\mathbf{x} - a(x)) \cdot \mathbf{n}(\mathbf{x}) \tag{2.14}$$

La surface $\mathcal{S}$ correspond donc à l'iso-valeur 0, c'est-à-dire $\mathcal{S} = \{\mathbf{x} \,|\, f(\mathbf{x}) = 0\}$. Afin d'augmenter l'ordre d'approximation, une variante possible est d'ajuster des polynômes d'ordre supérieur au lieu de simples plans [AA03a].

Par opposition à une construction algorithmique de la surface par des opérateurs de projection, Amenta et Kil [AK04] proposent une définition explicite des surfaces MLS. La surface est alors définie en terme de points critiques de la fonction d'énergie 2.7 le long de lignes déterminées par un champ de vecteurs. Notons que leur méthode permet de prendre en compte des points orientés. Dans leurs travaux, ils discutent également de la stabilité des opérateurs de projection traditionnels pour les points insuffisamment proches de la surface et proposent une procédure de projection alternative.

Toutes ces méthodes basées sur les MLS supposent que le nuage de points représente une surface lisse. Afin de palier à cette limitation, Fleishman et al. [FCOS05] ont proposé une technique de détection des arêtes franches au sein d'un nuage de points pouvant être bruité. Leur technique est basée sur des méthodes statistiques robustes.

Pour résumer, toutes ces méthodes basées sur les MLS permettent d'obtenir de très bons résultats en terme de continuité et d'oscillation de la surface reconstruite, même en cas de nuages de points bruités puisque le lissage de la surface est très facilement contrôlable par la taille du support de la fonction de poids ϕ. D'un point de vue applicatif, les surfaces MLS permettent donc le lissage/débruitage des nuages de points, le ré-échantillonnage de la surface (simplification ou raffinement) où encore la visualisation via des algorithmes de lancer de rayon. La définition de la surface par une procédure de projection est en effet très pratique, mais malgré toutes les simplifications proposées, son coût reste non négligeable. Actuellement, il est donc difficile d'utiliser une telle représentation dans des applications temps réel, c'est-à-dire si trop de projections doivent être calculées à chaque image, que ce soit pour la visualisation ou l'édition du nuage de points. Un bon compromis est alors d'utiliser une représentation hybride splats/MLS comme utilisée par Pauly dans [PKKG03] : la représentation par splat est utilisée pour l'étape de visualisation et pour effectuer les opérations ne nécessitant pas une grosse précision. Lorsque la représentation par splats n'est plus assez précise, par exemple pour déterminer si un point proche de la surface est à l'intérieur ou à l'extérieur de l'objet alors une approximation par MLS est réalisée.

Remarquons que les surfaces MLS ont quelques points communs avec la reconstruction paramétrique précédente puisque dans les deux cas la surface est approchée localement par des polynômes mélangés globalement par des fonctions de poids. Le principal avantage des MLS est d'être indépendant de toute paramétrisation. Une comparaison approfondie de ces deux approches peut être trouvée dans [Zwi03].

2.1.2 Les nuages de points structurés

Pour des raisons de compacité et d'efficacité, les premières méthodes utilisant les points comme primitives de base n'étaient pas basées sur une simple liste de points désorganisés, mais au contraire, proposaient de stocker les points dans un espace discret de telle sorte que deux ou même trois des coordonnées 3D des points deviennent implicites.

2.1.2.1 Les images de profondeurs

Dans cette première catégorie, les points sont stockés dans des images 2D associées à un repère 3D, tel que les coordonnées x et y des points dans le repère de l'image deviennent implicites. Chaque pixel représente alors un point 3D en stockant la profondeur du point, c'est-à-dire sa

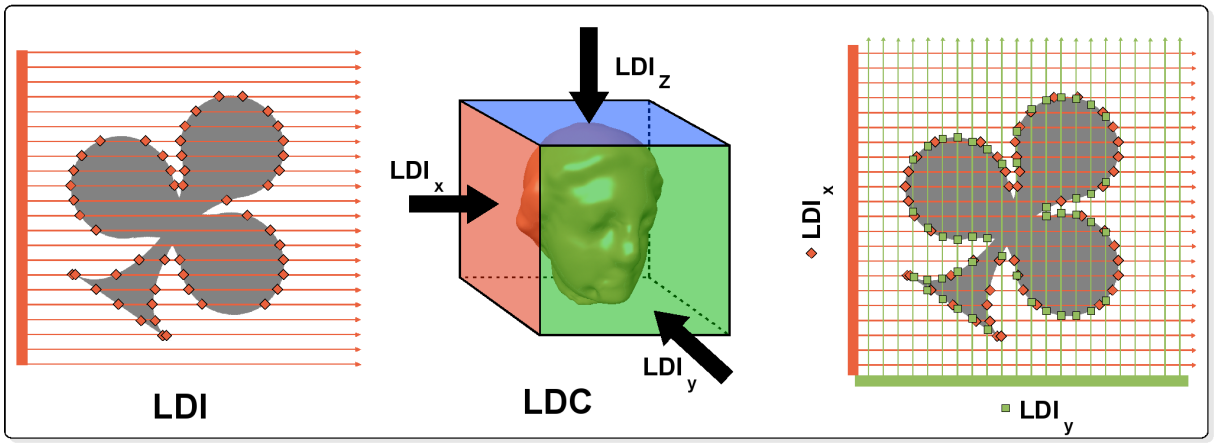


FIG. 2.7 – Illustration en 2D de l'échantillonnage par une LDI. Au centre : un LDC est un arrangement de trois LDI. À droite illustration en 2D de l'échantillonnage par un LDC.

coordonnée z ainsi que quelques attributs, typiquement une couleur et une normale utilisées pour le calcul de l'éclairage.

Une seule image n'est évidemment pas suffisante pour représenter un objet tout entier. Grossman et Dally [Gro98] proposent d'utiliser de nombreuses images de profondeur (dans leur cas 32) acquises à partir de différents points de vue positionnés régulièrement autour de l'objet. Afin de réduire la redondance induite par les différentes vues, les images sont découpées en petits blocs de 8x8 pixels. Un sous-ensemble de blocs non redondants est ensuite construit en sélectionnant itérativement les blocs parmi la liste des vues orthographiques. Un bloc est sélectionné si le morceau de surface qu'il représente n'est pas déjà correctement représenté par les blocs précédemment sélectionnés. Un premier inconvénient de cette approche est qu'il est possible que certaines parties de l'objet ne soient pas représentées car non visibles des points de vue utilisés.

Une autre approche consiste à utiliser des images de profondeur en couches communément appelées LDI pour "Layered Depth Images". Initialement proposée par Shade et al. [SGwHS98] en 1998, une LDI est simplement une image de profondeur où chaque pixel contient une liste de points se projetant sur le même pixel de l'image. En clair, une image de profondeur classique stocke uniquement les premières intersections entre la surface de l'objet et les rayons issus des pixels de l'image alors qu'une LDI stocke toutes les intersections. Cependant, une seule LDI ne permet pas un échantillonnage correct de la surface dans toutes les directions puisque les zones tangentielles à la direction de la LDI seront sous-échantillonnées. Lischinski et Rappaport proposent d'utiliser trois LDI orthogonales entre elles et appellent cet arrangement un "Layered Depth Cube" (LDC) [LR98].

Finalement, Pfister et al. proposent une méthode pour éliminer la redondance induite par l'utilisation de trois directions orthogonales appelée "3-to-1 réduction" [PZvG00]. La première étape de cette opération est de ré-échantillonner les points aux positions entières d'une grille de n^3, en assumant que les LDI ont une taille de n^2. Pour cela, ils prennent simplement l'échantillon le plus proche bien qu'il ne soit pas plus difficile d'utiliser un filtrage plus sophistiqué. Les nouveaux échantillons peuvent alors être stockés dans une unique LDI. Cette opération a pour

effet d'uniformiser la répartition des points puisque la distance entre deux points voisins est nettement moins variable puisque comprise entre :

- 0 et $\sqrt{3}h$ avant la réduction,
- h et $\sqrt{3}h$ après la réduction.

Ici h exprime l'espace entre deux pixels. En contrepartie, la précision/qualité de la représentation est nettement dégradée. En effet, avec un LDC, malgré la quantification de deux coordonnées sur trois, les points sont positionnés exactement sur la surface, ce qui n'est plus le cas ici. Cette réduction revient à quantifier les trois coordonnées de la position des points. Ceux-ci sont donc situés à une distance de la surface comprise entre 0 et $\frac{\sqrt{3}}{2}h$.

Finalement, après avoir quantifié les trois coordonnées de la position des points on peut se demander s'il est nécessaire de stocker explicitement une des coordonnées comme c'est le cas ici. Cette question fait l'objet des paragraphes de la section suivante.

2.1.2.2 Représentation volumique d'une surface

Mise à part le LDC, les méthodes précédentes étaient basées sur une quantification des deux coordonnées x et y de la position. Seule la troisième coordonnée, z, et les attributs des points devaient être explicitement stockés. En quantifiant les trois coordonnées x,y et z, et en stockant les points dans une grille 3D, seuls les attributs auraient à être stockés explicitement, les positions des points devenant implicites. Cependant, nous ne nous intéressons qu'à la surface de l'objet, de nombreuses cellules de la grille seraient alors vides. Cette représentation est donc inefficace. En effet, en utilisant une grille de résolution n la complexité serait de $O(n^3)$ pour représenter $O(n^2)$ échantillons alors que le coût de stockage explicite des coordonnées n'est que de $O(n^2 log(n))$.

Botsch et al. [BWK02] proposent une représentation hiérarchique efficace d'un nuage de points quantifié de la sorte. À partir d'une grille régulière binaire représentant implicitement les positions des points de la surface, un octree est construit en regroupant récursivement les cellules huit par huit. Après l'élimination des branches vides et un encodage entropique, le coût de cette représentation est optimal puisque de $O(n^2)$ (environ 1.5 bits par point !). Cependant, ce coût de stockage n'est que théorique et ne concerne que la position des points puisqu'en pratique il est nécessaire de stocker également les attributs des points avec au minimum une normale pour le calcul de l'éclairage.

Parmi les avantages de ces types de représentations dites *structurées*, on peut citer la compacité que ce soit pour l'octree de Botsch et al. ou celles basées images qui permettent de réutiliser les algorithmes classiques de compression d'images pour compresser les données. Pour le rendu, le principal avantage est que ces représentations permettent d'accélérer le rendu via des algorithmes de projection incrémentaux et l'utilisation de calculs sur les entiers. Cependant, ces avantages ne sont valables que pour un rendu logiciel puisque les processeurs graphiques prennent en entrée uniquement des positions 3D explicites. Or, la différence de puissance entre les processeurs graphiques (GPU) et les meilleurs CPU est devenue telle, que les GPU sont maintenant incontournables dès qu'il est question de rendu temps-réel. Ces types de représentations n'offrent pas non plus suffisamment de flexibilité, et sont principalement limités aux applications de rendu de scènes statiques. Il est en effet très difficile de réaliser une opération d'édition aussi simple que déplacer un point ! Ces représentations sont donc très peu adaptées aux applications d'animation ou d'édition d'objets modélisés par points.

2.2 Visualisation des nuages de points

Question : Comment visualiser un objet représenté par un nuage de points ?

Nous allons maintenant nous intéresser aux méthodes de visualisation de nuages de points proprement dites. D'une manière générale en synthèse d'image, deux catégories de techniques de visualisation s'opposent : les approches dites de lancer de rayon et les approches par rastérisation. Les méthodes de visualisation de nuages de points n'échappant pas à la règle, nous avons choisi de classifier les différentes méthodes sur ce critère.

2.2.1 Les approches "lancer de rayon" *(backward warping)*

Question : Comment intersecter un nuage de points ?

Dans cette première catégorie de méthodes, la problématique de la visualisation est posée de la manière suivante : étant donné un pixel de l'image à calculer, quel est le (ou les) morceau de surface responsable de sa couleur ? Cette catégorie correspond aux méthodes dites de lancer de rayon. Basiquement, pour chaque pixel de l'image à calculer, il s'agit de trouver le point d'intersection le plus proche de l'observateur entre la demi-droite issue du centre de projection et passant par ce pixel, et les surfaces des objets de la scène. Cette demi-droite est appelée rayon. Dans notre contexte, la problématique principale est le calcul de l'intersection entre un rayon et un nuage de points. Un rayon n'a en effet presque aucune chance de passer exactement par un point de l'échantillonnage.

Hormis les techniques de reprojection inverse d'images utilisant un stockage des points sous forme d'images de profondeur [BCD+99],la première proposition d'un lancer de rayons sur un nuage de points désorganisés revient à Schaufler et Jensen [SJ00]. Leur approche est basée sur une représentation par splats uniformement répartis dans l'espace de sorte que la distance r entre les points peut être connue de manière globale. Le calcul de l'intersection entre un rayon et la géométrie est réalisé en deux étapes : détection puis évaluation. La détection est effectuée en entourant le rayon par un cylindre de rayon r et en recherchant le point le plus proche de l'observateur à l'intérieur de ce cylindre. Cette recherche est accélérée en stockant les points dans un octree. Une fois qu'une intersection est détectée, le cylindre associé au rayon est coupé par deux plans orthogonaux au rayon et positionnés, l'un au niveau du point d'intersection et l'autre à ϵ dans la direction de visée (figure 2.8-a). L'ensemble des points à l'intérieur de ce petit cylindre est collecté et utilisé pour calculer plus précisement les attributs (position, couleur et normale) du point d'intersection. Pour cela, ils proposent de mélanger les attributs par la moyenne pondérée suivante :

$$attrib_{intersection} = \frac{\sum_i attrib_i * (r - d_i)}{\sum_i (r - d_i)} \tag{2.15}$$

où d_i est la distance entre le point $\mathbf{p}_i$ et l'intersection entre le rayon et le plan tangent de $\mathbf{p}_i$. Notons que, comme pour les algorithmes de splatting, cette reconstruction est dépendante du point de vue. Côté performances, leur implantation non optimisée est trois à quatre plus lente qu'un programme d'intersection rayon/triangle optimisé. Pour fixer les ordres de grandeurs, Schaufler et Jensen reportent un temps de rendu de 36s pour un modèle de 500 000 points et une image de 512^2.

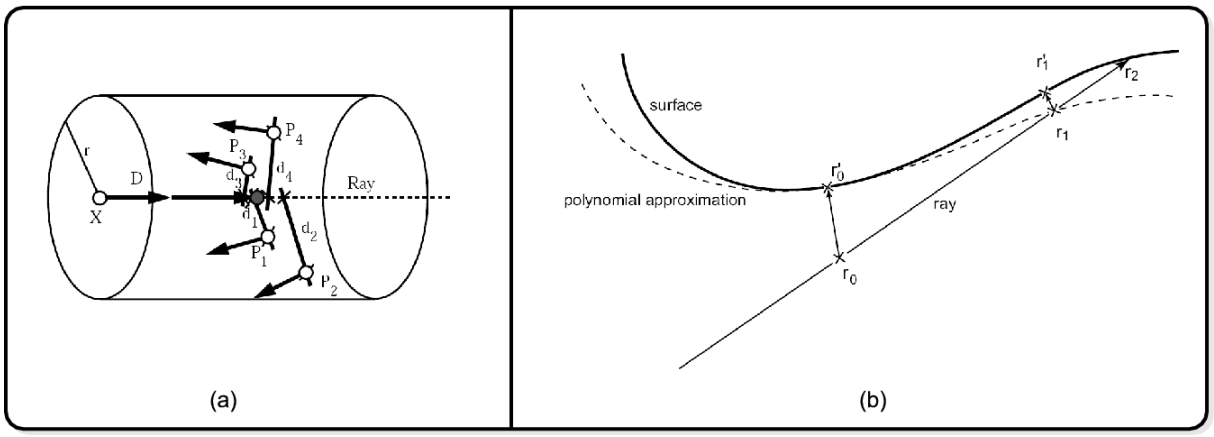


FIG. 2.8 – Intersection d'un rayon avec un nuage de points.
(a) Méthode de Schaufler et Jensen : les attributs des points contenue dans un petit cylindre autour du rayon sont moyennés. (source [SJ00]) (b) Méthode par MLS : à partir d'une première approximation de l'intersection, une approximation polynomiale de la surface est calculée localement. Une nouvelle approcimation de l'intersection rayon/points est donnée par l'intersection du rayon avec l'approximation polynomiale. (source [AA03b])

Les seconds travaux dans ce domaine sont ceux de Adamson et Alexa en 2003 [AA03b, AA03a]. Comparés à la méthode d'intersection précédente qui est plutôt empirique, leurs travaux sont basés sur une définition de la surface par MLS ou dérivée (section 2.1.1.6).

Dans [AA03b], le calcul de l'intersection entre un rayon et une surface MLS est réalisée en deux temps. Tout d'abord, l'intersection entre le rayon et le voisinage tubulaire de la surface est calculée. Ce voisinage tubulaire est représenté par une union de sphères englobant les points de la surface. Pour limiter le nombre de calculs d'intersection rayon/sphère, ces sphères sont stockées dans un octree. À partir de ce point d'intersection $\mathbf{x}_0$, une approximation polynomiale de la surface est calculée de la même manière que pour calculer la projection du point $\mathbf{x}_0$ sur la surface MLS (section 2.1.1.6). Ensuite, une nouvelle approximation de l'intersection $\mathbf{x}_1$ est donnée par l'intersection entre l'approximation polynomiale et le rayon. Ce processus est répété tant que la distance entre la surface et le point d'intersection calculé $\mathbf{x}_i$ est supérieure à ϵ (figure 2.8-b). Bien sûr, si lors de ce processus itératif, aucune intersection polynôme/rayon n'est trouvée, alors le processus doit être recommencé avec l'intersection rayon-sphère suivante. À cause du processus d'optimisation non linéaire requis pour déterminer le plan de référence support de l'approximation polynomiale, les temps de calculs reportés sont assez conséquents, de l'ordre de quelques heures par images. Ces faibles performances sont cependant à relativiser puisque Adamson et Alexa ont utilisé une implémentation de l'opérateur de projection existante loin d'être optimisée. En fait, un résultat plus intéressant est que seulement 2 ou 3 itérations par calcul d'intersection sont nécessaires pour une erreur maximale de $\epsilon = 10^{-13}$.

Peu de temps après, Adamson et Alexa ont proposé une variante de la définition des surfaces MLS évitant le processus d'optimisation non linéaire [AA03a]. Cette variante, présentée précédemment section 2.1.1.6, permet de réduire significativement les temps de calcul, de l'ordre d'une dizaine de secondes pour une image 200×400. En contrepartie, l'erreur pour un même nombre d'itérations augmente significativement, de l'ordre de 10^{-3} pour 3 itérations. Cela s'explique par le fait que la procédure d'optimisation non linéaire de la version précédente nécessitait également plusieurs itérations à chaque itération du processus de calcul d'intersection.

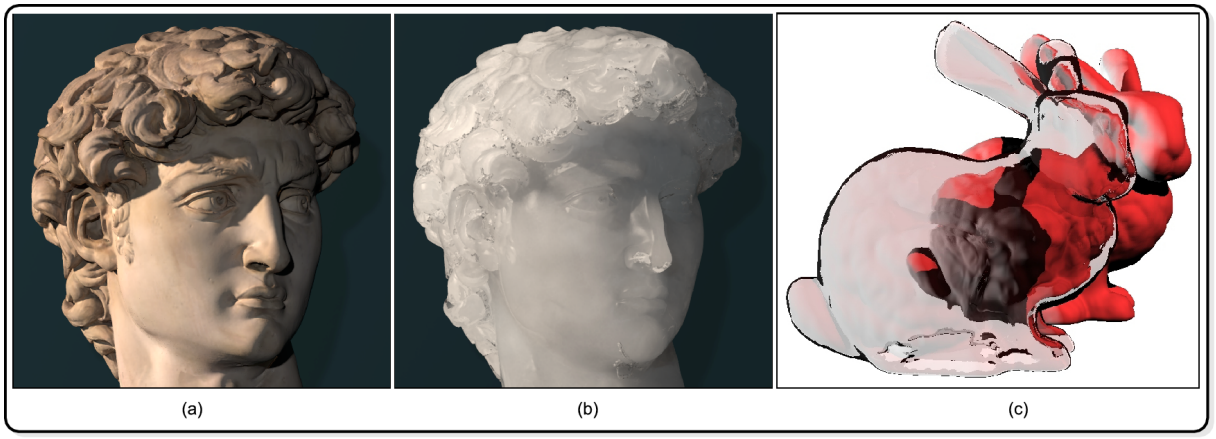


FIG. 2.9 – Les méthodes par lancer de rayons permettent de mettre en place des algorithmes de calculs de l'éclairage évolués.
(a) Illumination globale. (source [SJ00]) (b) Sub-surface scatering. (source [SJ00]) (c) Réfraction. (source [AA03b])

En s'appuyant sur la définition implicite d'Adamson et Alexa précédente, Wald et Seidel [WS05] ont récemment réussi à obtenir des temps de calcul interactifs, entre 2 et 6 images par seconde. Les ingrédients utilisés pour obtenir de telles performances sont :

- Le modèle de surface implicite est combiné à une représentation par splats optimisée de sorte que les rayons des splats soient minimaux tout en assurant une couverture complète de la surface. Les splats sont stockés dans un *kd*-tree optimisé, c'est-à-dire chaque voxel du *kd*-tree stocke la liste des splats l'intersectant.
- Un parcours rapide du *kd*-tree, permet de déterminer les voxels où une intersection rayon-surface est possible. Lorsqu'un tel voxel est trouvé, alors l'intersection précise entre le rayon et la surface est calculée via la définition d'une surface implicite. Mais au lieu d'une recherche des k plus proches voisins, seuls les points du voxel sont utilisés. De plus, au lieu d'utiliser un processus itératif, N valeurs de la fonction implicite sont calculées pour N échantillons uniformément répartis le long du rayon. Le point d'intersection est obtenu par interpolation linéaire entre les deux valeurs de signe opposé. En pratique, prendre $N = 4$ permet une implantation efficace via les instruction SIMD des processeurs actuels.

Avec une approche similaire à celle d'Adamson et Alexa, Adams et al. [AKP+05] ont proposé un algorithme de lancer de rayon optimisé pour les surfaces déformables [MKN+04]. Les points clés de leur méthode sont :

- Utilisation d'une hiérarchie de sphère englobante construite à partir des points. Les rayons des sphères de la hiérarchie sont mis à jour dynamiquement en fonction des déformations.
- Les relations de voisinage entre chaque voisin sont précalculées, évitant ainsi une recherche des k plus proches voisins à chaque calcul d'intersection.
- Prise en compte de la cohérence spatio-temporelle entre chaque image en mémorisant pour chaque pixel la sphère intersectée.

L'intérêt majeur de toutes ces méthodes d'intersection pour le lancer de rayon est bien sûr de permettre la mise en place d'algorithmes complexes de calcul de l'éclairage réaliste offrant des images de très haute qualité (figures 2.9). D'un point de vue géométrique, ces méthodes

requièrent tout de même un nuage de points suffisement dense pour que la surface reconstruite par MLS (ou dérivée) soit consistante. Si les conditions d'échantillonnage sont satisfaites, alors la reconstruction est lisse et les images d'une grande qualité visuelle. Cette qualité du rendu a cependant un coût important qui invalide ces approches pour tous types d'applications nécessitant une visualisation interactive.

2.2.2 Les approches z-buffer / *forward warping*

Question : Comment tracer un nuage de points ?

Par opposition aux approches précédentes, nous allons maintenant considérer les méthodes de rendu par projection des objets sur l'écran. La problématique du rendu est alors posée de la manière suivante : quels sont les pixels recouverts par la projection de la primitive courante ? Dans ces approches, les primitives sont donc considérées une à une, les pixels à travers lesquels la primitive courante est potentiellement visible sont déterminés par projection géométrique de celle-ci dans l'espace image. Cette primitive projetée est ensuite convertie en fragments (pixels potentiellement visibles) par une phase de discrétisation usuellement appelée *rastérisation*. La visibilité est finalement effectuée via un algorithme de z-buffer : un fragment n'est conservé que s'il est plus proche de l'observateur que le fragment précédemment écrit.

Lorsque la densité d'un nuage de points est suffisante, le processus de rendu devient trivial puisqu'il est alors suffisant de projeter chaque point sur un seul pixel de l'image pour obtenir une image sans trou. En réalisant les calculs d'éclairage par primitives, c'est-à-dire pour chaque point, on obtient alors un éclairage de haute qualité puisque équivalent à un éclairage par pixel. Cependant, dans la pratique, l'échantillonnage du nuage de points ne correspond jamais à l'échantillonnage du plan de la caméra et deux problèmes peuvent se poser :

- **Aliassage** en cas de réduction, c'est-à-dire lorsque la densité des points est telle que plusieurs points visibles sont projetés sur un même pixel. En effet, seul le point le plus proche sera conservé et une partie de la géométrie et texture de l'objet ne sera pas prise en compte dans le calcul de l'image finale.
- **Trous** ou image incomplète en cas d'agrandissement, c'est-à-dire lorsque au contraire la densité de points n'est pas suffisante. En effet, dès que l'espacement entre les points voisins dans l'espace image dépasse un pixel, le fond de l'image ou les objets de second plan sensés être non visibles apparaissent entre les échantillons visibles.

Finalement, lorsque nous considérons des points, le calcul de la visibilité, c'est-à-dire déterminer les points ou contributions des points réellement visibles devient un troisième problème fondamental. Comme nous le verrons par la suite, ces trois problèmes sont en fait très intimement liés les uns aux autres.

2.2.2.1 Raffinement dans l'espace objet

Le principe commun des méthodes de cette catégorie est de raffiner dynamiquement le nuage de points, c'est-à-dire d'augmenter le nombre de points pour que la densité soit localement suffisante pour permettre un rendu naïf sans trous. Ces méthodes ne faisant qu'accroître le nombre de points de la représentation, celles-ci ne sont pas incompatibles avec les autres approches de rendu basé points proposant des mécanismes de filtrage et d'anti-aliassage que nous verrons dans les sections suivantes.

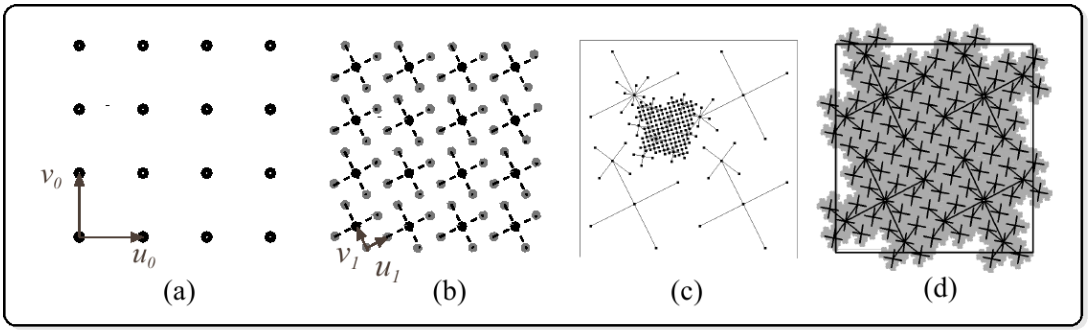


FIG. 2.10 – Le raffinement $\sqrt{5}$ (source [SD01]). (a) Grille initiale. (b) Grille après un pas de raffinement. (c) Raffinement adaptatif. (d) Zone fractale recouverte par les échantillons.

Cependant, raffiner un nuage de points nécessite généralement de connaître la surface sous-jacente afin de positionner les nouveaux points sur cette surface. Si cette surface est connue et n'est pas une représentation basée points alors nous ne pouvons pas vraiment parler de rendus des points mais plutôt de rendus par points.

En 2001, Stamminger et Drettakis [SD01] et Wand et al. [WFP+01] ont proposé en même temps deux techniques très similaires d'échantillonnage dynamique de maillages polygonaux. La technique du *randomized z-buffer* [WFP+01] nécessite une phase de précalcul au cours de laquelle les faces triangulaires sont triées spatialement dans un octree. Au moment du rendu, cet octree permet de former dynamiquement des groupes de faces ayant un facteur d'échelle proche. Ensuite, une estimation du nombre total d'échantillons nécessaires à la visualisation est réalisée pour chaque groupe. Cette estimation est basée sur la distance entre l'observateur et le volume englobant du groupe et sur l'aire totale des triangles du groupe. Les échantillons sont ensuite choisis aléatoirement sur les triangles avec une fonction de densité de probabilité proportionnelle à la surface des triangles. Afin d'accélérer le rendu, les échantillons ponctuels sont stockés dans un cache.

La technique de Stamminger et Drettakis [SD01] est vraiment très similaire puisque également basée sur un échantillonnage aléatoire. En phase de précalcul, une liste de points est générée par un tirage aléatoire. Au moment du rendu, le nombre d'échantillons N nécessaires à la visualisation de l'objet est estimé en fonction de l'aire de la surface de l'objet et de la distance entre l'objet et l'observateur. Si N est inférieur au nombre de points déjà présents dans la liste, alors un préfixe de N points est tracé. Dans le cas contraire, les échantillons manquants sont facilement générés par tirage aléatoire.

À cause d'un échantillonnage aléatoire de la surface, ces méthodes sont plutôt adaptées à la visualisation d'objets complexes de type arbres ou autres végétaux.

Dans le même article, Stamminger et Drettakis ont aussi proposé une méthode d'échantillonnage dynamique de terrain et surfaces procédurales. La clé de cette technique est un schéma de raffinement adaptatif $\sqrt{5}$ qui ne nécessite pas d'information de connectivité entre les point (figure 2.10). En revanche, les points initiaux doivent être uniformément répartis suivant une paramétrisation 2D de la surface. De plus, la représentation géométrique utilisée doit permettre la projection de n'importe quel point de l'espace sur la surface représentée.

Plus proche de la problématique de visualisation des nuages de points, Alexa et al. [ABCO+03] ont proposé d'associer à chaque point une approximation polynomiale de la surface définie dans

le plan tangent. Cette approximation est calculée par la méthode des *moving least squares*. Au moment du rendu, ces approximations polynomiales sont dynamiquement échantillonnées de manière à garantir une visualisation sans trou. Durant la phase de précalcul, en plus de l'évaluation des approximations polynomiales, il est nécessaire d'uniformiser la répartition des points ainsi que d'évaluer les domaines des polynômes afin de minimiser les risques de discontinuités et le sur-échantillonnage dû au chevauchement des approximations locales. En plus de la phase de précalcul et des éventuels problèmes de discontinuités, cette approche pose aussi les problèmes du coût de stockage et de l'interpolation des attributs de la surface comme la couleur dans le cadre d'objets texturés.

2.2.2.2 Reconstruction de l'image

Par opposition aux méthodes précédentes, nous allons présenter ici les techniques de reconstruction dans l'espace image. Après projection des points sous la forme de simples pixels, les éventuels trous sont détectés puis reconstruits par interpolation ou filtrage des échantillons les plus proches. Une première difficulté est alors la classification des pixels qui peuvent être soit valides (c'est-à-dire atteints par un point réellement visible) soit à reconstruire. Ceci est à relier aux classiques problèmes de calcul de visibilité en synthèse d'images. Pour ce type d'approche par projection de l'objet vers l'écran, le challenge est donc d'avoir un z-buffer cohérent, c'est-à-dire sans trous, permettant d'éliminer les points non visibles. Le second challenge est, pour un pixel donné à reconstruire, de trouver puis d'interpoler les pixels valides les plus proches.

Pour résoudre le problème de la visibilité, Grossman et Dally [GD98, Gro98] ont proposé d'utiliser d'une hiérarchie de z-buffer à résolution décroissante. Au moment du rendu, pour chaque point à tracer, le z-buffer ayant une résolution suffisamment faible pour que la taille du point (ou l'espacement local entre les points) dans l'espace image soit inférieure à un pixel, est sélectionné. Le point est alors tracé à la fois dans le tampon d'image (couleur, normale et profondeur) et dans le z-buffer sélectionné. Après avoir tracé tous les points, une comparaison des profondeurs des différents niveaux est effectuée afin de détecter les pixels corrects des pixels à reconstruire. Afin de ne pas éliminer des points effectivement visibles, cette comparaison doit être effectuée avec un seuil de tolérance ϵ. Afin de réduire le crénelage au niveau des silhouettes, le résultat de cette comparaison est en fait, pour chaque pixel, un coefficient réel entre 0 et 1 qui indique à quel point le pixel est visible ou non.

La reconstruction des trous est alors effectuée par un algorithme *pull-push* adapté de l'algorithme de Gortler [GGSC96]. Dans la phase de *pull* une série d'images à résolutions décroissantes est calculée en utilisant les coefficients précédents pour moyenner les pixels entre eux. La somme des coefficients est également stockée en chaque pixel. Dans la phase de *push*, l'image à faible résolution est utilisée pour reconstruire les pixels incomplets (c'est-à-dire ayant un coefficient inférieur à 1) de l'image de résolution supérieure. Cet algorithme de reconstruction est illustré figure 2.11.

Le principal intérêt de cette approche est l'utilisation d'une hiérarchie de z-buffer simplifiant à l'extrême la rastérisation des points puisque réduite à un pixel. Mais il s'agit également de l'inconvénient majeur puisque cela signifie que la projection d'un point orienté est approchée dans l'espace image par un carré parallèle aux axes, d'une taille d'une puissance de deux et pas nécessairement centré sur la projection du point. Cela génère donc un aliassage très fort dès que la projection des points dépasse quelques pixels.

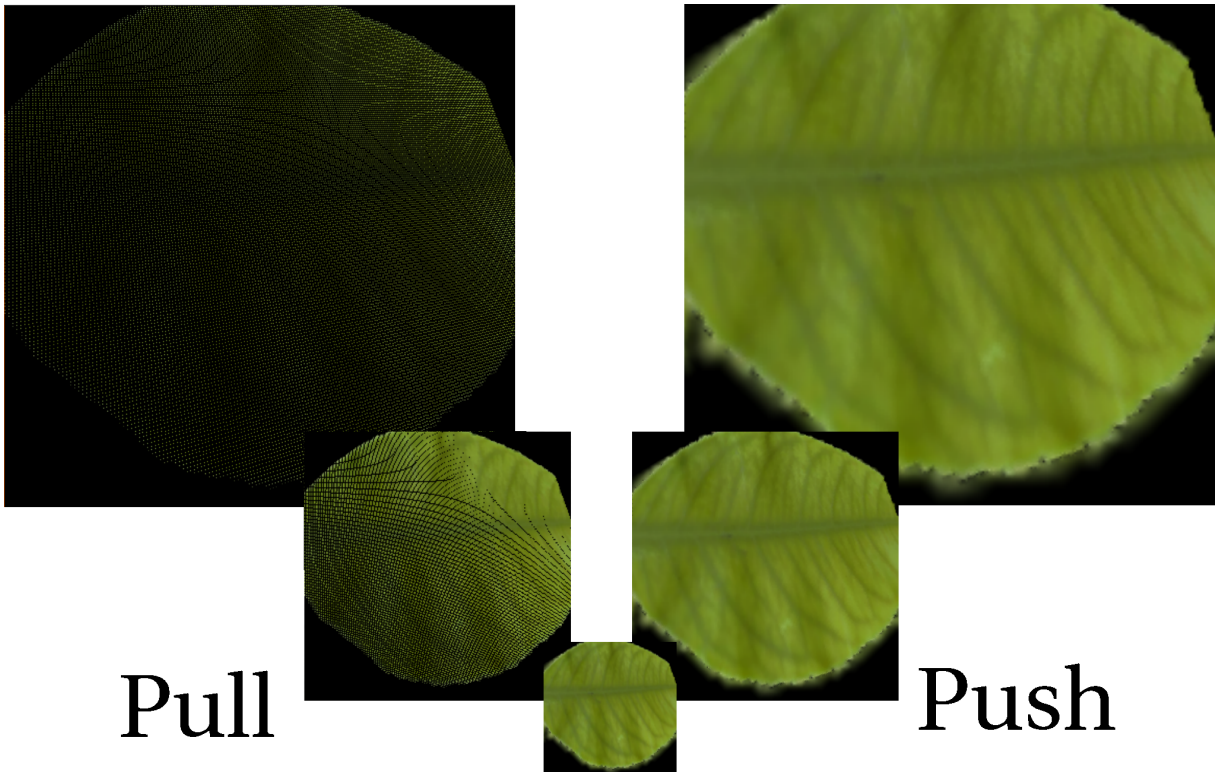


FIG. 2.11 – Reconstruction par pull-push.

À partir de ces travaux, Pfister et al. ont donc proposé de reconstruire plus précisement le z-buffer en utilisant une technique appelée *visibility splatting*. Bien qu'utilisant une représentation par splats associée à un processus de splatting pour résoudre le problème de la visibilité, nous avons classé cette technique dans cette catégorie car la reconstruction proprement dite est bien effectuée exclusivement dans l'espace image en s'appuyant sur une représentation purement ponctuelle.

Un point est alors projeté sur un seul pixel dans le tampon chromatique mais sur plusieurs pixels du z-buffer via un vrai processus de rastérisation. Pour des raisons d'efficacité, la projection d'un point ou splat dans l'espace image est approchée par un parallélogramme aligné sur au moins un axe. Durant cette rastérisation, la valeur présente dans le z-buffer est comparée à la valeur calculée. Si cette différence est trop importante, c'est-à-dire supérieure à ϵ, le pixel est alors marqué comme étant à reconstruire. Le remplissage des trous est alors effectué en interpolant la couleur des pixels valides les plus proches. Pour cela, un filtre Gaussien symétrique est placé au centre de chaque pixel à reconstruire, le rayon du filtre étant estimé localement à partir d'espacement entre les points dans l'espace image.

Finalement, une des caractéristiques de ces méthodes est de séparer les calculs de visibilité de la reconstruction de l'image. Ainsi, les coûteux calculs de reconstruction ne sont réalisés que pour les pixels qui ont réellement besoin d'être reconstruits. Ces algorithmes de rendu sont donc conceptuellement particulièrement efficaces. Malheureusement, ceux-ci ne sont actuellement implantables que par voie logicielle et ne peuvent bénéficier d'une implantation accélérée par les GPU. De plus, ces algorithmes utilisent une reconstruction isotrope dans l'espace image et ne peuvent donc prendre en compte de larges agrandissements sans une forte dégradation de la

qualité, comme par exemple : aplats de couleur, flou exagéré ou encore crénelage de la silhouette. Les mécanismes d'anti-aliassage sont quant à eux absents dans la méthode de Grossman et Dally ou nécessitent une phase de précalcul pour celle de Pfister et al., cette dernière étant limitée à un filtrage isotrope.

2.2.2.3 Splatting

Les méthodes de visualisation par splatting associent une reconstruction C^{-1} par splats (disques ou ellipses) dans l'espace objet à un mélange lisse des splats dans l'espace image. Nous pouvons ainsi qualifier ces méthodes comme étant intermédiaires entre les approches de reconstruction purement dans l'espace objet ou purement dans l'espace écran. D'un côté, le chevauchement des splats dans l'espace objet est suffisant pour obtenir un rendu sans trou dans l'espace image. D'un autre côté, le rendu naïf de splats s'interpénétrant a pour conséquence des discontinuités de couleur dans l'image finale. Les splats doivent ainsi être mélangés entre eux lors du rendu.

La méthode de splatting faisant référence est celle de Zwicker et al. appelée *EWA surface splatting* où EWA signifie *elliptical weighted average* [ZPvBG01]. Leur algorithme de rendu combine la représentation paramétrique à base de splats présentée section 2.1.1.5 pour gérer les agrandissements avec une technique d'anti-aliassage anisotrope à base de filtres de reconstruction Gaussiens elliptiques. Cette technique d'anti-aliassage a été inspirée par des travaux de Greene et Heckbert [GH86, Hec89]. Rappelons que dans cette représentation, chaque splat de position $\mathbf{p}_i$ est associé à une normale $\mathbf{n}_i$. Les vecteurs $\mathbf{u}_i$ et $\mathbf{v}_i$ définissent une paramétrisation du plan tangent où $\mathbf{y}_i = (y_{i,u}, y_{i,v})$ sont les coordonnées d'un point dans cette paramétrisation locale. Les attributs du point (position, normale, couleur, ...) sont approchés localement par les polynômes $\mathcal{P}_i^A(\mathbf{y}_i)$ où A dénote un attribut. Dans ce contexte de rendu, la position est approchée par le plan tangent tandis que les autres attributs sont approchés par des polynômes constants. Chaque point est également associé à un noyau de reconstruction local $\phi_i(\mathbf{y}_i)$ déterminé uniquement par la donnée d'un scalaire représentant l'espacement local entre les points.

Dans le contexte du rendu, l'objectif est de reconstruire une surface continue dans l'espace image en utilisant l'équation 2.4. L'image est alors générée en échantillonnant cette reconstruction. Pour cela, les splats, c'est-à-dire les approximations $\mathcal{P}_i^A(\mathbf{y}_i)$ et noyaux $\phi_i(\mathbf{y}_i)$ sont projetés dans l'espace image qui est utilisé comme domaine de paramétrisation globale pour le mélange des splats entre eux. Le point clé est alors de définir le morphisme 2D $\mathcal{M}_i$ qui transforme les coordonnées locales $\mathbf{y}_i$ en coordonnées de l'écran $\mathbf{x}_i$. De plus, afin d'éliminer l'aliassage dû a l'échantillonnage dans l'espace image, les fonctions reconstruites doivent vérifier le critère de Nyquist. Aussi, un filtre passe-bas h supprimant les hautes fréquences est appliqué à la reconstruction par convolution. Finalement, la fonction de reconstruction lissée $\tilde{S}_A$ de l'attribut A est :

$$\tilde{S}_A(\mathbf{x}) = (S_A \otimes h)(\mathbf{x}) \approx \frac{\sum_i (\phi'_i \otimes h)(\mathbf{x} - \mathbf{x}_i)\mathcal{P}_i^{A'}(\mathbf{x} - \mathbf{x}_i)}{\sum_i (\phi'_i \otimes h)(\mathbf{x} - \mathbf{x}_i)} \tag{2.16}$$

où $\mathbf{x}_i$, $\phi'_i(\mathbf{x})$, $\mathcal{P}_i^{A'}(\mathbf{x})$ sont respectivement les projections dans l'espace image du point $\mathbf{p}_i$, du noyau $\phi_i(\mathbf{y}_i)$ et de l'approximant $\mathcal{P}_i^A(\mathbf{y}_i)$. C'est à dire :

$$\mathbf{x}_i = \mathcal{M}_i((0,0)) \ , \quad \phi'_i(\mathbf{x}) = \phi_i(\mathcal{M}_i^{-1}(\mathbf{x})) \ , \quad \mathcal{P}_i^{A'}(\mathbf{x}) = \mathcal{P}_i(\mathcal{M}_i^{-1}(\mathbf{x})) \tag{2.17}$$

La transformation $\mathcal{M}_i$ doit prendre en compte toutes les transformations géométriques de la paramétrisation locale d'un point à l'espace écran. L'étape de projection perspective n'est malheureusement pas linéaire, ce qui rend impossible la dérivation d'un filtre de ré-échantillonnage Gaussien. Aussi, Zwicker et al. proposent de remplacer $\mathcal{M}_i$ par son approximation linéaire locale $\overline{\mathcal{M}}_i$, qui est donnée par les deux premiers termes du développement de Taylor :

$$\overline{\mathcal{M}}_i(\mathbf{y}_i) = \mathbf{x}_i + \mathbf{J}_i \cdot \mathbf{y}_i \tag{2.18}$$

où $\mathbf{J}_i$ est le Jacobien de $\mathcal{M}_i$ au point $\mathbf{y}_i = (0,0)$, c'est-à-dire :

$$\mathbf{J}_i = \frac{\partial \mathcal{M}_i}{\partial y_i}(0,0) = [...] \tag{2.19}$$

En pratique, les dérivées partielles définissant le Jacobien $\mathbf{J}_i$ sont calculées en transformant les vecteurs tangents $\mathbf{u}_i$ et $\mathbf{v}_i$ dans l'espace image. Voir [ZPvBG01] ou [RPZ02] pour plus de détails.

En utilisant des Gaussiennes pour les noyaux de reconstruction $\phi_i(\mathbf{y}_i) = g_{\mathbf{R}_i}(\mathbf{y}_i)$ et pour le filtre passe-bas $h(\mathbf{x}) = g_{\mathbf{H}}(\mathbf{x})$, il est alors possible de dériver une Gaussienne pour le filtre de ré-échantillonnage $\rho_i(\mathbf{x}) = (\phi'_i \otimes h)(\mathbf{x} - \mathbf{x}_i)$. Soit $g_{\mathbf{V}}(\mathbf{x})$ une Gaussienne 2D de matrice de covariance $\mathbf{V} \in \mathbb{R}^{2\times 2}$, $g_{\mathbf{V}}(\mathbf{x})$ est définie par :

$$g_{\mathbf{V}}(\mathbf{x}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x}} \tag{2.20}$$

Pour être en accord avec la résolution de l'image de sortie, le filtre passe-bas a généralement une variance unitaire, c'est-à-dire $\mathbf{H} = \mathbf{I}$. La matrice de covariance des filtres de reconstruction est quant à elle dépendante de la densité locale. Si r_i correspond à l'espacement moyen entre le point $\mathbf{p}_i$ et ses voisins, alors nous pouvons prendre :

$$\mathbf{R}_i = \begin{pmatrix} r_i^2 & 0 \\ 0 & r_i^2 \end{pmatrix} \tag{2.21}$$

La projection du noyau de reconstruction dans l'espace image est donnée par la Gaussienne :

$$\phi'(\mathbf{x}) = \phi_i(\mathbf{J}_i^{-1} \cdot x) = \frac{1}{|\mathbf{J}_i^{-1}|} g_{\mathbf{J}_i\mathbf{R}_i\mathbf{J}_i^T}(\mathbf{x}) \tag{2.22}$$

La convolution de cette Gaussienne par la Gaussienne du filtre passe-bas donnant le filtre de ré-échantillonnage est toujours une Gaussienne :

$$\begin{aligned} \rho_i(\mathbf{x}) \quad &= (\phi'_i \otimes h)(\mathbf{x} - \mathbf{x}_i) \quad = \frac{1}{|\mathbf{J}_i^{-1}|} g_{\mathbf{J}_i\mathbf{R}_i\mathbf{J}_i^T + H}(\mathbf{x} - \mathbf{x}_i) \\ &\qquad\qquad \frac{1}{|\mathbf{J}_i^{-1}|} g_{\mathbf{V}_\rho}(\mathbf{x} - \mathbf{x}_i) \end{aligned} \tag{2.23}$$

La reconstruction d'un nuage de points attribués est alors finalement résumée par l'équation suivante :

$$(S_A \otimes h)(\mathbf{x}) \approx \frac{\sum_i \rho_i(\mathbf{x}) \cdot \mathcal{P}_i^{A'}(\mathbf{x} - \mathbf{x}_i)}{\sum_i \rho_i(\mathbf{x})} \tag{2.24}$$

Bien sûr, dans l'équation précédente seules les contributions réellement visibles doivent être sommées entre elles.

Le rendu d'un nuage de points est alors effectué de la manière suivante :

Pour chaque point $\mathbf{p}_i$, calculer les paramètres du filtre de ré-échantillonnage ρ_i et déterminer le rectangle englobant aligné aux axes contenant l'ellipse représentant ρ_i. Le support de ρ_i est en effet limité aux pixels de l'image pour lesquels l'exposant $\mathbf{x}^T {\mathbf{V}_\rho}^{-1} \mathbf{x}$ de ρ_i est inférieur à un seuil donné. Ce seuil est déterminé afin de limiter le recouvrement des splats. Pour chaque pixel $\mathbf{x}_k$ de l'englobant, la profondeur $z_{i,k} = \mathcal{P}_i^{\mathbf{p}'}(\mathbf{x}_k - \mathbf{x}_i)_z$ du splat est évaluée et comparée à ϵ près à la valeur $z(\mathbf{x}_k)$ présente dans le z-buffer. Appelons fragment le morceau du splat courant correspondant au pixel courant et le n-upplet $\left(\rho_i(\mathbf{x}), \rho_i(\mathbf{x}){\mathcal{P}_i^{A_1}}'(\mathbf{x} - \mathbf{x}_i), \ldots, \rho_i(\mathbf{x}){\mathcal{P}_i^{A_{n-1}}}'(\mathbf{x} - \mathbf{x}_i)\right)$ la contribution de ce fragment ($A_1, \ldots, A_{n-1}$ dénotent les $n-1$ attributs du point qui sont reconstruits). Trois possibilités sont alors à envisager :

- si $|z_{i,k} - z(\mathbf{x}_k)| < \epsilon$, le fragment appartient au même morceau de surface que le pixel courant. La contribution est donc ajoutée aux données déjà présentes.
- si $z_{i,k} - z(\mathbf{x}_k) > \epsilon$, le fragment n'est pas visible et la nouvelle contribution est simplement oubliée.
- si $z_{i,k} - z(\mathbf{x}_k) < -\epsilon$, le fragment est bien plus proche que les fragments des contributions précédentes. Les données stockées au pixel courant sont alors remplacées par la nouvelle contribution.

Après avoir tracé tous les points, chaque pixel $\mathbf{x}$ du tampon de destination contient la somme des poids $\sum_i \rho_i(\mathbf{x})$ et la somme pondérée de chacun des attributs A. La reconstruction lisse d'un attribut est alors obtenue en divisant la somme pondérée de l'attribut par la somme des poids. Cette division est généralement appelée *normalisation*. Nous n'avons plus qu'à appliquer un modèle d'éclairage en chaque pixel et afficher l'image résultante. En pratique, seule la couleur, la normale et la profondeur ont à être reconstruites pour un éclairage par pixels avec le modèle de Phong et des sources de lumières locales.

Afin de pouvoir prendre en compte des surfaces semi-transparentes, Zwicker et al. ont adapté le *Z^3-algorithm* de Jouppi et Chang [JC99] qui est lui-même une adaptation du célèbre *A-buffer* de Carpenter [Car84]. Le principe est d'utiliser plusieurs tampons de destination et rendre chaque couche de surface dans un tampon différent. Les couches sont ensuite combinées par alpha blending. Les objets semi-transparents peuvent ainsi être rendus sans se préoccuper de l'ordre du tracé.

Pour résumer, cette technique permet un rendu des nuages de points avec une très bonne qualité, principalement grâce aux caractéristiques suivantes :

- Reconstruction lisse des attributs en cas d'agrandissement.
- Filtrage anisotrope de haute qualité en cas de sur-échantillonnage (élimine tous les effets d'aliassage).
- Éclairage par pixel en reconstruisant la texture et les normales de la surface avant d'appliquer le modèle d'éclairage (principe appelé "deferred shading").
- Permet de rendre des surfaces semi-transparentes.

En contrepartie, les coûts de calcul sont relativement élevés, et avec un nombre de splats tracés par seconde inférieur à un million, les performances ne permettent généralement pas une navi-

gation temps-réel.

Implantation du splatting sur GPU

En parallèle ou à partir de ces travaux précédant, de nombreuses variantes visant à utiliser les capacités des cartes graphiques ont été proposées. Apportant à chaque fois un peu plus de flexibilité, chaque nouvelle génération de carte graphique fut ainsi accompagnée de nouveaux algorithmes de splatting. Les différentes approches varient principalement sur la manière de tracer et de calculer la forme des splats dans l'espace-image, ou autrement dit, sur la manière de calculer et de tracer les filtres de ré-échantillonnage. Dans tous les cas, afin d'obtenir un rendu suffisamment lisse, les splats doivent être associés à un noyau de reconstruction Gaussien (ou similaire) permettant lors du rendu un mélange par accumulation des contributions visibles. Comme nous l'avons vu, accumuler uniquement les contributions visibles nécessite l'utilisation d'un z-buffer avec tolérance ou plus généralement une sorte de *A-buffer*. Aucune de ces fonctionnalités n'est malheureusement disponible sur les cartes graphiques actuelles. Bien que leur méthode soit antérieure aux précédents travaux, Rusinkiewicz et Levoy [RL00] ont proposé avec leur projet QSplat de simuler le test de profondeur flou par un rendu en deux passes.

- Lors de la première passe de rendu, généralement appelée splatting de visibilité (*visibility splatting*), un tampon de profondeur sans trou est précalculé en traçant les points avec un décalage de ϵ dans la direction de visée. Cette passe de rendu n'affecte que le tampon de profondeur et aucun calcul d'éclairage ou autres n'est effectué.
- Lors de la seconde passe, la mise à jour du tampon de profondeur doit être désactivée de sorte que tous les fragments visibles passent le test de profondeur, c'est-à-dire les fragments les plus proches de l'observateur à ϵ près. Les splats sont alors tracés avec des poids Gaussiens calculés par pixel et stockés dans la composante alpha des fragments. Le blending est activé de sorte à obtenir en chaque pixel RGBA une somme pondérée de couleur $\sum_i \alpha_i rgb_i$.

Dans le projet QSplat, les splats sont simplement tracés par des quadrilatères (ou triangles) auxquels sont appliquées une texture 2D Gaussienne.

En se basant sur ces premiers travaux, Ren et al. [RPZ02] proposent en 2002 une première implantation matérielle de l'EWA splatting. Comme précédemment, les contributions des splats sont accumulées dans l'espace-image en traçant pour chaque point un quadrilatère texturé par une Gaussienne 2D. Les positions des sommets d'un quadrilatère sont calculées dans l'espace-objet grâce à une reformulation du filtre de ré-échantillonnage de l'EWA splatting dans l'espace-objet. Tous ces calculs sont réalisés par un *vertex shader*[2]. Au moment de ces travaux, il n'était pas possible de réaliser de manière matérielle la passe finale de normalisation par la somme des poids. Bien qu'il soit possible de réaliser une normalisation par pixel par le CPU en stockant la somme des poids dans le canal alpha du tampon de destination, cette solution n'est pas satisfaisante en raison de la lenteur du transfert du tampon de couleur vers la mémoire centrale. Au lieu de cela, Ren et al. proposent de prénormaliser la contribution de chaque splat. Les facteurs de normalisation des splats sont obtenus en réalisant un très grand nombre de rendus de l'objet sous différents angles de vue. Bien qu'accélérant nettement le rendu par rapport à l'implantation logicielle de l'EWA surface splatting, cette approche a un inconvénient majeur qui est de quadrupler le nombre d'informations à stocker et à envoyer à la carte graphique, puisque

[2] Un *vertex shader* est un petit programme exécuté par la carte graphique pour chaque sommet. Ce programme permet entre autre de personnaliser les calculs de transformation et d'éclairage des sommets.

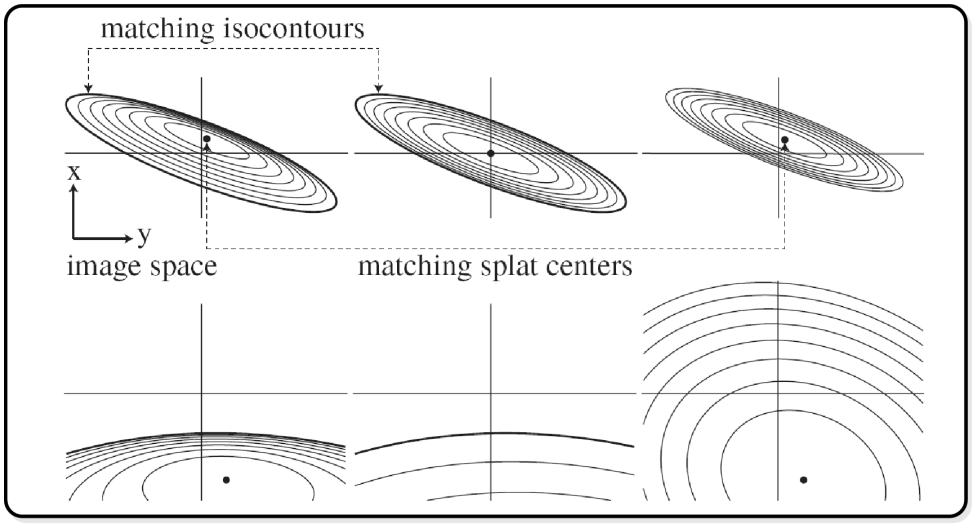


FIG. 2.12 – Les différentes approximations de la projection perspective. (source [ZRB+04])

qu'un point requiert maintenant quatre sommets ! Les calculs relativement coûteux du filtre de ré-échantillonnage sont donc également réalisés quatre fois par points.

L'apparition en 2003 des GeForce5 et des *fragment shaders*[3], a permis deux avancées majeures dans le rendu à base de points accéléré matériellement :

- La première est de permettre une normalisation des contributions par pixels vraiment efficace : lors de la seconde passe, les splats sont accumulés dans une texture au lieu de l'écran. Une troisième passe de normalisation consiste alors à tracer un quadrilatère de la taille de l'écran texturé avec le résultat de la passe précédente. Le *fragment shader* réalisant la normalisation est très simple, puisqu'il suffit de diviser les composantes RGB par la composante alpha, [GP03, BK03].

- Mais, la possibilité la plus intéressante avec l'apparition des *fragment shader* est la rastérisation de Gaussiennes elliptiques en utilisant un seul sommet par point et la primitive `point` des cartes graphiques. La primitive `point` entraîne la rastérisation d'un carré aligné aux axes de l'écran et de profondeur constante. Aussi, dans [GP03] et [BK03], un *vertex shader* est utilisé pour calculer la forme et la taille de la projection des splats dans l'espace-écran. Ces paramètres sont ensuite transmis à un *fragment shader* évaluant pour chaque pixel, la profondeur réelle du fragment ainsi que la contribution à accumuler. Pour calculer la forme des splats, Botsch et Kobbelt [BK03] approchent de manière globale la projection perspective par une projection orthographique. Pour notre part [GP03], nous avons repris la méthode originale de l'EWA surface splatting qui a l'avantage de fournir un anti-aliassage anisotrope ainsi qu'une meilleure approximation de la projection perspective puisque réalisée localement.

Cependant, toutes les méthodes que nous venons de voir utilisent une approximation affine de la projection perspective afin de calculer la forme des filtres de reconstruction dans l'espace

[3]Un *fragment shaders* est un petit programme exécuté par la carte graphique pour chaque fragment généré par la rastérisation. Ce programme permet entre autre de personnaliser l'application des textures et d'effectuer des calculs d'ombrage (éclairage) par pixels.

image. Cette simplification peut, dans certaines conditions, générer des trous dans l'image lorsqu'un objet, proche des bords de la fenêtre est vu de manière rasante. Ce problème fut attaqué la première fois par Zwicker et al. en 2004 [ZRB+04] avec leur technique appelée *accurate perspective splatting*. Leur technique utilise toujours une approximation affine, mais celle-ci est évaluée de telle sorte que que le contour externe du splat soit correctement projeté (au lieu du centre), figure 2.12. Bien que l'intérieur du splat soit légèrement incorrect, ce qui peut être problématique au niveau des arêtes, cette méthode permet effectivement de garantir un rendu sans trou. De plus, comme il s'agit toujours d'une approximation affine, celle-ci peut parfaitement être intégrée à l'EWA surface splatting et ainsi profiter d'un anti-aliasing de qualité.

Une projection perspective correcte est néanmoins tout à fait réalisable en calculant pour chaque pixel l'intersection exacte entre le splat et un rayon issu de ce pixel [BK04]. Aussi surprenant que cela puisse paraître, il s'agit sans doute d'une des méthodes les plus simples à implanter. En contrepartie, il n'est pas possible d'utiliser le filtrage anisotrope de l'EWA splatting.

Les méthodes de rendu que nous avons vues jusqu'ici utilisent un champ de normales constant par splats. Mise à part la version logicielle de l'EWA splatting dans laquelle couleurs de texture et normales sont filtrées ou interpolées avant les calculs d'éclairage résultant en un éclairage par pixel de haute qualité, les autres méthodes se contentent d'interpoler ou de filtrer le résultat du calcul de l'éclairage réalisé par splats. Les résultats sont alors comparables au lissage de Gouraud : bien que l'éclairement varie de manière lisse, le résultat semble flou particulièrement au niveau des reflets spéculaires. Une solution alternative au *deferred shading* avec interpolation des normales est alors d'utiliser des points différentiels, définissant un champ de normales non-constant (voir section 2.1.1.4), et un éclairage par pixel réalisé au niveau des *fragment shaders* de la seconde passe de splatting [KV01, KV03, BK04]. Les principaux inconvénients de ce type d'approche viennent directement de la représentation elle-même et ont déjà été discutés section 2.1.1.4. Un inconvénient supplémentaire est d'alourdir les *fragment shaders* par les calculs d'ombrage augmentant significativement les temps de rendu ($\times 5$ pour un modèle d'éclairage simple).

Un dernier point sur ces techniques de splatting concerne le rendu des bords, arêtes franches ou coins. Comme nous l'avons vu section 2.1.1.3, de telles discontinuités dans la surface peuvent être représentées en utilisant des splats tronqués par des lignes de découpe définies dans leur plan de référence local. Comme l'ont proposé Zwicker et al. [ZRB+04], de tels splats peuvent facilement être rendus en intégrant un test de découpe par pixel au niveau des *fragment shaders* des passes de splatting.

Parmi toutes les méthodes de visualisation des nuages de points, ces méthodes par splatting offrent clairement le meilleur compromis vitesse-qualité. D'un point de vue de la qualité géométrique de la visualisation, le splatting de surface est en effet très proche des méthodes par lancer de rayon et possède un très gros avantage en cas de réduction grâce à un mécanisme d'anti-aliasing performant. Bien sûr, la qualité de l'éclairage qu'il est possible d'obtenir n'est pas comparable puisque les approches par lancer de rayon supportent la plupart des algorithmes d'illumination globale. En plus des performances accrues, une approche par splatting offre aussi plus de flexibilité qu'une approche par lancer de rayon car tant que le nombre de primitives n'est pas trop important (jusqu'à quelques millions tout de même) le rendu en temps interactif du nuage de points peut être effectué sans aucun précalcul ni construction d'une quelconque structure de données accélératrice.

Toutefois finissons par un petit bémol, quelle que soit la méthode de visualisation utilisée la qualité du rendu d'un point de vue géométrique est tout de même sensible à la densité et à la régularité locale de l'échantillonnage. Ceci est particulièrement vrai pour les méthodes par splatting puisque lorsque la densité n'est pas suffisante, le rayon des splats dans l'espace-image devient grand, ce qui génère des artefacts aux niveaux de la silhouette et rend l'image flou à l'intérieur de l'objet. Les méthodes par lancer de rayon sont un peu moins sensibles à ce problème puisque basées sur une définition de surface indépendante du point de vue, mais cette définition requiert tout de même une certaine densité pour être consistante.

2.3 Méthodes pour le rendu temps-réel des scènes complexes

Question : Comment sélectionner les points à tracer ?

D'une manière générale, les algorithmes de type lancer de rayon (hautement optimisés) sont assez peu sensibles à la complexité de la scène. Un partitionnement spatial adéquat de la scène permet de n'avoir à calculer explicitement qu'une ou deux intersections par rayon.

En revanche, les méthodes par rastérisation ont par défaut une complexité linéaire avec le nombre de primitives de la scène puisque la visibilité est déterminée au moment même du processus de rastérisation via un z-buffer. Le nombre de pixels de l'image à calculer étant borné, il est clair que la totalité des primitives composant une scène ne peuvent être toutes visibles à la fois. Dès que le nombre de primitives composant la scène devient trop important il est donc primordial de sélectionner parmi l'ensemble des primitives un sous-ensemble de primitives potentiellement visibles qui soit le plus petit possible. Bien sûr, plus un algorithme de sélection sera précis et plus rapide sera le processus de rendu final puisque moins de primitives seront tracées. D'un autre côté, le coût du processus de sélection est dépendant de sa précision. Pour avoir un quelconque intérêt, ce processus de sélection doit être le plus rapide possible, tout au moins aussi rapide que le rendu. Il s'agit donc de trouver le meilleur compromis. Parmi ces algorithmes de sélection nous pouvons distinguer deux grandes familles :

- Les algorithmes d'élimination des parties cachées ou de *culling* qui permettent grâce de rapides tests de visibilité de savoir si un ensemble de primitives est potentiellement visible ou non.
- Les algorithmes de sélection des niveaux de détails dont l'objectif est de fournir pour un objet donné une représentation simplifiée et adaptée à la résolution de l'image calculée. En effet, lorsqu'un objet s'éloigne de l'observateur, la taille des primitives dans l'espace écran diminue jusqu'à ce que plusieurs primitives se projettent sur un seul pixel. Ces primitives peuvent alors être avantageusement remplacées par une seule, réduisant d'autant le nombre de primitives à tracer.

Pour être efficaces, ces algorithmes peuvent travailler au niveau de l'objet ou de manière plus précise, par groupes de primitives. Nous pouvons donc distinguer deux cas de figure. Dans le premier cas, tests de visibilité et sélections des niveaux de détails sont réalisés pour l'objet tout entier. Bien que simple et rapide, une telle approche n'est cependant que rarement utilisée car trop grossière. Il est en effet courant qu'un objet ne soit que partiellement visible et/ou que la densité de primitives nécessaires varie d'une partie de l'objet à l'autre. La plupart des méthodes travaillent donc sur des groupes de primitives qui sont avantageusement structurées

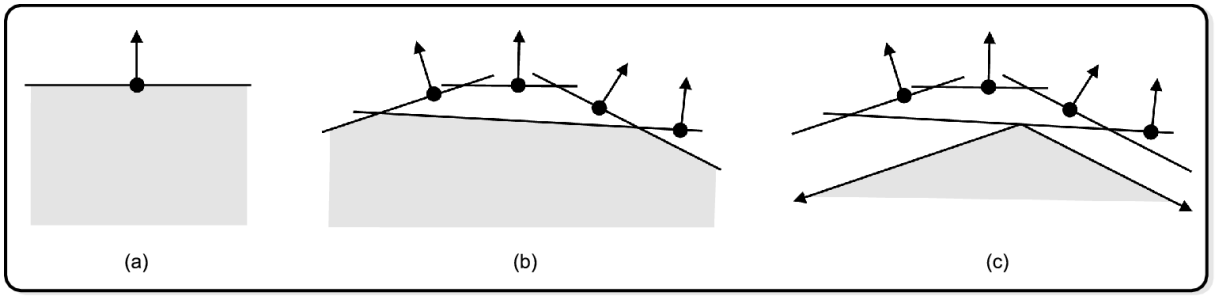


FIG. 2.13 – Illustration des cônes de normales.
(a) Un point orienté définit un demi-espace à partir duquel sa face externe ne peut être visible. (b) Intersection de ces demi-espaces pour un groupe de points. (c) Approximation de cette intersection par un cône.

au sein d'une hiérarchie dans laquelle tests de visibilités et sélection des niveaux de détails sont intimement liés.

Dans la suite, nous allons donc commencer par présenter les différents tests de visibilité qu'il est possible de réaliser sur des nuages de points, puis nous présenterons les différentes structures de données multi-résolutions qui ont été proposées.

2.3.1 Les tests de visibilité

D'une manière générale, les tests de visibilité que nous allons présenter sont réalisés sur un groupe de points approchés par un volume englobant. En général, les volumes englobants sont choisis pour être simples à manipuler et permettre une structuration hiérarchique (voir section suivante 2.3.2). Nous retrouvons donc principalement :

1. La boîte alignée aux axes définie par deux points 3D.
2. La sphère définie par un point et un rayon.

Nous distinguons trois types de tests de visibilité : le view-frustum culling, le back-face culling et l'occlusion culling.

2.3.1.1 View-frustum culling

Le view-frustum culling élimine les régions non visibles car situées à l'extérieur de la pyramide de vision. Ce test n'a donc rien de spécifique aux points puisqu'en pratique il suffit de tester l'intersection du volume englobant de l'objet avec la pyramide de vision. La pyramide de vision est définie par six plans, quatre plans passant par le centre de projection et les côtés du plan image, et deux autres définissant les distances minimale et maximale.

2.3.1.2 Back-face culling

Lorsqu'un objet fermé est représenté de manière surfacique, nous pouvons distinguer deux faces de la surface : la face interne (côté matière) et la face externe. Par définition, la face externe est la seule pouvant être visible. Le *back-face culling* consiste donc à éliminer les primitives surfaciques qui ne présentent à l'observateur que la face interne. Avec une représentation par point, il s'agit des points pour lesquels l'angle entre la direction de visée et la normale est supérieur à $\frac{\pi}{2}$.

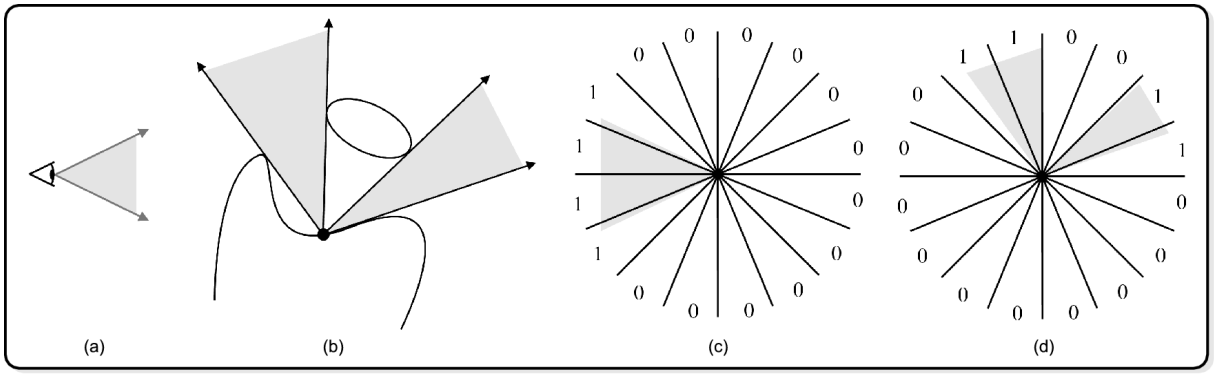


FIG. 2.14 – Illustration des masques de visibilités.
(a) Caméra. (b) Ensemble des directions à partir desquelles le point courant est visible. (c) Masque de visibilité associé à la caméra. (d) Masque de visibilité associé au point courant.

Lorsque nous considérons tout un groupe de points, nous avons besoin d'un moyen permettant de tester rapidement l'ensemble des orientations des points par rapport à la position de l'observateur. Une solution est la technique des cônes de normales, initialement proposée par Shirmun et al. [SAE93] et adaptée aux points par Grossman et Dally [Gro98]. Un cône de visibilité est associé à chaque groupe de points et représente un espace depuis lequel aucun point du bloc n'est visible (figure 2.13). Le cône de visibilité d'un groupe est construit à partir des normales et des positions des points du groupe. Basiquement, chaque point-normale définit un demi-espace à partir duquel le point ne peut être visible. Le cône de visibilité est alors choisi de manière à approcher au mieux et de manière conservative l'intersection de ces demi-espaces. Au moment du rendu, il suffit de tester si l'observateur est ou non à l'intérieur du cône.

Une seconde approche est de quantifier les directions des normales, en prenant N directions différentes et de classer l'ensemble des points d'un objet par normales [DVS03]. De cette manière, l'élimination des points en face arrière est extrêmement rapide. Le problème est qu'ainsi les points ne peuvent être organisés spatialement ce qui rend la réalisation des autres tests de visibilité par groupe de primitives impossible. De plus, cette méthode fait l'hypothèse que la direction de visée est constante pour tous les points de l'objet, ce qui est faux dans le cas d'une projection perspective.

2.3.1.3 Occlusion culling

Le but des tests d'occlusion culling est de supprimer les objets, ou parties d'objets non visibles car cachés derrière un autre objet. Il s'agit bien sûr du problème le plus difficile à résoudre efficacement, et peu de méthodes spécifiques aux points ont été proposées. Le problème des occlusions est cependant un très vaste domaine de recherche. Parmi la large variété des méthodes existantes [COCSD03] celles basées image peuvent assez facilement être adaptées à notre problématique. Ces méthodes sont en effet très génériques puisque s'appuyant principalement sur des volumes englobants contenant des primitives ratérisables par z-buffer.

Les masques de visibilité
En s'inspirant des masques de normales de Zhang et Hoff [ZKEH97], Grossman et Dally ont proposé l'utilisation de masques de visibilité [Gro98] pour gérer l'occlusion culling au sein d'un

objet (figure 2.14). Leur technique utilise un partitionnement de l'espace des directions obtenu par une subdivision de la sphère en 128 triangles. Ensuite, un masque de bits est associé à chaque groupe de points. Chaque bit correspond à un ensemble de directions, c'est-à-dire à un triangle de la subdivision. Le k^{me} bit d'un masque d'un groupe vaut 1 si et seulement si au moins un point du groupe est visible à partir d'une direction liée au k^{me} triangle. Les masques de visibilité sont calculés en réalisant de nombreux rendus de l'objet à partir de différents points de vue. Au moment du rendu un même masque est calculé pour le volume de visualisation et il suffit de réaliser un `ET` logique bit à bit entre les deux masques pour savoir si le bloc est visible ou non.

En pratique, cette technique est assez limitée puisque l'observateur doit être à l'extérieur de l'enveloppe convexe de l'objet et que seules les occlusions de l'objet avec lui-même sont partiellement prises en compte. De plus, la phase de précalcul des masques de visibilité est assez importante, ce qui limite cette technique aux objets rigides ne subissant pas de déformations. Notons que le back-face culling est également en partie pris en compte par ces masques de visibilité.

Occlusion culling dans l'espace image

Le principe de base de ces méthodes d'occlusion est de tester la visibilité d'un groupe de primitives par un processus de rastérisation du volume englobant. Ces requêtes peuvent être réalisées de deux manières différentes.

- Les premières méthodes utilisaient un z-buffer hiérarchique permettant de réduire le processus de rastérisation du volume englobant à seulement quelques pixels [GKM93, ZMHH97].
- À l'heure actuelle, les cartes graphiques permettent également de réaliser des requêtes d'occlusion sur le z-buffer courant. Ces requêtes ont l'avantage sur un z-buffer hiérarchique d'être beaucoup plus précises tout en étant accélérées par le matériel graphique. Notons au passage que certaines cartes graphiques gèrent en interne une hiérarchie de z-buffer.

Ces types de requêtes ne permettent cependant pas à elles seules d'éliminer efficacement les objets cachés puisque la visibilité d'un objet est testée par rapport au contenu du z-buffer courant. Il est également important d'essayer de limiter le nombre de requêtes à effectuer. Deux classes d'approches complémentaires sont alors possibles. Une première consiste à sélectionner parmi la liste des objets de la scène une liste d'occluders qui seront tracés en premier afin d'initialiser le z-buffer [ZMHH97]. La visibilité des autres objets peut alors être testée. La difficulté de cette approche est bien sûr la sélection des occluders qui idéalement doivent être au premier plan et occuper une large partie de l'écran. Une autre approche est de trier les groupes de primitives spatialement et de les tracer d'avant en arrière [GKM93]. Cela évite l'étape de sélection des occluders mais requiert un tri des objets problématique dans le cadre de scènes dynamiques.

2.3.2 Structures de données multi-résolutions

Bien qu'un grand nombre de structures de données aient été proposées afin d'accélérer le rendu à base de points, la plupart, pour ne pas dire toutes, partagent les mêmes principes de base. D'une manière générale, il s'agit de structures de données arborescentes dans lesquelles chaque noeud représente une portion de l'objet. L'union des noeuds d'un niveau donné l représente la totalité de l'objet, chaque niveau correspondant à un degré de simplification différent. Le niveau 0, c'est-à-dire les feuilles de la hiérarchie, correspond à la meilleure résolution de la représentation. Le noeud d'un niveau l représente une portion de l'objet correspondant à l'union de ses fils, mais avec une densité de points réduite.

Décrite ainsi, une telle structure de données hiérarchique permet, au moment du rendu :

- Une sélection locale des niveaux de détails : les niveaux de détails sélectionnés peuvent varier d'une zone de l'objet à une autre.
- D'effectuer des tests de visibilité de manière hiérarchique : un objet non visible dans sa totalité sera éliminé d'un seul coup lors du test du noeud racine, et au contraire, si l'objet n'est que partiellement visible, une descente dans la hiérarchie permettra d'éliminer une partie des portions non visibles.

La sélection des niveaux de détails adéquats est généralement réalisée en estimant la distance entre les points du noeud courant dans l'espace image. Si cette valeur est supérieure à un seuil donné alors la densité n'est pas suffisante et la récursion est appliquée sur les fils. D'une manière générale, au moment du rendu, le parcours récursif d'une telle structure de données hiérarchique est donc réalisé de la manière suivante :

```
procedure parcours_hiérarchique(Noeud n)
{
  si n est potentiellement visible alors
      si ( n est une feuille ou
             la densité des points de n est suffisante ) alors
          tracer les points de n;
      sinon
          pour chaque fils f de n faire
              parcours_hiérarchique(f);
      finsi
  finsi
}
```

Basées sur ce schéma général, les méthodes varient principalement sur le type de la hiérarchie utilisée (*kd*-tree, octree, sphère englobante), mais aussi sur la façon de stocker les points au niveau des noeuds et sur leur construction. Les premières propositions de structures de données multi-résolutions pour les représentations par points sont le LDC tree de Pfister et al. [PZvG00] et le QSplat de Rusinkiewicz et Levoy [RL00]. L'une étant basée sur un octree et l'autre sur une hiérarchie de sphères englobantes, ces deux approches sont suffisamment différentes pour servir de point de départ à une classification.

2.3.2.1 Les octrees et dérivés

Le LDC-tree est basé sur un partitionnement spatial sous forme d'octrees où chaque noeud est un LDC (voir section 2.1.2.1). Le LDC d'un noeud du niveau $l > 0$ est obtenu par l'union des LDC des huit fils dont la résolution a été divisée par 2, c'est-à-dire, seuls les pixels ayant des coordonnées paires sont conservés. Ainsi, le nombre de points par noeud est constant pour chaque niveau, puisque la résolution des images de profondeur composant les LDC est constante. Le choix de la taille des LDC est un compromis entre la précision et la rapidité de la sélection des noeuds à tracer. Dans leur implantation, Pfister et al. ont utilisé des blocs de $8 \times 8 \times 8$.

Les avantages et inconvénients du LDC-tree viennent directement de ceux des représentations par image de profondeur. D'un côté cela permet une projection incrémentale des points lorsque le rendu de ceux-ci est réalisé de manière logicielle. En contrepartie, nous pouvons noter

une perte de flexibilité, puisque qu'il s'agit d'une représentation très statique, ainsi qu'une totale inefficacité avec les méthodes de rendu accélérées par carte graphique.

Coconu et Hege [CH02] ont proposé une structure de données similaire dans laquelle chaque noeud de l'octree stocke les points dans une *loose* LDI (LLDI). Cette dernière est d'une certaine façon assez similaire à une LDI à la différence près que les coordonnées (x, y, z) d'un point sont explicitement stockées en chaque pixel, ce qui assure une très bonne précision dans chaque direction contrairement à l'utilisation d'une seule LDI. Dans leur système, une LLDI est obtenue à partir d'un LDC réduit à une seule image de référence. Ce stockage particulier a comme principal intérêt de permettre un rendu des points d'avant en arrière.

Avec le développement des techniques de splatting sur les cartes graphiques, l'intérêt de ces méthodes de stockage sous forme d'images devient moindre. Afin de limiter les transferts du CPU vers le GPU, il est important que les points soient stockés directement en mémoire vidéo, dans un ou plusieurs gros tampons appelés *vertex buffer objects* (VBO). Les points sont alors tracés en envoyant à la carte graphique la liste des indices des points à tracer, soit par le biais d'un tableau d'indices, soit par le biais d'un ensemble d'indices consécutifs défini par l'indice du premier élément et le nombre d'éléments à tracer. La deuxième version est bien sûr à préférer lorsque cela est possible puisque cela diminue à la fois les coûts de stockage (un noeud a juste à stocker les indices du premier et dernier élément) ainsi que la masse des données à transférer. La contrainte est que les points d'un noeud doivent être stockés de manière séquentielle dans le VBO associé. Une telle approche a par exemple été utilisée dans [RPZ02] et [GP03].

Ces concepts ne sont bien sûr pas limités aux octrees. Par exemple, dans [GM04] Gobetti et Marton ont utilisé un stockage par *kd*-tree associé à une architecture client-serveur permettant la visualisation progressive d'objets complexes à travers un réseau. La construction des versions simplifiées des objets est effectuée simplement par une sélection aléatoire des points de manière à ce que le nombre de points par noeud soit constant. La seule différence réelle qui existe entre l'utilisation d'un octree ou d'un *kd*-tree est que la résolution du nuage de points doit, entre chaque niveau, être multipliée par 2 pour le premier cas et par $\sqrt{2}$ pour le second cas.

Pour résumer, ce type de structures de données, octree ou *kd*-tree, est assez facile à construire et bien adapté à la réalisation de tests de visibilité efficaces. L'adéquation avec un rendu par splatting sur GPU est très bonne à condition que le nombre de points par noeud soit assez important (de l'ordre de mille [GM04]) afin d'envoyer les points au GPU par gros paquets et non point par point ce qui serait totalement inefficace.

La principale faiblesse de ces méthodes vient de la sélection des niveaux de détails qui n'est pas optimale. Celle-ci est réalisée pour chaque noeud en estimant la distance moyenne entre les points dans l'espace image. Ceci implique que la répartition des points au sein d'un noeud soit relativement uniforme. De plus, aucune information sur la courbure locale ou variation de la texture n'est prise en compte. En fait, cela est très difficile puisque pour être efficace la taille des noeuds doit être assez importante et donc chaque noeud représente un morceau de surface non homogène ne permettant pas ce type d'optimisation.

2.3.2.2 Les hiérarchies de sphères englobantes

Une seconde variété de structures de données hiérarchiques est la hiérarchie de sphères englobantes. Dans le projet QSplat [RL00], une telle hiérarchie est construite à partir du nuage de points où chaque point correspond à une feuille de l'arbre. Le niveau supérieur de l'arborescence

est obtenu en fusionnant les points les plus proches deux à deux. Les attributs du noeud père sont obtenus en moyennant les attributs de ses deux fils. Le rayon est choisi de manière à ce que la sphère définie par sa position et son rayon englobe les sphères associées de ses fils. Ce processus est répété jusqu'à obtenir une seule sphère correspondant à la racine de l'arbre.

Cette hiérarchie est donc caractérisée par une granularité très fine puisqu'un noeud correspond à un point. Les avantages par rapport aux méthodes précédentes sont donc une sélection très fine des points à tracer et la possibilité de stocker des nuages de points non uniformément répartis. Par contre, cette fine granularité implique un long et inefficace parcours dans la hiérarchie consommant beaucoup de ressources du CPU et ne permettant pas d'utiliser toute la puissance de calcul et de stockage des cartes graphiques modernes.

En 2003, Dachsbacher et al. [DVS03] ont proposé une généralisation de cette hiérarchie de sphères englobantes permettant entre autre d'ajuster la densité des points en fonction d'erreurs orthogonales (silhouette), tangentielles (courbure) et de texture. Cette généralisation est accompagnée d'une séquentialisation en largeur d'abord de l'arborescence qui peut alors être stockée en mémoire vidéo. Cette version séquentielle du QSplat est appelée *sequential point tree* (SPT). L'intérêt de cette séquentialisation est qu'il est maintenant possible de réaliser la sélection des niveaux de détails par le GPU lui-même. Au moment du rendu, le CPU sélectionne un préfixe grossier de points envoyé au GPU. Ce dernier est ensuite chargé de la sélection fine point par point. Cette sélection est réalisée par un *vertex shader* qui supprime les points superflus. L'intérêt majeur est illustré figure 2.15, l'utilisation du CPU est quasi nulle, ce dernier est donc libre pour réaliser d'autres tâches telles que l'animation de la scène par exemple.

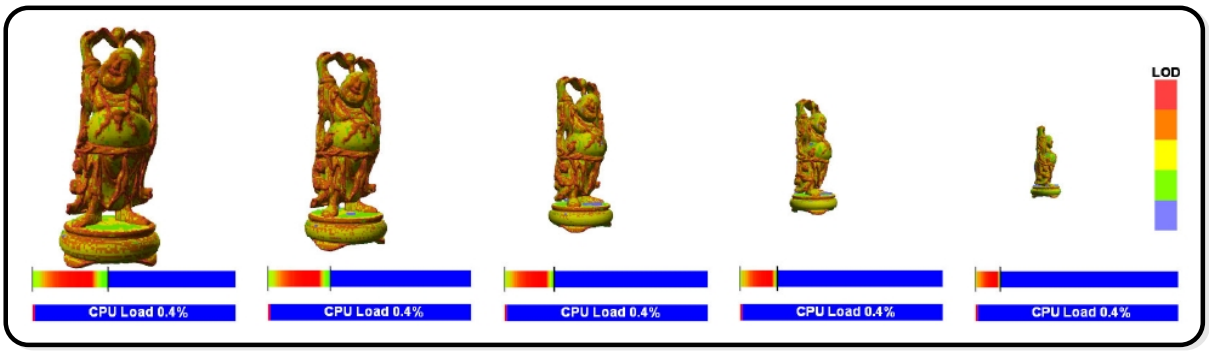


Fig. 2.15 – Rendu du multi-résolutions par les *sequential point trees* (source [DVS03]). La première ligne indique le préfixe sélectionné par le CPU tandis que la second ligne indique la charge du CPU.

En revanche, les inconvénients sont l'impossibilité de réaliser des tests de visibilité autrement que sur l'ensemble de l'objet tout entier, et le traitement inutile d'un grand nombre de points par le *vertex shader* de rendu. De plus, la construction d'une telle structure de données est beaucoup plus lourde au niveau des précalculs que la construction d'un octree par exemple.

2.3.3 Rendu hybride points/polygones

Comme nous l'avons vu à plusieurs reprises, à partir d'un certain degré de complexité une représentation par points est conceptuellement et pratiquement bien plus efficace qu'une représentation par polygones. Les polygones restent cependant bien plus adaptés lorsqu'il s'agit de

représenter des objets simples, c'est-à-dire présentant de larges zones plates efficacement représentées par quelques polygones texturés. La notion de "large" étant relative à la résolution de l'écran et au facteur d'échelle de la perspective, un objet complexe représenté par des millions de polygones peut localement apparaître simple en cas de fort zoom. Le concept de rendu hybride points/polygones prend alors tout son sens et le challenge est donc de choisir le plus rapidement et précisement possible quand et où utiliser l'une ou l'autre des représentations. Un triangle étant représenté par trois sommets, une possibilité est de remplacer un triangle par des points dès que celui-ci fait moins de trois pixels dans l'espace-image.

La plupart des systèmes de rendu hybride qui ont été proposés sont basés sur le même schéma : la représentation par points est efficacement stockée dans une structure de données multi-résolutions et hiérarchique (comme celles que nous venons de voir) où chaque feuille de l'arborescence contient, en plus des points, la représentation polygonale.

Ainsi, lors du parcours récursif de la hiérarchie, si la densité du noeud courant n'est pas suffisante et que le noeud est une feuille alors les points du noeud sont avantageusement remplacés par les polygones intersectant le volume englobant associé à ce noeud.

Partageant ce principe, nous pouvons citer le système POP de Chen et Nguyen [CN01] et les *sequential point trees* [DVS03] tous deux basés sur une hiérarchie QSplat, mais aussi l'arborescence de LLDI de Coconu et Hege [CH02].

Dans [CAZ01] Cohen et al. proposent une methode un peu plus complexe dans laquelle la structure hiérarchique contient également une simplificication de la représentation polygonale.

2.4 Conclusion

Comme nous l'avons remarqué en conclusion de la section 2.2, parmis toutes les méthodes de rendu de nuages de points que nous avons présenté, les méthodes par splatting offrent sans aucun doute le meilleur compromis flexibilité/qualité/efficacité que nous nous sommes fixé. Les composantes principales permettant d'atteindre ce compromis sont :

- pas de précalcul (flexibilité)
- anti-aliasing de la texture (qualité)
- éclairage par pixel (qualité)
- implantable sur les cartes graphiques (efficacité)

Cependant, aucune des méthodes que nous avons présentées n'allient toutes ces composantes à la fois. Aussi, dans le chapitre 3 nous montrerons comment implanter efficacement sur GPU un algorithme d'EWA splatting avec interpolation des attributs des points, *deferred shading*, support des surfaces semi-transparentes et filtrage des transitions lors d'un rendu hybride points/polygones.

Malgré un très bon anti-aliasing et un éclairage par pixel, nous avons également fait remarquer que la qualité d'un rendu par splatting se dégrade rapidement dès que la densité des points n'est pas suffisante. Afin de palier à ce problème majeur, nous présenterons dans le chapitre 5 une méthode de raffinement dynamique de nuages de points permettant de maintenir une densité des points suffisantes tout au long du processus de visualisation, garantissant ainsi une visualisation de haute qualité par splatting.

Bien que nous proposons une implantation sur GPU d'un rendu par splatting de qualité pouvant être qualifié d'"efficace", les coûts de calcul requit par les opérations de filtrage et d'interpolation sont encore trop importants pour permettre un rendu temps-réel de scènes complexes.

Nous pouvons remarquer que la plupart des systèmes basés points de rendu de scènes complexes se contentent d'un rendu des points de très faible qualité. Afin de rendre compatible les trois adjectifs qualité, complexité et temps-réel, nous proposons au chapitre 4 une réorganisation des passes des splatting permettant de rendre les coûts d'un splatting de qualité indépendant de la complexité de la scène.

Chapitre 3

Splatting de haute qualité accéléré par GPU

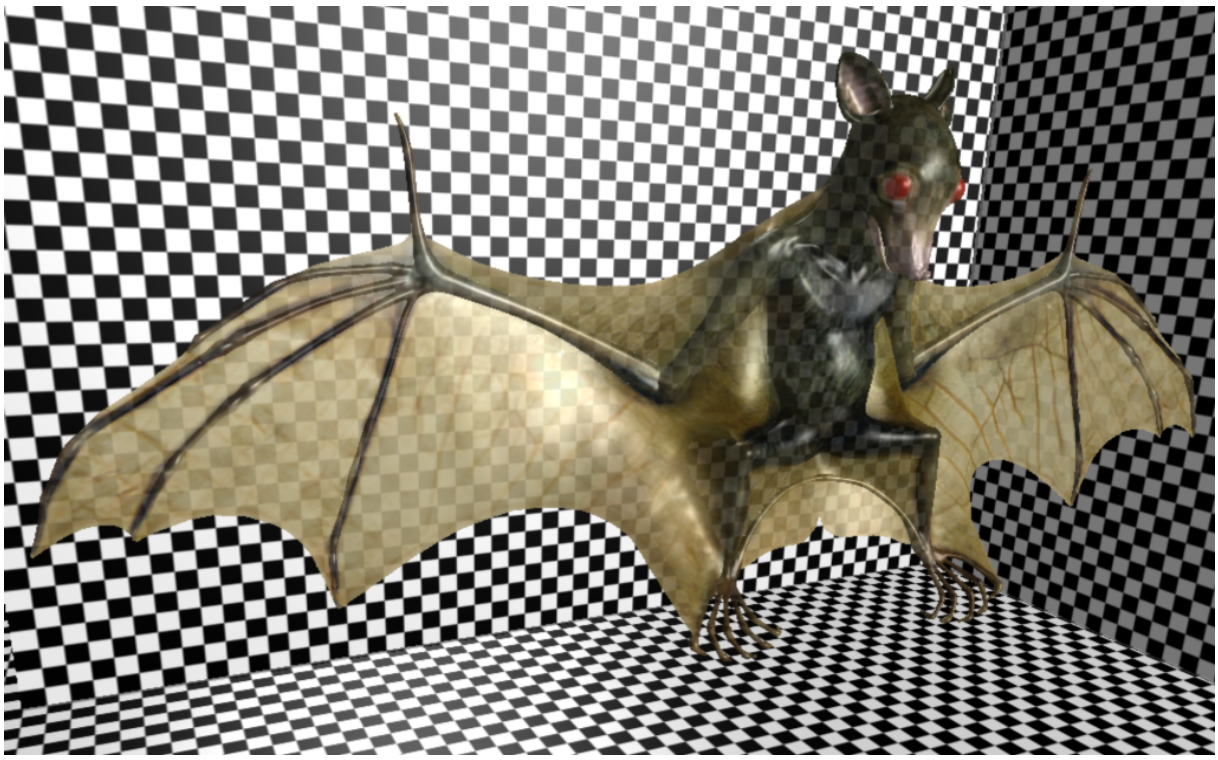


FIG. 3.1 – Une scène entièrement modélisée par splats et rendue en temps-réel par notre algorithme de splatting. La chauve-souris est un modèle semi-transparent composé de 480 000 points.

Introduction

Lorsque l'interactivité est un critère important, les approches les plus intéressantes pour la visualisation de nuages de points sont les méthodes de splatting. Ce sont effectivement celles-ci

qui offrent, de loin, le meilleur compromis entre flexibilité, rapidité et qualité. Cependant, comme nous l'avons déjà remarqué, aucune des méthodes de splatting existantes n'allient réellement ces trois critères à la fois.

Dans ce chapitre nous allons présenter une implantation efficace d'un algorithme de rendu par splatting de haute qualité supportant les surfaces semi-transparentes et le filtrage des transitions lors d'un rendu hypride points/polygones.

D'après ce que nous avons vue en état de l'art, section 2.2.2.3, les points clés pour obtenir un rendu de qualité par splatting sont :

- Accumulation de filtres Gaussiens (ou similaires) pour l'interpolation et filtrage des attributs.
- Utilisation d'un filtre passe-bas pour gérer les problèmes d'aliasing en cas de réduction.
- Utilisation d'une approche par *deferred shading* permettant à la fois des calculs d'éclairage par pixel et de ne réaliser ceux-ci que pour les pixels visibles.

Faute de pouvoir développer une carte graphique dédiée, notre algorithme de splatting doit pouvoir être implanté à 100% sur un GPU[4] moderne afin de tirer profit de leurs performances accrues, mais aussi de libérer le CPU pour effectuer les autres tâches (animation, simulation, etc.) en parallèle.

Une approche par splatting offre de facto un bon niveau de flexibilité puisqu'elle ne nécessite pas forcément de précalcul d'une structure accélératrice par exemple. On peut, en revanche, s'interroger sur la primitive de base. Les méthodes par splatting ont en effet été adaptées pour le rendu de splats isotropes, de splats elliptiques et même de points différentiels (splats elliptiques + information de courbure). Les splats isotropes étant les plus simples, il est clair que ce sont eux qui offrent le plus de flexibilité puisque très faciles à manipuler. D'un autre côté, en cas de déformation par exemple, la surface peut subir des étirements anisotropes transformant des splats isotropes en splats elliptiques. Tout en privilégiant les splats isotropes, il paraît donc important de laisser la possibilité de tracer des splats elliptiques.

Ces dernières années ont vu l'émergence de cartes graphiques dites programmables. Cette programmabilité est disponible à différents niveaux. Au niveau des sommets, les *vertex shaders* permettent à l'utilisateur (programmeur) de personnaliser les transformations des attributs des sommets ou même de calculer ces attributs à la volée. Les *fragment shaders* permettent quant à eux une personnalisation des calculs d'ombrage à l'échelle du pixel. Le lien entre *vertex shader* et *fragment shader* est réalisé par l'étape de rastérisation du triangle interpolant les attributs issus du *vertex shader* pour les fournir au *fragment shader* qui est exécuté pour chaque fragment du triangle.

Ce modèle de programmation du pipe-line de rendu offre de nombreuses possibilités aux utilisateurs. Il paraît donc important que notre pipe-line de splatting offrent ces mêmes possibilités de programmation.

Le *pipe-line* de splatting et son implantation que nous proposons aujourd'hui a été développé et mis à jour tout au long de ma thèse afin de suivre les évolutions des cartes graphiques. Une version préliminaire, développée début 2003 a été publiée dans [GP03]. Notons que, en parallèle à ces travaux, Botsch et al. ont publié milieu 2005 une implantation du splatting sur GPU similaire à la notre [BHZK05] dont nous discuterons des différences dans la section 3.4.3.

[4]GPU signifie *Graphic Processor Unit* et désigne le processeur de la carte graphique [SA04].

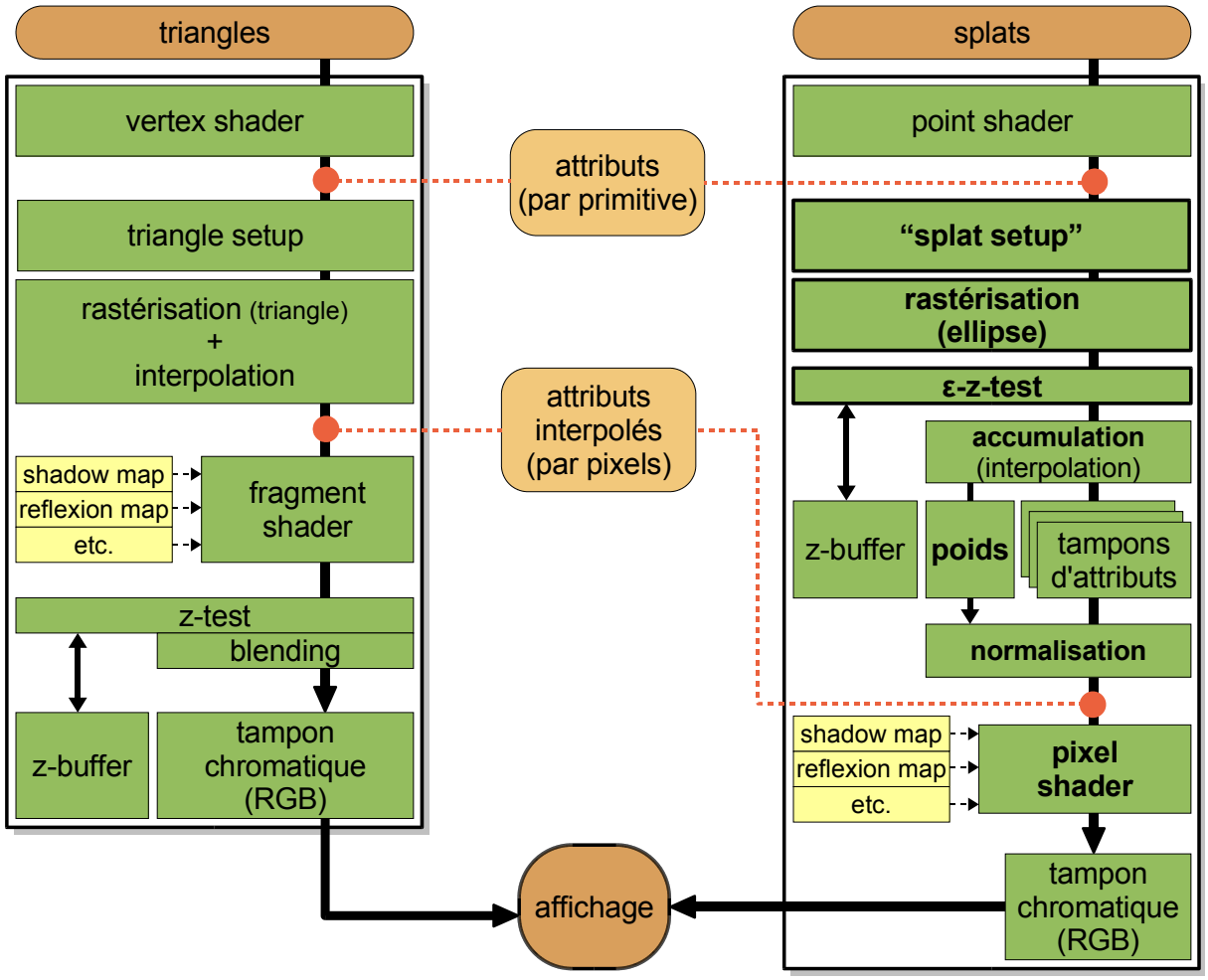


FIG. 3.2 – À gauche schéma d'un GPU actuel. À droite design de notre *pipe-line* de splatting.

Ce chapitre est organisé de la manière suivante :

- Section 3.1 nous présenterons les concepts de notre *pipe-line* et un aperçu de son implantation sur GPU.
- Dans la section suivante 3.2, nous détaillerons notre implantation efficace du splatting de surface sur GPU.
- Section 3.3, nous montrerons comment réaliser un rendu hybride points/polygones avec filtrage des transitions puis comment notre pipe-line peut être adapté au rendu des surfaces semi-transparentes.

3.1 Aperçu du *pipe-line* de splatting programmable

3.1.1 Conception

Conceptuellement notre pipe-line de rendu de points par splatting se présente de la manière suivante (voir figure 3.2).

Étage 1 D'une façon similaire aux cartes graphiques actuelles, un ensemble de splats, stockés en mémoire vidéo, est envoyé à un *point shader* qui exécute pour chaque primitive un petit programme défini par l'utilisateur. Par défaut, ce *point shader* prend en entrée un splat standard (position, normale, couleur, rayon) et se contente d'appliquer les transformations de modélisation de vue sur la position et la normale. L'intérêt d'avoir cet étage de transformation programmable est de permettre à l'utilisateur de définir ses propres transformations de modélisation, de calculer certains paramètres du point dynamiquement, etc. Un *point shader* est donc quasiment identique aux *vertex shader* classiques à quelques exceptions près : premièrement, la transformation de projection n'est ici pas programmable ; deuxièmement, le shader doit fournir en sortie :

- la position du point exprimée dans le repère de la caméra (avant la projection perspective)
- la normale du point exprimée dans le repère de la caméra.
- le rayon du splat dans l'espace scène.
- un certain nombre d'attributs à interpoler entre les points (e.g. la couleur, ...)

Étage 2 À partir des trois premiers attributs fournis par le *point shader* (position, normale, rayon), le "splat setup" effectue la projection perspective du splat dans l'espace image, ainsi que le fenêtrage de la primitive. Le splat projeté est ensuite rastérisé, c'est-à-dire converti en fragments. Chaque fragment correspond à un pixel de l'écran et contient les attributs à interpoler issus du vertex shader : la normale du splat, sa profondeur et son poids (issu de l'évaluation du noyau de ré-échantillonnage).

La profondeur du fragment z_f est ensuite comparée à ϵ près à la valeur z_d présente dans le z-buffer (f signifie fragment et d destination) :

---

si $z_f < z_d - \epsilon$ **alors**
    // mise à jour
    z_d = z_f ; *// mise à jour du z-buffer*
    w_d = w_f ; *// mise à jour du poids*
    // mise à jour des attributs
    pour chaque attribut A associé au fragment **faire**
        A_d = w_f * A_f ;

sinon si $z_f < z_d + \epsilon$ **alors**
    // accumulation
    w_d += w_f ;
    pour chaque attribut A associé au fragment **faire**
        A_d += w_f * A_f

Étage 3 À la fin du rendu de tous les points, chaque pixel des tampons de destination est normalisé et envoyé au *pixel shader*. Le *pixel shader* est un petit programme défini par l'utilisateur. C'est ici que sont traités les attributs variables qui viennent d'être interpolés par splatting. Différents modèles d'éclairage peuvent être implémentés ici, qu'ils soient ou non photo-réalistes. Différentes unités de texture doivent être disponibles afin de réaliser des réflexions ou des ombres portées via la technique des *shadow maps* par exemple. La sortie d'un *pixel shader* est uniquement la couleur RGB du pixel qui sera affiché à l'écran. En fait, la seule vraie différence

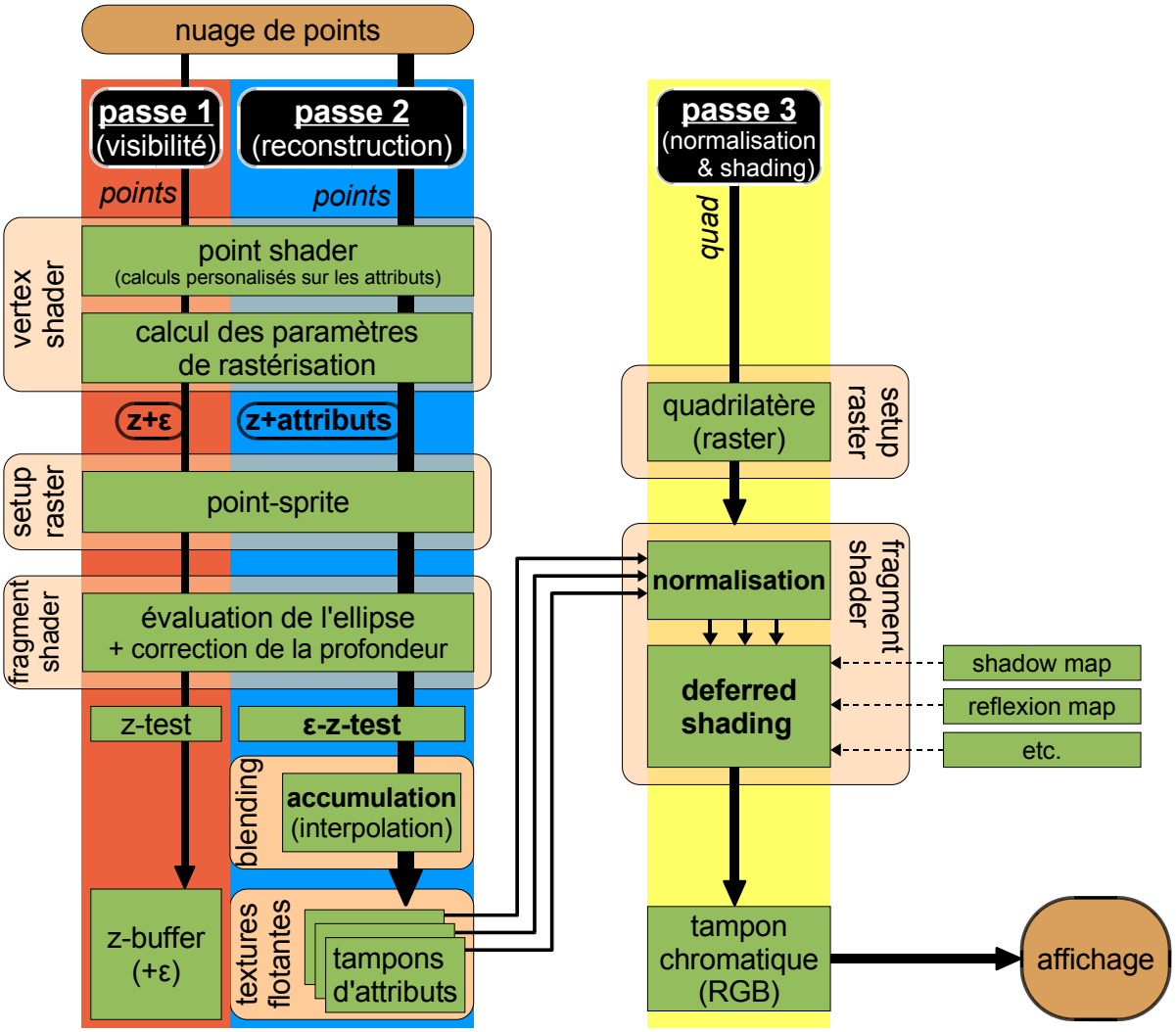


FIG. 3.3 – Apercu de l'algorithme de splatting de surface multi-passes.

conceptuelle qui existe avec les unités de *fragment shader* des GPU actuels est que le shader n'est exécuté que pour les pixels réellement visibles.

3.1.2 Les clés d'une implantation sur les GPU actuels

Dans cette section, nous allons voir comment le pipe-line que nous venons de décrire peut être implanté sur les GPU actuels, ou inversement comment les GPU actuels permettent de simuler notre propre pipe-line. En effet, toutes les fonctionnalités requises ne sont pas encore disponibles et devront pour certaines être simulées. Les deux principaux manques concernent le module de rastérisation d'ellipse et le test de profondeur avec tolérance. Une solution maintenant classique au manque d'un z-buffer flou est de précalculer le z-buffer via une passe de rendu supplémentaire [RL00, RPZ02]. Pour la rastérisation d'ellipse, nous avions déjà montré qu'il était possible d'utiliser la primitive point et les *fragment shaders* disponibles sur la génération précédente de GPU [GP03].

Avant l'arrivée de la génération NV40 des GPU de NVIDIA, il n'était pas possible d'interpoler par splatting plus que les trois composantes de la couleur. Une approche par *deferred shading* aurait alors nécessité au minimum une passe supplémentaire pour l'interpolation des normales. De plus, le tampon de destination utilisé pour accumuler les contributions était limité à 8 bits par composante, ce qui pouvait entraîner soit une saturation des valeurs soit une dégradation de la qualité à cause de la trop faible précision. Les derniers GPU permettent maintenant de tracer dans plusieurs tampons de destination à la fois. En fait, jusqu'à quatre couleurs RGBA sont accessibles en une seule passe de rendu. De plus, une précision flottante est disponible d'un bout à l'autre du pipe-line, c'est à dire au niveau des *vertex* et *fragment shaders*, des textures, du blending et des tampons de destination. Ces deux nouvelles fonctionnalités sont exactement ce dont nous avons besoin pour simuler nos tampons d'attributs. En réservant une composante des tampons de destination pour la somme des poids, nous avons un maximum de 15 autres scalaires en précision flottante disponibles pour accumuler les attributs des splats, ce qui en pratique est largement suffisant.

Pour résumer, la figure 3.3 illustre comment notre propre pipe-line de splatting est adapté pour être simulé sur un GPU moderne :

- Les splats sont envoyés à la carte graphique sous la forme d'un seul sommet attribué en utilisant la primitive `point`.
- Le vertex shader du GPU est utilisé à la fois pour exécuter notre propre *point shader* et pour émuler en partie l'étage de "splat setup".
- La seconde partie du "splat setup", i.e. la division perspective et fenêtrage, est réalisée par les fonctions dédiés du GPU.
- La rastérisation du splat est réalisée par la rastérisation d'une primitive point (carré de pixels parallèle aux axes et au plan image) et par un *fragment shader* évaluant pour chaque fragment la profondeur et le poids (valeur du filtre de ré-échantillonnage).
- Le test de profondeur avec tolérance est quant à lui émulé par un rendu en deux passes. La première se contente de précalculer le z-buffer tandis que la seconde accumule les contributions.
- L'accumulation des contributions est facilement réalisée par l'étage de blending du GPU configuré en mode additif. Pour les tampons d'attributs et de poids nous utilisons de une à quatre textures flottantes de quatre composantes chacune.
- Finalement, les pixels obtenus sont traités par un unique *fragment shader* réalisant la normalisation des attributs et exécutant notre propre *pixel shader*.

3.2 Implantation du splatting sur GPU par *deferred shading*

3.2.1 Rastérisation d'un splat

La rastérisation d'un splat, ou plutôt du filtre de ré-échantillonnage, comprend bien sûr l'étage de rastérisation mais aussi l'étage de "setup". Afin de garantir les meilleures performances, il est important que les calculs induits par ces deux composantes soient les plus simples possibles, d'autant plus qu'en pratique ils vont être simulés par des *vertex* et *fragment shaders*. Bien entendu, la qualité de la reconstruction ne doit pas être oubliée. Toute la difficulté est donc de trouver le meilleur compromis.

Comme nous l'avons vu dans l'état de l'art, une première possibilité serait d'utiliser une technique de lancer de rayon tel que celle proposée par Botsch et Kobbelt [BK04] qui a l'avantage

d'être correcte au niveau de la projection perspective et ne requiert qu'un très simple "setup". En contrepartie, les calculs d'intersection sont un peu plus coûteux qu'avec d'autres approches et un matériel dédié ne pourrait tirer profit d'une rastérisation rapide par incréments. De plus, il n'est pas possible d'utiliser un filtrage EWA.

Une approche utilisant une approximation locale affine de la projection perspective semble alors plus pertinente puisque cela permet à la fois une rastérisation vraiment efficace, mais aussi l'utilisation du filtrage anisotropique de l'EWA splatting. Reste à déterminer comment calculer le plus rapidement possible les paramètres définissant la forme du noyau de ré-échantillonnage. Comme nous l'avons vu, deux possibilités sont envisageables pour approcher la projection perspective : avoir une projection exacte du centre du splat [ZPvBG01] ou bien avoir une projection exacte du contour externe [ZRB+04]. Bien que la deuxième solution soit la seule garantissant une reconstruction sans trous, nous avons opté pour la première qui, d'une part, a l'avantage d'être moins gourmande en calculs et qui, d'autre part, se montre suffisante en pratique grâce au recouvrement des splats.

Finalement, nous avons choisi de reprendre la méthode originelle de l'EWA splatting [ZPvBG01] dont nous présentons ici une nouvelle implantation sur GPU. Les méthodes qui ont été proposées dans [ZPvBG01, RPZ02, GP03] pour calculer les paramètres du filtre de ré-échantillonnage ρ (équation 2.23) requièrent de nombreuses opérations. Avec ces approches, l'étage de "splat setup" devient finalement tout aussi coûteux que le "setup" pour la rastérisation d'un triangle.

Nous proposons donc ici une méthode alternative s'appuyant sur une décomposition en valeurs propres et vecteurs propres qui permet de calculer directement, à partir de la normale du splat, les paramètres requis par l'étage de rastérisation. Nous proposons également quelques approximations permettant de réduire quelque peu les temps de calculs sans pour autant perdre en qualité.

3.2.1.1 Évaluation de la forme du splat

Reprenons le formalisme de l'EWA splatting introduit section 2.2.2.3. Rappelons qu'en approximant la projection perspective par une transformation affine et en prenant pour le noyau de reconstruction ϕ_i une Gaussienne $g_{\mathbf{R}_i}$ de variance $\mathbf{R}_i$ alors le noyau de reconstruction dans l'espace écran ϕ'_i est aussi une Gaussienne. Appelons $\mathbf{R}'_i$ sa matrice de covariance, nous pouvons réécrire l'équation 2.22 de ϕ'_i de la manière suivante :

$$\phi'_i(\mathbf{x}) = |\mathbf{J}_i| g_{\mathbf{R}'_i}(\mathbf{x}) = \frac{|\mathbf{J}_i|}{2\pi|\mathbf{R}'_i|^{\frac{1}{2}}} e^{-\frac{1}{2}\mathcal{D}_{\mathbf{R}'_i}(\mathbf{x})} \tag{3.1}$$

où $\mathcal{D}_{\mathbf{R}'_i}(\mathbf{x}) = \mathbf{x}^T {\mathbf{R}'_i}^{-1} \mathbf{x}$ définit un champ de distance elliptique caractérisant la forme de la Gaussienne. $\mathcal{D}_{\mathbf{R}'_i}$ est parfois également appelée distance de Mahalanobis. Par définition, la matrice de covariance $\mathbf{R}'_i$ est une matrice symétrique et peut donc être décomposée en valeurs propres et vecteurs propres. Les vecteurs propres déterminent les axes principaux de l'ellipse alors que les valeurs propres associées représentent la variance le long de ces axes. Par exemple, si nous nous intéressons à l'ellipse d'iso-valeur 1 alors les valeurs propres correspondent exactement aux carrés des rayons de l'ellipse. Soit $\mathbf{V}$ la matrice orthogonale des vecteurs propres et $\mathbf{\Lambda} = \begin{bmatrix} \lambda_0 & 0 \\ 0 & \lambda_1 \end{bmatrix}$ la matrice diagonale des valeurs propres associées. Nous avons alors :

$$\mathbf{R}'_i = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} \tag{3.2}$$

D'après l'équation 2.23, la forme du filtre de ré-échantillonnage ρ_i est caractérisée par le champ elliptique de variance $\mathbf{R}'_i + \mathbf{H}$ où la variance $\mathbf{H}$ de filtre passe-bas est en fait égale à la matrice identité $\mathbf{I}$. Nous pouvons alors écrire :

$$\mathbf{R}'_i + \mathbf{H} \quad = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} + \mathbf{I} = \mathbf{V}(\mathbf{\Lambda} + \mathbf{I})\mathbf{V}^{-1} \tag{3.3}$$

Ce résultat est intéressant car il signifie que l'application du filtre passe-bas par convolution correspond à une mise à l'échelle (non uniforme et non linéaire) des rayons de l'ellipse. Lors de l'évaluation, ou rastérisation, du noyau de ré-échantillonnage, nous avons besoin de l'inverse de la matrice de covariance précédente. Grâce aux propriétés des matrices orthogonales et diagonales, nous avons :

$$(\mathbf{R}'_i + \mathbf{H})^{-1} = \mathbf{V}(\mathbf{\Lambda} + \mathbf{I})^{-1}\mathbf{V}^{-1} \tag{3.4}$$

$$= \mathbf{V} \begin{bmatrix} \frac{1}{\lambda_0+1} & 0 \\ 0 & \frac{1}{\lambda_1+1} \end{bmatrix} \mathbf{V} \tag{3.5}$$

L'intérêt de ce jeu d'écriture est que nous pouvons maintenant calculer la matrice dont nous avons besoin très rapidement. En effet, l'axe $\mathbf{a}_1$ associé à la plus petite valeur propre λ_1 est porté par la projection de la normale $\mathbf{n}_i$ du splat sur l'écran. En considérant que $\mathbf{n}_i$ est exprimée dans le repère de la caméra, nous avons :

$$\mathbf{a}_1 = \frac{(\mathbf{n}_{i,x}, \mathbf{n}_{i,y})^T}{\|(\mathbf{n}_{i,x}, \mathbf{n}_{i,y})^T\|} \tag{3.6}$$

L'autre axe $\mathbf{a}_0$, correspondant à la plus grande valeur propre, est simplement obtenu par une rotation de $-\frac{\pi}{2}$:

$$\mathbf{a}_0 = (\mathbf{a}_{1,y}, -\mathbf{a}_{1,x})^T \tag{3.7}$$

Soit η le facteur d'échelle associé à la pyramide de vision. η peut être calculé à partir des angles d'ouverture fov_l et fov_h donnés respectivement selon l'axe horizontal et vertical et les dimensions de la fenêtre f_l, f_h en pixels :

$$\eta = max\left(\frac{f_l}{2tan(\frac{fov_l}{2})}, \frac{f_h}{2tan(\frac{fov_h}{2})}\right) \tag{3.8}$$

Le facteur d'échelle local η_i pour le point de coordonnées $\mathbf{p}_i$ exprimées dans le repère de la caméra est alors : $\eta_i = \frac{\eta}{\mathbf{p}_{i,z}}$. La plus grande valeur propre λ_0 correspond au carré du rayon de la sphère englobante projeté sur l'écran, i.e. :

$$\lambda_0 = (\eta_i r_i)^2 \tag{3.9}$$

Le ratio α entre les rayons de l'ellipse est donné par le cosinus de l'angle entre la normale $\mathbf{n}_i$ du splat et la direction de visée $\mathbf{v}_i$, d'où :

$$\alpha^2 = \frac{\lambda_1}{\lambda_0} = \mathbf{n}_i \cdot {\mathbf{v}_i}^2 \tag{3.10}$$

Finalement, le filtre de ré-échantillonnage doit être normalisé par son aire dans l'espace source. Le résultat de l'équation 3.3 montre que l'application du filtre passe-bas par convolution correspond à une mise à l'échelle des rayons (ou des variances) de l'ellipse. Comme nous utilisons

localement une approximation affine de la transformation de projection, et que nous considérons des splats isotropes, les rayons (ou variances) du filtre de ré-échantillonnage dans l'espace source subissent la même mise à l'échelle. D'où le facteur de normalisation Δ_i suivant :

$$\Delta_i = \frac{1}{2\pi} \left| \begin{array}{cc} r_i^2 \frac{\lambda_0+1}{\lambda_0} & 0 \\ 0 & r_i^2 \frac{\lambda_1+1}{\lambda_1} \end{array} \right|^{-\frac{1}{2}} \tag{3.11}$$

$$= \frac{1}{\sqrt{r_i^4 \frac{\lambda_0+1}{\lambda_0} \frac{\lambda_1+1}{\lambda_1}}} \tag{3.12}$$

3.2.1.2 Implantation sur le GPU

Nous allons maintenant voir rapidement comment implanter ce rendu de splat sur une carte graphique moderne.

Ces splats sont tracés via une primitive spéciale des cartes graphiques appelée *point sprite*. Tout comme la primitive point standard, celle-ci trace à l'écran un carré aligné aux axes et parallèle à l'écran (profondeur constante). Les *point sprite* ont en plus deux particularités importantes pour notre cas. La première est au niveau de la précision puisque ni la taille ni les coordonnées du centre du carré généré ne sont arrondies au pixel le plus proche. La seconde particularité est qu'une paramétrisation du point est disponible via les coordonnées de texture. Cette paramétrisation, accessible à partir d'un *fragment shader*, va de $(0,0)$ pour le coin inférieur gauche à $(1,1)$ pour le coin opposé. Une simple translation par $(-\frac{1}{2}, -\frac{1}{2})$ permet d'obtenir une paramétrisation de $(-\frac{1}{2}, -\frac{1}{2})$ à $(\frac{1}{2}, \frac{1}{2})$ centrée sur la position du centre du splat. Soit $\mathbf{y}$ les coordonnées d'un fragment dans cette paramétrisation :

$$\mathbf{x} = \mathbf{x}_i + 2 * pointsize * \mathbf{y} \tag{3.13}$$

L'évaluation de $\rho_i(\mathbf{x})$ pour ce fragment est alors équivalent à :

$$\rho_i(\mathbf{x}) = \Delta_i e^{-\frac{1}{2}\mathbf{y}^T \mathbf{Q} \mathbf{y}} \tag{3.14}$$

Comme $pointsize = \sqrt{\lambda_1 + 1}$, la matrice $\mathbf{Q}$ est égale à :

$$\mathbf{Q} = 4\mathbf{V} \begin{bmatrix} \frac{\lambda_0+1}{\lambda_1+1} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{V}^T \tag{3.15}$$

Pour résumer, à partir de la position $\mathbf{p}_i$ et normale $\mathbf{n}_i$ du point tous deux exprimés dans le repère de la caméra, et du rayon r_i du splat, le *vertex shader* simulant la première partie du "splat setup" réalise dans l'ordre :

- Évaluer le ratio α des rayons du noyau de reconstruction projeté : $\alpha = \mathbf{n}_i \cdot \frac{-\mathbf{p}_i}{\|\mathbf{p}_i\|}$ (4 opérations). Notons au passage que si le résultat de ce produit scalaire est négatif, alors cela signifie que le splat est en face arrière. Ainsi, si l'objet est un objet solide, alors ce splat ne peut être visible et peut donc être ignoré. Pour cela, le point est envoyé à l'infini en mettant sa coordonnée homogène w à 0.
- Calculer $\eta_i = \frac{\eta}{\mathbf{p}_{i,z}}$ (η est passé au *vertex shader* par une variable *uniform*, 2 opérations).
- En déduire le ratio $\beta = \frac{\lambda_0+1}{\lambda_1+1}$ des valeurs propres du noyau de ré-échantillonnage : $\beta = \frac{(\eta_i r_i)^2+1}{(\alpha\eta_i r_i)^2+1}$ (4 opérations).
- Calculer l'axe $\mathbf{a} = \mathbf{a}_1$ (équation 3.6, 3 opérations).

- Calculer la matrice $\mathbf{Q} =$ (équation 3.15, 3 opérations) :

$$2 \begin{bmatrix} \beta \mathbf{a}_x^2 + \mathbf{a}_y^2 & \beta \mathbf{a}_x \mathbf{a}_y - \mathbf{a}_x \mathbf{a}_y \\ \beta \mathbf{a}_x \mathbf{a}_y - \mathbf{a}_x \mathbf{a}_y & \mathbf{a}_x^2 + \beta \mathbf{a}_y^2 \end{bmatrix} \tag{3.16}$$

- Évaluer la taille de la primitive *point sprite* : $2\sqrt{(\eta_i r_i)^2 + 1}$ (2 opérations)
- À partir des calculs précédents, le coefficient de normalisation Δ_i (équation 3.11) peut être calculé en seulement 4 opérations : $\Delta_i = \frac{\alpha\lambda_0}{r_i^4 \beta(\lambda_1+1)}$.

Soit au total 22 opérations vectorielles ou élémentaires disponibles à partir d'un langage assembleur pour GPU telle que l'assembleur ARB OpenGL [Ba02, La02].

Le fragment programme doit pour chaque fragment généré corriger sa profondeur et évaluer $\rho_i(\mathbf{x})$:

- Évaluer l'exposant de l'exponentielle : $\mathbf{y}^T Q \mathbf{y}$ (3 opérations).
- Le résultat est utilisé pour accéder à la texture stockant les valeurs de l'exponentielle. (1 accès texture)
- Le poids du fragment est alors obtenu en multipliant la valeur de l'exponentielle par le facteur de normalisation Δ_i et est stocké dans les composantes alpha (ou w) des attributs à interpoler (couleur, normale, ...)

3.2.1.3 Correction de la profondeur

(a) (b) (c)

FIG. 3.4 – Illustration de l'intêret d'une correction de la profondeur.
(a) Splatting sans correction de la profondeur ($\epsilon = \frac{1}{2} r_i$). (b) Splatting sans correction de la profondeur ($\epsilon = \frac{3}{2} r_i$). (c) Splatting avec correction de la profondeur ($\epsilon = \frac{1}{2} r_i$).

Comme les splats sont tracés via la primitive point qui a une profondeur constante, il est nécessaire de réaliser une correction de la profondeur de chaque pixel. En effet, sans cette correction, garantir l'accumulation de toutes les contributions visibles implique l'utilisation d'un ϵ important dans l'algorithme de z-buffer flou (figure 3.4-a). Cependant, plus la valeur de ϵ est grande et plus le risque de mélanger des splats non visibles est important, comme le montre la figure 3.4-b. Effectuer une correction de la profondeur par fragment permet donc d'utiliser une petite valeur pour ϵ et donc de limiter les artefacts, figure 3.4-c. Le choix pour la valeur de ϵ dépend de l'espacement local entre les points et doit donc être ajustée pour chaque splat. En pratique, nous avons trouvé que prendre $\epsilon = \frac{1}{2} r_i$ est un bon compromis (r_i dénote le rayon du point $\mathbf{p}_i$).

Pour cette correction, nous effectuons un calcul exact de la profondeur z_f^c du fragment exprimée dans le repère de la caméra qui est obtenue en calculant l'intersection entre le rayon issue du fragment et le plan tangent du splat :

$$z_f^c = \frac{\mathbf{p}_i \cdot \mathbf{n}_i}{\mathbf{x} \cdot \mathbf{n}_i} \tag{3.17}$$

où $\mathbf{x}$ sont les coordonnées du fragment dans le repère de la caméra. En fait, les valeurs de profondeur contenues dans le z-buffer ne sont pas exprimimées dans le repère de la caméra, mais en coordonnées écran via la formule de normalisation suivante :

$$z_f = \frac{z_{max} + z_{min}}{z_{max} - z_{min}} + \frac{2 z_{max} z_{min}}{z_{max} - z_{min}} \frac{1}{z_f^c} \tag{3.18}$$

où z_{min} zt z_{max} définissent les plans *near* et *far* du volume de vision. Après développement et simplifications, le calcul de z_f est effectué pour chaque fragment très efficacement par un simple produit scalaire.

3.2.1.4 Approximation du filtrage EWA

Pour obtenir un maximum de performances, il est encore possible d'effectuer quelques approximations et simplifications. La première est de négliger le facteur de normalisation Δ_i. En effet, à cause de la troncature du support des filtres de reconstruction, seules les contributions des points voisins sont mélangées entre elles. En considérant que deux points voisins ont des rayons et orientations assez proches, alors les facteurs de normalisations associés sont également très proches et peuvent donc être tout simplement ignorés grâce à la normalisation finale par la somme des poids. En pratique, cela permet de simplifier à la fois le *vertex shader* (-4 opérations) et *fragment shader* (-1 opération), soit un gain d'environ 4-6%.

Une seconde optimisation est d'approcher la forme du filtre de ré-échantillonnage obtenu par convolution. L'équation 3.3 nous apprend que l'application du filtre passe-bas correspond à l'ajout de 1 aux variances du filtre de reconstruction projeté. En d'autres termes, cela revient à une mise à l'échelle des rayons par la formule suivante : $r' = \sqrt{r^2 + 1}$. Une approximation possible est de prendre $r' = max(r, 1)$ (voir figure 3.5). En effet, le filtre passe-bas est particulièrement utile pour lisser les hautes fréquences du signal reconstruit en cas de réduction, c'est-à-dire lorsque la taille des noyaux de reconstruction est inférieure au pixel. Cette formule garantie simplement une taille des filtres de ré-échantillonnage d'au moins deux pixels indispensable au filtrage. D'un point de vue pratique, cette approximation permet simplement une réduction de deux ou trois opérations ce qui n'est pas suffisant pour être vraiment significatif. Par contre, en cas d'agrandissement le filtre passe-bas n'est plus vraiment utile. Avec notre approximation, la taille des noyaux de reconstruction reste inchangée, ce qui permet de ne pas augmenter inutilement le nombre de pixels à rastériser et à traiter. Les gains observés sont d'environ 10 %.

3.2.2 Implantation de l'algorithme multi-passes

3.2.2.1 Passe de visibilité

Rappelons que l'objectif de cette première passe est de précalculer un z-buffer sans trous, de manière à ce que seules les contributions visibles soient accumulées lors de la seconde passe de reconstruction. Pour cela, les splats sont tracés sous la forme de disques opaques décalés de ϵ dans la direction de visée. Ce décalage est réalisé par le vertex shader. La forme de la projection

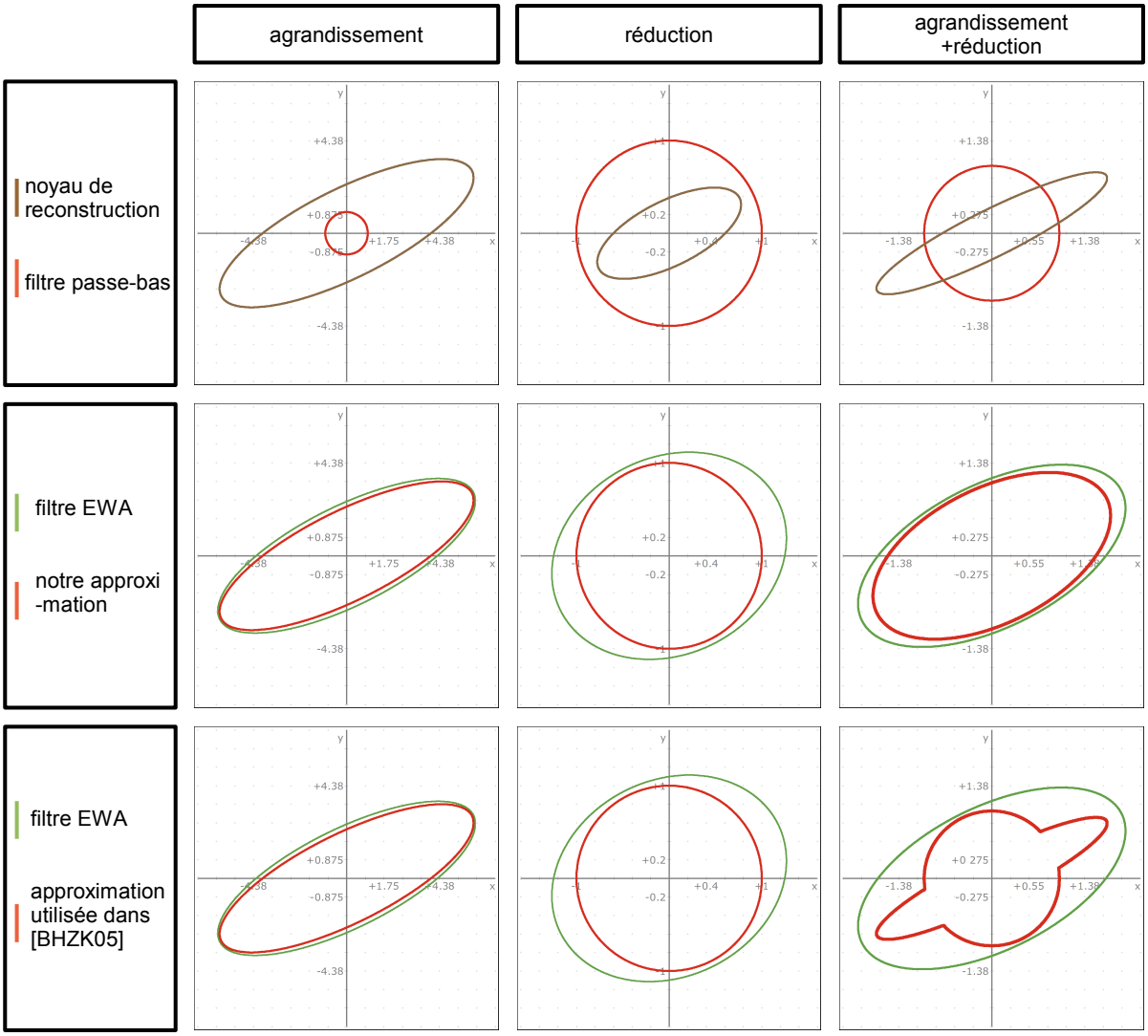


FIG. 3.5 – Différentes approximations du filtrage EWA.

du splat sur l'écran est calculée exactement de la même manière qu'expliqué dans la section précédente, sauf qu'ici nous n'avons pas besoin d'appliquer le filtre passe-bas ni de calculer et appliquer le facteur de normalisation. Au niveau du fragment shader, nous avons donc juste à évaluer l'exposant $d = \mathbf{y}^T Q' \mathbf{y}$ et à le comparer à 1 pour savoir si le fragment courant est à l'intérieur du splat ou non. En pratique, afin d'alléger le fragment shader, cette comparaison est réalisée par le test alpha. La valeur de d est donc stockée dans le canal alpha. Comme la valeur de la composante alpha est bornée entre 0 et 1, nous mettons la matrice Q' à l'échelle au niveau du vertex shader de sorte que l'iso-valeur définissant la frontière du splat soit 0.5.

3.2.2.2 Passe de reconstruction

Lors de cette passe les splats sont accumulés dans les tampons d'attributs. Nous utilisons pour ces tampons des textures RGBA en précision flottante avec 16 bits par composante. Le nombre de textures utilisées est fonction du nombre total des composantes des attributs à accumuler.

Afin d'alléger le fragment shader réalisant l'évaluation du filtre de ré-échantillonnage, il est possible de réaliser la multiplication des valeurs des attributs par le poids au niveau de l'étage de blending. Le poids est alors stocké dans la composante alpha et le blending est configuré de manière à réaliser :

$RGB_{dest} \leftarrow RGB_{dest}$ `+` RGB_{frag} `*` A_{frag}
$A_{dest} \leftarrow A_{dest}$ `+` A_{frag}

Contrairement à ce que l'on pouvait attendre, nous avons observé aucun surcoût dû à l'utilisation de deux tampons de destination par rapport à un seul[5]. Par contre, l'utilisation de trois ou quatre tampons réduit considérablement les performances. Il est donc conseillé, lorsque cela est possible, de se limiter à deux tampons de destination. Dans ces conditions, il peut être nécessaire de reporter le produit `attribut` $\times$ `poids` au niveau du *fragment shader* afin de ne pas consommer une composante inutilement.

De plus, toujours dans le souci de maximiser le nombre de composantes disponibles, il est possible de *compresser* la normale sur seulement deux composantes puisque celle-ci ne représente qu'une direction. En coordonnées sphériques, une direction peut en effet être représentée par seulement deux angles (θ, ϕ) (la troisième coordonnée, i.e. le rayon, est toujours égale à 1). Le problème est que, d'une manière générale, il n'est pas possible d'interpoler ou de moyenner des directions exprimées en coordonnées sphériques à cause de la discontinuité de l'angle θ. Cependant, les normales que nous interpolons sont toutes orientées vers l'observateur. En choisissant habilement la convention pour le passage en coordonnées sphériques, il est donc possible d'éviter d'avoir à moyenner entre elles deux normales situées de part et d'autre de la discontinuité.

Soit (n_x, n_y, n_z) les coordonnées d'une normale exprimées dans le repère de la caméra. Le passage en coordonnées sphériques (θ, ϕ) peut par exemple être réalisé de la manière suivante :

$$\theta = signe(n_x) acos\left(\frac{n_z}{\sqrt{n_x^2, n_z^2}}\right) \in [-\pi, \pi] \tag{3.19}$$

$$\phi = acos(n_y) \in [0, \pi] \tag{3.20}$$

La décompression, c'est-à-dire le passage de (θ, ϕ) à (n_x, n_y, n_z), est donnée par les formules suivantes :

$$n_x = sin\phi\, sin\theta \tag{3.21}$$

$$n_y = cos\phi \tag{3.22}$$

$$n_z = sin\phi\, cos\theta \tag{3.23}$$

En pratique, afin d'éviter l'utilisation de coûteuses formules trigonométriques, spécialement les *acos*, la compression est efficacement réalisée grâce à une *cube map* précalculée tandis que la décompression peut être précalculée dans une texture 2D.

Se pose également la question de la profondeur. Celle-ci est en effet indispensable si l'on a besoin de connaître la position 3D du pixel lors de la passe de deferred shading. C'est par exemple

[5] La bande passante nécessaire à un blending sur huit composantes de 16 bits chacune est le double de celle disponible sur une Geforce6 ou Geforce7. Une explication possible est que le surcoût doit être masqué par le coût des *vertex* et *fragment shaders* qui sont pourtant très simples.

le cas pour calculer un éclairage avec des sources locales. Bien que la profondeur des pixels soit déjà stockée dans le z-buffer, utiliser ce dernier comme texture pour obtenir la profondeur pose des problèmes de précision majeurs. En effet, les valeurs contenues dans le z-buffer sont d'une part décalées de ϵ_k (qui n'est pas constant pour tous les pixels) et d'autre part sont seulement C^{-1} continues. Plus important encore, un autre problème d'ordre technique est que lorsqu'une texture de profondeur est utilisée pour lire la valeur contenue, et non pour réaliser une comparaison, alors seuls les 8 bits les plus significatifs sont accessibles, ce qui pose évidement de gros problèmes de précision. En pratique, il est donc préférable d'interpoler séparement la profondeur. Pour résumer, voici une configuration typique d'utilisation des huit composantes offertes par deux tampons de destination au format RGBA, avec reconstruction d'une couleur $\mathbf{c}$, d'une normale $\mathbf{n}$, d'un coefficient de transparence α (voir section 3.3.2) et de la profondeur z :

$$\text{Tampon 0} \begin{cases} R & \leftrightarrow & \mathbf{c}_{rouge} \\ G & \leftrightarrow & \mathbf{c}_{vert} \\ B & \leftrightarrow & \mathbf{c}_{bleu} \\ A & \leftrightarrow & w \text{ (somme des poids)} \end{cases}, \qquad \text{Tampon 1} \begin{cases} R & \leftrightarrow & \theta \\ G & \leftrightarrow & \phi \\ B & \leftrightarrow & z \\ A & \leftrightarrow & \alpha \end{cases}$$

3.2.2.3 Passe de normalisation et *deferred shading*

Les étages de normalisation et de *deferred shading* sont exécutés par un même *fragment shader*. Un fragment est généré pour chaque pixel en traçant un rectangle de la taille de l'écran. Les attributs attachés à ce pixel sont obtenus en utilisant les tampons de destination de la passe précédente comme textures. La normalisation consiste juste à multiplier les valeurs accumulées par l'inverse de la somme des poids.

Ensuite, la plupart des techniques de calcul d'ombrage couremment utilisées avec les méthodes de rendu par polygones sont également réalisables ici. Typiquement, l'éclairage local peut être calculé via le modèle de Phong. Les ombres portées peuvent être obtenues par une technique basée image telle que celle des *shadow maps*. Les réflexions peuvent quant à elles être réalisées en utilisant des *cube maps*.

3.3 Améliorations

3.3.1 Rendu hybride splats/polygones

Dans le cadre du rendu de scènes complexes, de nombreux travaux ont montré qu'un rendu hybride points/polygones est le plus efficace actuellement (voir section 2.3.3). La représentation polygonale est utilisée pour les zones des objets pour lesquelles les points deviennent moins efficace, c'est-à-dire dès que la taille d'un point dans l'espace image dépasse quelques pixels. Mais aucun travail ne s'est vraiment penché sur la qualité du rendu de la transition splats/polygones. Afin de limiter les artefacts visuels, il est en effet nécessaire de réaliser un lissage au niveau de ces transitions points/polygones.

Nous proposons ici une méthode qui n'est sans doute pas optimale d'un point de vue de la qualité mais qui a pour avantages d'être facile à mettre en oeuvre et de ne pas alourdir les calculs de rendu. L'idée est d'utiliser les poids issus des filtres Gaussiens des splats pour mélanger par alpha-blending les splats et les polygones appartenant à une même surface. Nous pouvons remarquer que les splats situés à la frontière splats/polygones ne sont pas recouverts par d'autres splats du côté des polygones (figure 3.6). Ainsi, les sommes des poids calculées par accumulation lors de la passe de reconstruction varient approximativement de 1 au centre des

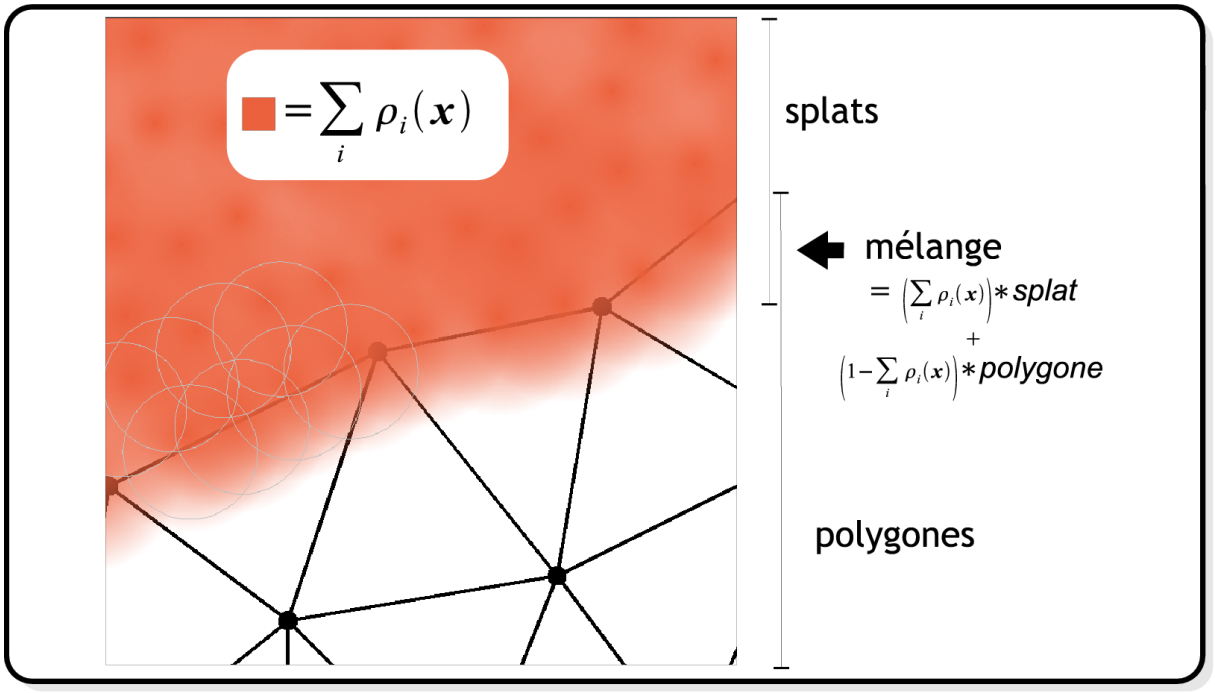


FIG. 3.6 – Rendu hybride splat/polygones : le mélange peut être effectué par la somme des poids accumulés lors du splatting.

splats à 0 aux bords des splats. Ces poids peuvent donc être utilisés pour mélanger les splats et polygones d'une même surface, c'est-à-dire ayant une différence de profondeur inférieure à ϵ. L'algorithme de rendu devient alors :

1. Passe de splatting de visibilité.
2. Tracé des polygones : le z-buffer précédent est utilisé pour éliminer les parties cachées par la représentation par splats. Ce rendu est effectué dans une texture RGBA avec ou sans *deferred shading*. Durant cette passe, le z-buffer est mis à jour.
3. Passe de reconstruction/accumulation des attributs des splats. Grâce à la mise à jour du z-buffer par la passe précédente, seules les contributions réellement visibles (i.e. visibles par rapport aux splats et aux polygones) sont accumulées. Afin de réaliser un test de profondeur à ϵ près avec les polygones, les splats sont décalés de $-\epsilon$ dans la direction de visée.
4. Passe de normalisation et *deferred shading*. En plus, en chaque pixel, la somme des poids doit être copiée dans la composante alpha du tampon de rendu final.
5. Finalement l'image obtenue à la passe 2 (image de la représentation polygonale), est combinée au résultat précédent par alpha-blending. Mais contrairement à un alpha-blending classique, le coefficient alpha utilisé pour le mélange est celui du tampon de destination :

$$RGB_{dest} \leftarrow RGB_{dest} * alpha_{dest} + RGB_{fragment} * (1 - alpha_{dest})$$

ou autrement dit :

$$RGB_{final} \leftarrow RGB_{splats} * poids_{splats} + RGB_{polygones} * (1 - poids_{splats})$$

Bien sûr, ce mélange doit être effectué uniquement pour les zones de recouvrement splats-polygones. Pour les zones ou il n'y a pas de splats, la valeur de $poids_{splats}$ est nulle et donc la couleur des polygones n'est pas altérée. Par contre, les régions où il n'y a pas de polygones projetés doivent être éliminées en amont car nous n'avons aucune garantie que $poids_{splats}$ soit égal à 1 pour ces régions. Les pixels *vide* de l'image de la représentation polygonale doivent donc être éliminés lors de cette dernière passe de rendu, par exemple en utilisant le test alpha ou le test de *stencil*.

3.3.2 Rendu des objets semi-transparents

La visualisation de scènes comportant des surfaces semi-transparentes est généralement réalisée par alpha-blending avec un rendu ordonné des primitives d'arrière en avant. Après avoir tracé l'ensemble des objets opaques de manière classique, la mise à jour du z-buffer est désactivée et les polygones des objets non opaques sont tracés du plus éloigné au plus proche de d'observateur. Les fragments des primitives sont mélangés avec le fond de la manière suivante :

$$RGB_{dest} \leftarrow RGB_{dest} * (1 - alpha_{fragment}) + RGB_{fragment} * alpha_{fragment} \tag{3.24}$$

Ce type d'algorithme n'est malheureusement pas transposable à un rendu par splatting puisque les attributs de couleur et de transparence doivent d'abord être reconstruits (i.e. accumulés et normalisés) avant d'être mélangés. De plus, une telle approche n'est de toute façon pas compatible avec une technique de rendu par *deferred shading*.

Jusqu'à présent la seule solution qui a été proposée pour gérer la transparence est d'utiliser un A-buffer modifié, comme nous l'avons vu section 2.2.2.3. Rappelons que le principe de base était d'accumuler les splats ayant une différence de profondeur trop importante, c'est-à-dire n'appartenant pas à la même surface, dans des tampons/couches différentes et de les combiner par alpha-blending après avoir appliquer la passe de deferred shading sur chacune des couches. Malheureusement, les GPU actuels ne proposent aucune fonctionnalité de ce type.

Aussi, nous proposons ici de simuler un A-buffer par un rendu multi-passe (*depth-peeling*). Tout d'abord, les objets opaques sont tracés de manière classique : passe de visibilité, passe de reconstruction et passe de deferred shading. Le z-buffer des objets opaques est copié dans une texture de profondeur. Ensuite, les objets semi-transparents sont tracés k fois de manière à obtenir k images RGBA représentant les différentes couches reconstruites (et éclairées) de la plus proche à la plus éloignée de l'observateur. Chaque passe est en fait composée d'une passe de visibilité, d'une passe de construction et d'une passe de deferred shading. Celles-ci sont réalisées de la manière suivante :

- Copier le z-buffer obtenu à la passe précédente dans une texture accédée en lecture (sauf pour la première passe).
- Vider les tampons de destination (z-buffer et tampons d'attributs).
- Passe de splatting de visibilité. Un test de profondeur supplémentaire est réalisé au niveau du fragment shader via un accès texture au z-buffer de l'étape précédente (sauf pour la première passe). La comparaison est configurée de sorte que seuls les fragments étant plus éloignés passent le test. Le z-buffer est quant à lui configuré de manière classique.
- Passe de reconstruction. Comme pour la passe de visibilité un test de profondeur supplémentaire est réalisé de sorte à ne laisser passer que les fragments situés derrière la couche précédente. Pour cette passe, la profondeur des fragments est également comparée

au z-buffer des objets opaques. Ainsi seules les contributions réellement visibles seront accumulées.
- Passe de normalisation et deferred shading classique sauf que le résultat est stocké dans une texture.

Les images obtenues doivent ensuite être combinées entre elles afin d'obtenir l'image finale. Pour cela, l'étage de blending est configuré en mode alpha-blending et les images des couches sont tracées une à une dans le tampon contenant l'image des objets opaques. Les couches doivent être tracées de la plus éloignée (numéro $k-1$) à la plus proche (numéro 0).

Reste bien sûr la question du nombre de passes qui doivent être réalisées. Théoriquement, il faudrait réaliser au moins autant de passes qu'il y a de couches de surfaces semi-transparentes, mais évaluer ce nombre de couches à chaque image est un problème difficile. De plus, afin de limiter les temps de calcul il est nécessaire de limiter au maximum le nombre de passes. En pratique, nous suggérons de borner le nombre de passes à trois ou quatre. Les cas de figure dans lesquels plus de quatre morceaux de surface semi-transparente se projettent sur un même pixel sont en effet assez rares. De plus, lorsque tel est le cas, les contributions des couches les plus éloignées sont très atténuées. Il doit donc être possible de mélanger les couches des plus éloignées dans une seule et même couche sans générer d'artefacts important. La dernière passe est alors modifiée de manière à accumuler entre eux tous les splats compris entre la dernière couche semi-transparente reconstruite et les objets opaques. Celle-ci est alors considérablement simplifiée puisque la passe de splatting de visibilité est simplement remplacée par une copie du z-buffer des objets opaques dans le z-buffer de rendu. Le test par rapport à ce z-buffer effectué au niveau du fragment shader de la passe de reconstruction n'est donc plus nécessaire. Une astuce pour limiter les artefacts est d'utiliser le coefficient de transparence pour moduler le poids des contributions. De cette manière, si deux surfaces différentes sont mélangées alors la plus opaque aura plus d'importance. Une autre solution, peut-être meilleure, pourrait être d'utiliser les rapports de profondeur entre le fragment, la couche précédente et les objets opaques afin de donner plus de poids aux fragments les plus proches.

3.4 Résultats et comparaisons

3.4.1 Qualité du rendu

Notre algortihme de rendu étant initialement basée sur la technique de l'EWA surface splatting, la qualité des images produites est bien sûr identique à celle de l'EWA surface splatting. Dans le but de maximiser les performances, nous avons cependant proposé, à la section 3.2.1.4, une approximation du filtrage EWA dont il convient d'évaluer son impact sur la qualité du filtrage. Pour cela, nous avons effectué quatre types de rendu différents sur un modèle de damier comportant 360 000 points (figure 3.7) et un modèle de caméléon comportant plus de 700 000 points (figure 3.8).

1. Rendu de référence avec un filtrage EWA complet.

2. Rendu sans filtre-bas : nous pouvons remarquer que pour les deux modèles, les images présentent un fort aliasing dû aux hautes fréquences des textures.

3. Rendu avec notre approximation de l'application du filtre passe-bas : pour le modèle du damier nous pouvons remarquer, avec beaucoup d'attention, une très légère baisse de la

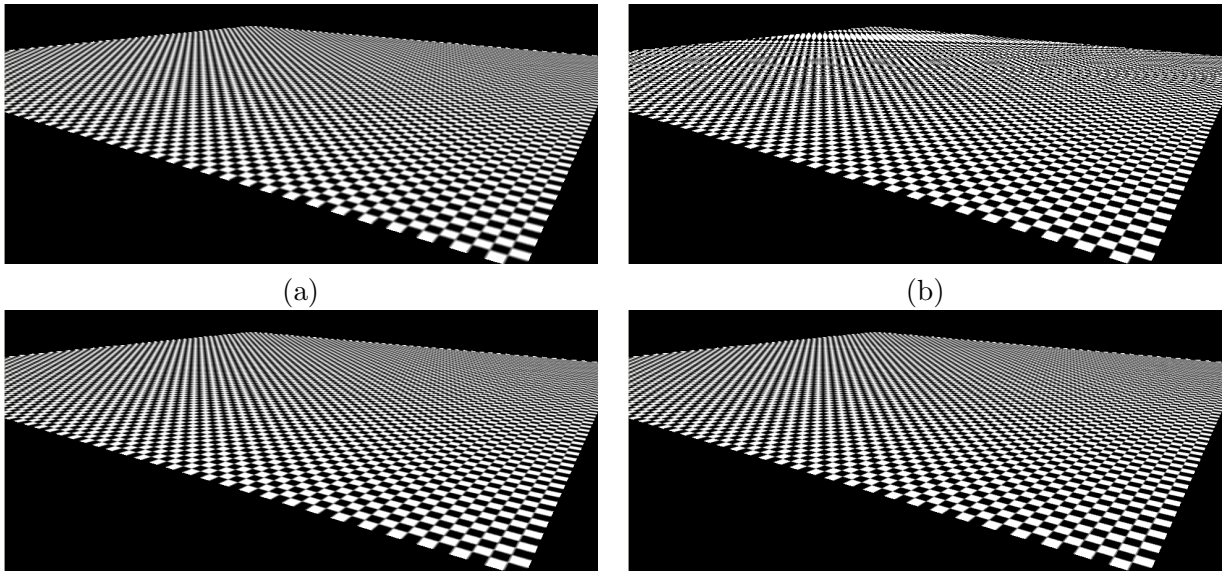


FIG. 3.7 – Comparaison des approximations de l'EWA splatting. (a) EWA splatting (référence) (b) Splatting sans aucun filtre passe-bas. (c) Notre approximation de l'EWA splatting. (d) Approximation de l'EWA splatting de [BHZK05] (plus perspective exacte).

qualité du filtrage pour les splats ayant une taille de l'ordre du pixel ; les splats plus petits sont quant à eux correctement filtrés. Nous pouvons aussi remarquer moins de flou dans les zones d'agrandissement. Cependant, pour un modèle plus courant comme le caméléon, aucune différence n'est décelable.

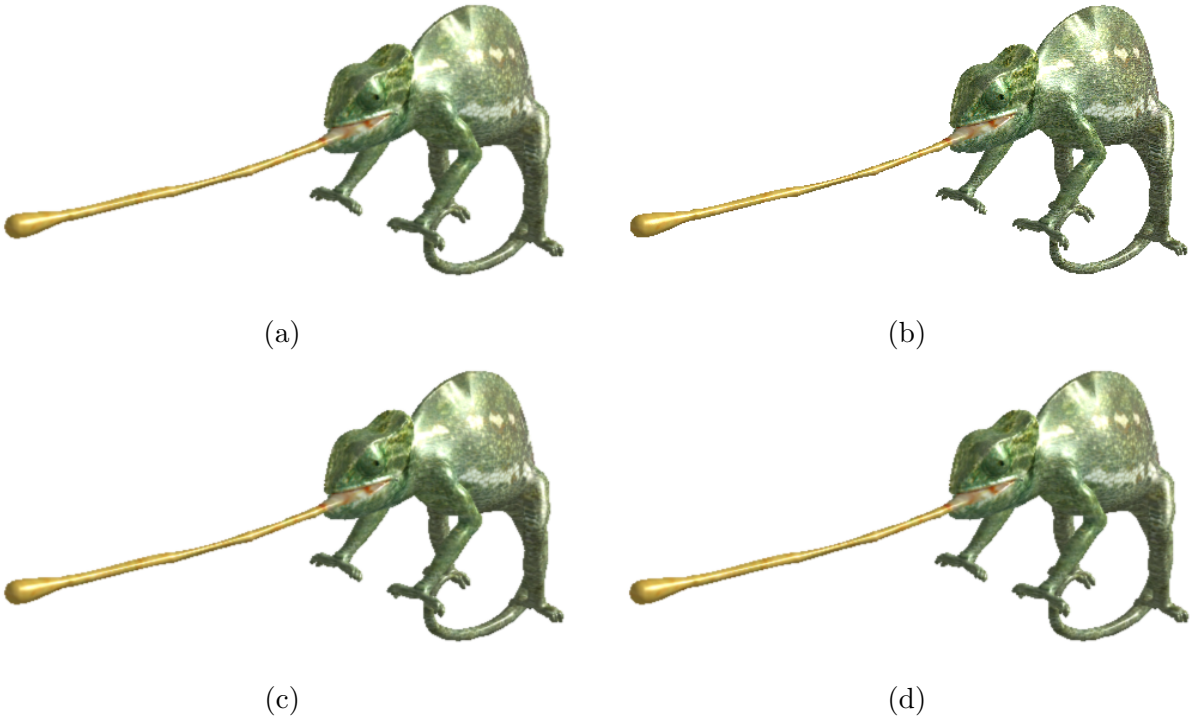
4. Rendu avec la méthode de [BHZK05]. Bien que théoriquement plus éloignée du filtrage EWA que notre propre approximation (voir figure 3.5), les résultats de leur approximation du filtrage EWA sont sensiblement équivalents aux notres.

L'impact de nos approximations sur la qualité du rendu étant quasiment nul, dans la suite des résultats nous considérerons uniquement notre implantation avec ces approximations. Les figures 3.1 et 3.9-a montrent respectivement le rendu par notre algorithme d'un objet semi-transparent et la combinaison de notre algorithme de splatting avec la technique des *shadow maps* pour le rendu des ombres portées.

3.4.2 Performances

Du point de vue des performances, nous allons comparer notre nouvelle implantation à quatre autres implantations du splatting :

1. EWA splatting sans les optimisations du calcul du filtre de ré-échantillonnage ni les approximations que nous avons proposées. Cela correspond en quelque sorte à l'implantation que nous avions proposée dans [GP03] mais avec une approche par deferred shading.
2. La version du splatting de Botsch et al. [BHZK05].
3. Splatting via la primitive point standard et deferred shading. Cette technique de rendu permet essentiellement d'évaluer les coûts de nos shaders puisque celle-ci fait principale-



(a) (b)

(c) (d)

FIG. 3.8 – Un modèle de caméléon de 770 000 points avec une texture hautement détaillée. (a) EWA splatting (référence) (b) Splatting sans filtrage des hautes fréquences. (c) Splatting avec notre approximation du filtrage EWA. (d) Approximation de l'EWA splatting de [BHZK05] (plus perspective exacte).

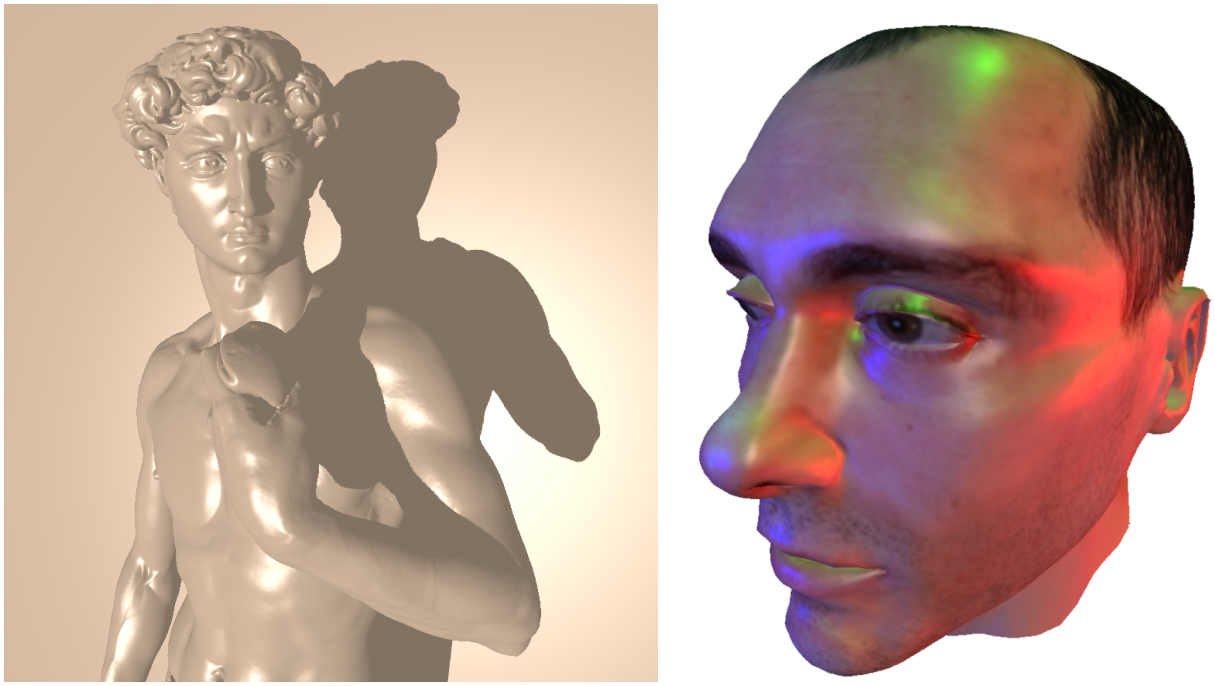
ment appel au *pipe-line* fixe de la carte graphique. Cependant, nous avons quand même dû utiliser un petit *fragment shaders* indispensable pour pouvoir effectuer le rendu dans des tampons de destination flottants. Ce *fragment shader* se contente d'effectuer la copie des attributs. Afin de reproduire une sorte de filtrage EWA, nous avons défini la taille minimale des points à 2×2 pixels.

4. Splatting avec nos optimisations mais sans deferred shading : le rendu est effectué dans un seul tampon de destination avec 8 bits par composante. Cette technique permet essentiellement d'évaluer le coût lié à l'utilisation de plusieurs tampons de destination en précision flottante.

Dans tous les cas, nous avons activé le filtrage EWA (ou l'approximation respective) et utilisé un modèle d'éclairage local très simple de type Blinn. Excepté pour la dernière implantation, nous avons utilisé deux tampons de destinations avec 16 bits par composante. Les performances respectives de ces implantations sont résumées dans le tableau 3.1. Ces mesures ont été effectuées avec une résolution d'image de 768×768 et pour trois objets de complexité très différente : un visage de 285k points (figure 3.9-b), un caméléon de 772k points (figure 3.8) et une statue de David de 3.6 millions de points (figure 3.9-a).

De plus, précisons que ces résultats ont été obtenus avec une carte graphique NVida GeForce 6800 et un processeur AMD Athlon64 à 2.2GHz sous GNU/Linux. Tous les shaders utilisés ont été implantés en utilisant le langage de shader d'OpenGL 2.0 [KBR04].

À partir de ce tableau, nous pouvons tout d'abord noter une augmentation significative des performances grâce à nos optimisations. Bien que nous pouvons remarquer un léger surcoût dû



(a) (b)

FIG. 3.9 – (a) Statue de David (3 millions de points). Sur cette image les ombres portées ont été obtenues avec la technique des shadow-maps. (b) Modèle de tête (285k points).

Méthode de rendu	# instr.	Tête (286k)	Caméléon (772k)	David (3.6M)
0 - Notre approche de l'EWA splatting	37/6(+3)	39/11.1	28/21.6	6.3/22.8
1 - EWA splatting sans nos optimisations	51/6(+3)	33/9.4	20/15.4.2	4.5/16.2
2 - [BHZK05]	28/11(+3)	32/9.2	23/17.8	5.2/18.8
3 - Splatting via le *pipe-line* fixe	0/0(+3)	54/15.4	39/30.1	9.3/33.6
4 - Notre approche (sans deferred shading)	39/6(+1)	56/16	28.5/22	6.25/22.6

TAB. 3.1 –
Performances de différentes implantations de l'EWA splatting. La deuxième colonne indique le nombre d'instructions des *vertex* et *fragment shaders* de la passe d'accumulation. Les instructions de copie des attributs au niveau des *fragment shaders* ne sont pas compatibilisées mais leur nombre est indiqué entre parenthèses. Pour chaque modèle, nous reportons le nombre d'images par seconde (fps) et le nombre de millions de points tracés par seconde.

à l'utilisation de deux tampons de destination pour un objet relativement simple, ce surcoût est rapidement amorti dès que la complexité des objets augmente. Malgré l'utilisation de shaders relativement simples, leur coût est loin d'être négligeable, comme le montrent les résultats de l'implantation numéro 3. De plus, pour cette implantation le nombre de fragments accumulés est bien plus important puisque les ellipses des filtres de ré-échantillonnage sont approchées

par des carrés parallèles aux axes. Si les performances augmentent alors que le nombre de fragment augmente, cela signifie bien que l'utilisation du blending flottant avec deux tampons de destination n'est pas limitant. Le surcoût important observé pour un objet simple est donc plutôt surprenant.

3.4.3 Comparaison avec les travaux de Botsch et al. [BHZK05]

Bien que la récente proposition de splatting sur GPU de Botsch et al. soit d'une certaine façon similaire à la notre, nous pouvons tout de même noter de nombreuses différences :

1. Botsch et al. presentent un calcul exact de la projection perspective des splats par lancer de rayon au niveau des *fragment shaders*.
2. Ils utilisent aussi une approximation de l'application d'un filtre passe-bas en prenant le max de l'évaluation du noyau de reconstruction et du filtre passe-bas. Comme le montre la figure 3.5, cette approximation est assez grossière, particulièrement lorsque la projection des splats est très allongée. Mais le principal inconvénient de leur approximation est de nécessiter trois opérations supplémentaires au niveau du *fragment shader*, ce qui est assez coûteux puisque qu'il s'agit déjà du goulot d'étranglement d'un rendu par splatting. Par comparaison, notre approximation ne nécessite que deux opérations supplémentaires exécutées une seule fois par splat et non pour chaque fragment.
3. De notre côté, nous proposons en plus un support des objets semi-transparents et un support pour le filtrage des transitions splats/polygones en cas de rendu hybride.

Au niveau des performances, notre approche présente un meilleur équilibre de la répartition des calculs au niveau des *vertex* et *fragment shaders*. Une approche par lancer de rayon implique un *vertex shader* trivial mais aussi un *fragment shader* plus complexe, surtout si le filtre passe-bas est activé.

3.5 Conclusion

Notre méthode de rendu de nuages de points par splatting accéléré par le GPU présente sans doute l'un des meilleurs compromis qualité/vitesse du moment. Notre nouvelle méthode de calcul des paramètres du filtre de ré-échantillonnage associée à une évaluation par pixel de la profondeur et du filtre de ré-échantillonnage très efficace permet une très bonne répartition des calculs entre les *vertex* et *fragment shaders*. En outre, les fonctionnalités des dernières générations de cartes graphiques nous ont permis de proposer une approche par *deferred shading* augmentant simultanément les trois composantes du compromis rapidité/qualité/flexibilité que nous nous sommes fixées au départ :

- Performances. Le *deferred shading* permet d'effectuer les calculs de l'ombrage uniquement pour les pixels visibles. Avec un rendu par splatting requérant un recouvrement des splats, cet avantage est significatif même pour une scène simple.
- Qualité. Le *deferred shading* permet d'interpoler les attributs entre les points avant les calculs d'ombrage. De cette manière, les calculs d'ombrage sont d'une très bonne qualité puisque réalisés par pixel.
- Flexibilité. Le gain en flexibilité apporté par le *deferred shading* est de deux natures. Premièrement, de nombreux attributs peuvent maintenant être interpolés entre les points permettant la mise en place de shaders complexes. Deuxièmement, la séparation des calculs nécessaires au splatting des calculs d'ombrage, simplifie le développement de *shaders*

et permet même la réutilisation de *shaders* développés pour les maillages polygonaux.

Nous avons également proposé deux extensions de notre méthode permettant la prise en compte des objets semi-transparents et le filtrage des transitions splats/polygones en cas de rendu hybride.

Le principal inconvénient de notre approche par rapport aux autres méthodes de splatting existantes est le fait que nous nous appuyons sur une approximation affine de la projection perspective ne permettant pas de garantir l'absence de trous. Cependant, grâce au recouvrement des splats dans l'espace objet nous n'avons jamais remarqué le moindre artefact. De plus, c'est précisement cette approximation qui nous a permis de dériver une implantation aussi efficace.

À l'heure actuelle, la principale limitation des dernières cartes graphiques pour le splatting de nuages de points est toujours l'absence d'un z-buffer offrant plus de flexibilité (c'est-à-dire permettant de réaliser des opérations de blending différentes en fonction du résultat d'un test de profondeur réalisé à ϵ près), ce qui doit être compensé par une coûteuse passe de splatting supplémentaire. Cependant, comme nous allons le voir dans le chapitre suivant, cette séparation des calculs de visibilité et de la reconstruction peut être tourné en avantage via une réorganisation des passes de splatting.

Chapitre 4

Visualisation des scènes complexes par *deferred splatting* [6]

FIG. 4.1 – Une forêt de 6800 arbres, composés de 750 000 points chacun, rendue en temps réel par *deferred splatting*.

Introduction

Comme nous l'avons vu au chapitre précédent, les cartes graphiques actuelles offrent suffisement de flexibilité pour implanter un rendu de haute qualité des nuages de points. Cependant, les performances obtenues sont loin de celles que nous pourrions obtenir avec un matériel dédié, ou avec une carte graphique qui supporterait nativement la rastérisation et mélange de splats. Alors que conceptuellement plus simple, à l'heure actuelle, le rendu d'un point est finalement environ

[6] Ces travaux ont été publiés dans : [GBP04a].

deux fois plus coûteux que le rendu d'une face triangulaire. Dans ces conditions, les points ne deviennent plus intéressant que les triangles que lorsque la taille d'un triangle devient inférieur à un demi-pixel. Ce constat est absurde puisque cela signifie qu'il est équivalent de tracer deux triangles (i.e. 6 sommets) se projetant sur un seul pixel ou de tracer un seul point ! Ceci explique pourquoi la plupart des techniques de visualisation de scènes complexes basées points, qu'elles soient hybrides ou non, se sont contentées d'un simple rendu de points sans aucun filtrage. Pour donner une idée sur le coût d'un splatting de qualité comparé au rendu simple de points via la primitive *point* des cartes graphiques, une GeForce6 est, par exemple, capable de tracer entre 100 et 130 millions de points par seconde contre seulement 20 millions de splats filtrés.

Notre principal objectif ici est donc de permettre l'utilisation efficace de notre technique de splatting de haute-qualité pour le rendu de scènes complexes. Le splatting en lui-même ne pouvant être optimisé d'avantage, il est primordial de limiter les opérations coûteuses du splatting au splats réellement visibles.

La détermination des points visibles est réalisée par des algorithmes de sélection, en amont de l'opération de rendu. Ces algorithmes de sélection incluent les algorithmes de *view-frustum*, *back-faces* et *occlusion culling* ainsi que les méthodes de sélection des niveaux de détails permettant de supprimer les points superflus en cas de réduction. Ces algorithmes de sélection peuvent être réalisés à différents niveaux :

Haut niveau : Comme leur nom l'indique, ces algorithmes travaillent sur une représentation de haut niveau de la scène. En général, il s'agit de volumes englobants d'objets ou d'ensembles de primitives géométriques, structurés au sein d'une hiérarchie.

Bas niveau : Par opposition à la catégorie précédente, ces algorithmes travaillent au niveau même des primitives et, dans la phase finale de rastérisation, au niveau du pixel.

Les derniers travaux dans ce domaine s'intéressent principalement aux algorithmes de sélection de haut niveau [COCSD03] qui fournissent rapidement une solution de visibilité grossière en amont de la sélection de bas niveau. En général, les calculs de visibilité finaux sont laissés entièrement à la charge de la carte graphique qui réalise uniquement des tests de view-frustum et back-face culling par primitive et des tests d'occlusion culling par pixel. Cependant, lors de l'utilisation de *shaders* complexes, il peut être vraiment avantageux de réaliser cette sélection de bas niveau nous-mêmes, avant l'envoi des primitives au matériel graphique.

Notre idée de départ est de tirer profit d'un rendu très rapide par points non filtrés afin de ne conserver que les points réellement visibles pour les passes coûteuses de splatting. Par analogie à la technique du *deferred shading* qui consiste à réaliser les calculs d'ombrages coûteux après le tracé des primitives afin de réaliser ceux-ci uniquement sur les pixels réellement visibles, nous avons appelé notre technique *deferred splatting*. Notre technique est basée sur une sélection efficace des points, primitive par primitive, tout en tirant avantage de la cohérence temporelle. Les principales caractéristiques sont :

- **Sans précalcul** : afin d'être exploitable pour le rendu de scènes dynamiques ou interactives, notre algorithme ne doit pas dépendre d'une phase de précalcul coûteuse.
- **Précis** : notre algorithme doit réaliser une sélection de view-frustum et occlusion culling au niveau des points.
- **Bas niveau** : notre algorithme doit pouvoir prendre en entrée le résultat de tout algorithme de sélection de haut niveau.
- **Gestion de la cohérence temporelle** : notre algorithme exploite la cohérence spatio-temporelle afin d'accélérer les traitements.

- **Implantation sur carte graphique** : notre algorithme sait tirer parti de la puissance des cartes graphiques actuelles.

La suite de ce chapitre est organisée de manière suivante. La section suivante est consacrée à la présentation de l'algorithme de *deferred splatting*. Dans la section 4.2, nous discutons des impacts et limites de notre nouvelle algorithme de sélection. Section 4.3, nous montrons comment notre méthode permet la mise en oeuvre simple d'un algorithme d'occlusion culling efficace et nous montrons également qu'en réalisant quelques approximations, notre algoritme est implantable à 100% sur le GPU. Enfin, section 4.5 nous discutons de l'avenir du *deferred splatting* et de quelques autres points.

4.1 L'algorithme *deferred splatting*

4.1.1 Aperçu de l'algorithme

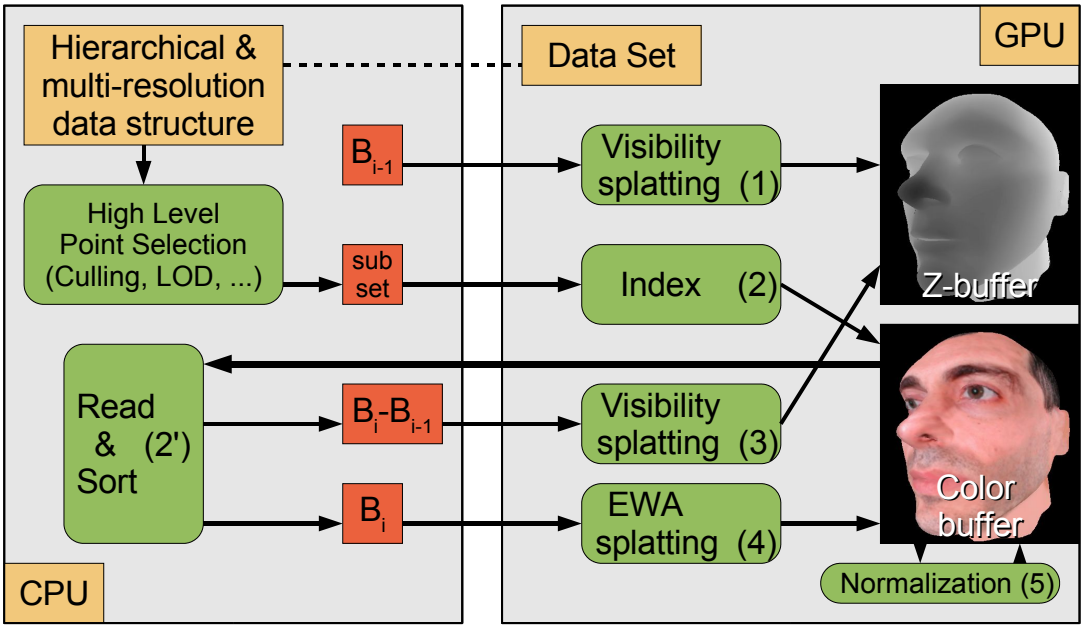


FIG. 4.2 – Aperçu du fonctionnement de l'algorithme de deferred splatting.

L'objectif du deferred splatting est de réaliser les opérations coûteuses du splatting (visibilité et attributs) uniquement sur les splats visibles.

La première idée est d'utiliser le tampon de profondeur obtenu par la première passe de splatting de visibilité pour calculer l'ensemble des points B_i visibles à l'image numéro i. Nous ajoutons donc une passe de rendu dans un tampon d'indices (passe numéro 2), où l'ensemble des points potentiellement visibles est tracé sous la forme d'un pixel contenant un identificateur unique. À la fin de cette passe simple et rapide, le tampon de destination contient donc uniquement les indentificateurs des points visibles puisque les points non visibles auront été rejetés par le test de profondeur. Ainsi, nous pouvons accélérer la passe de splatting des attributs (passe numéro 4) en ne traçant que les points de B_i.

La seconde idée est de tirer avantage de la cohérence spatio-temporelle entre deux images successives. Cela permet d'accélérer la première passe de splatting de visibilité (passe numéro

1) en ne traçant que les points B_{i-1} visibles à l'image précédente. Bien évidement, le z-buffer ainsi calculé est incomplet car les points non visibles à l'image précédente mais visibles à l'image courante ne seront pas tracés. Avant la passe de splatting des attributs, nous devons donc réaliser une passe de splatting de visibilité supplémentaire (passe numéro 3) en traçant les points nouvellement visibles définis par l'ensemble d'indices $B_i - B_{i-1}$.

Pour résumer, le rendu multi-passes de l'image numéro i devient (voir figure 4.2) :

1. Splatting de visibilité avec B_{i-1}
2. Passe de sélection (on obtient B_i)
3. Splatting de visibilité avec $B_i - B_{i-1}$
4. Splatting des attributs avec B_i
5. Normalisation des attributs et *deferred shading*

Dans les deux sous-sections suivantes, nous détaillons le processus de sélection point par point (passe 2) et la gestion de la cohérence temporelle (passe 3).

4.1.2 Sélection point par point

Cette section explique en détails la seconde passe de notre nouvel algorithme de rendu dont l'objectif est de sélectionner les points visibles un à un.

Le challenge est de tracer le plus rapidement possible l'ensemble des points potentiellement visibles avec un identificateur unique à la place de la couleur. Ainsi, les points sont envoyés à la carte graphique à l'aide de la primitive matérielle point avec une taille constante de un pixel. Le rendu est réalisé dans le tampon de couleur RGBA courant, limitant ainsi les identificateurs à 32 bits. De plus, nous devons être capable de trier rapidement les identificateurs par objets, car chaque objet a une matrice de transformation différente, des propriétés de matériaux différents, etc.

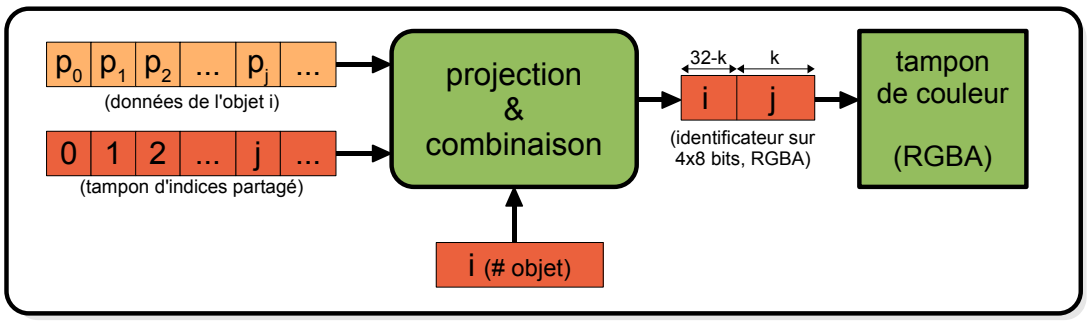


FIG. 4.3 – Calcul et rendu des identificateurs.

La solution la plus simple pour former un identificateur est d'utiliser k bits pour stocker l'indice j du point dans l'objet et les $32-k$ bits restant pour le numéro de l'objet i. Le choix de k est alors un compromis entre le nombre maximal de points par objet et le nombre maximal d'objets dans la scène. Un bon compromis est de prendre $k = 20$ autorisant plus d'un million de points par objets et 4096 objets potentiellement visibles. Comme la liste des objets potentiellement visibles varie au cours du temps, il est nécessaire de calculer les identificateurs à la volée au moment du rendu. Cela permet en plus de ne pas avoir de surcoût pour le stockage des identificateurs. Cette combinaison est réalisée par le GPU grâce à un simple *vertex shader*, ne nécessitant qu'une seule addition puisque les transformations de modélisation et projection

peuvent être réalisées par le *pipe-line* fixe. Le numéro de l'objet i est passé au *vertex shader* via un paramètre *uniform* tandis que l'indice du point est passé via l'attribut de couleur du point sous la forme de 4 octets non signés. Pour des raisons d'efficacité, les indices des points sont stockés dans un tableau en mémoire vidéo partagé par tous les objets. Ce tableau contient 2^k entiers consécutifs de 0 à $2^k - 1$. Le calcul et le rendu des identificateurs sont illustrés figure 4.3.

Comme tous les pixels du tampon de destination ne sont pas obligatoirement atteint par la projection d'un point, à la fin de cette passe, tous les pixels ne contiennent pas forcément un identificateur valide. Un dernier détail concerne donc le choix de la couleur d'effacement du tampon de destination : celle-ci ne doit jamais représenter l'identificateur d'un point existant. La couleur d'effacement doit donc correspondre à l'identificateur le plus improbable, c'est-à-dire `0xFFFFFFFF` qui correspond au dernier point permis du dernier objet potentiellement visible permis.

A la fin de cette passe de rendu, le tampon de couleur contenant les identificateurs des points réellement visibles est lu à partir de la mémoire vidéo et un tableau d'indices est construit pour chaque objet. Ces tableaux d'indices correspondent à l'ensemble que nous avons appelé B_i et sont utilisés pour le splatting des attributs de l'image courante et pour la passe de splatting de visibilité de l'image suivante.

Calcul des identificateurs pour les scènes très complexes

Dans la pratique, la valeur de k est choisie dynamiquement pour s'adapter à la vue courante en prenant le plus petit entier tel que $2^k \geq n$ où n est le nombre de points requis par le plus gros objet. Par exemple, une vue aérienne d'une forêt peut comporter bien plus que 4096 arbres (qui seront représentés par peu de points par objets puisque éloignés de l'observateur) tandis qu'un large objet proche de l'observateur peut nécessiter plusieurs millions de points.

Cependant, il se peut qu'aucun compromis pour le choix de k ne soit possible, i.e. si le nombre d'objets sélectionnés est supérieur à 2^{32-k}. Pour résoudre ce problème, on pourrait imaginer d'augmenter le nombre de bits disponibles pour stocker les identificateurs en utilisant par exemple les 8 bits du *stencil buffer*. Cette approche n'est malheureusement pas recommandée car cela augmente la quantité de données à transférer de la mémoire vidéo vers la mémoire centrale. Pour des raisons techniques, les transferts de la mémoires du GPU vers la mémoire centrale sont relativement lents, et en pratique cela doublerait le temps nécessaire à la lecture des identificateurs.

En fait, nous pouvons remarquer que, pour une vue donnée, peu d'objets sont réellement proches de l'observateur et donc peu d'objets à la fois requièrent un grand nombre de points. Le nombre de points requis par les objets les plus éloignés est considérablement réduit grâce aux niveaux de détails. Nous proposons donc un codage un peu plus sophistiqué des identificateurs consistant à utiliser plusieurs schémas de répartition des bits. Certains bits doivent alors être réservés pour coder le schéma utilisé. Les identificateurs et schémas sont calculés dynamiquement (à chaque image) de la manière suivante :

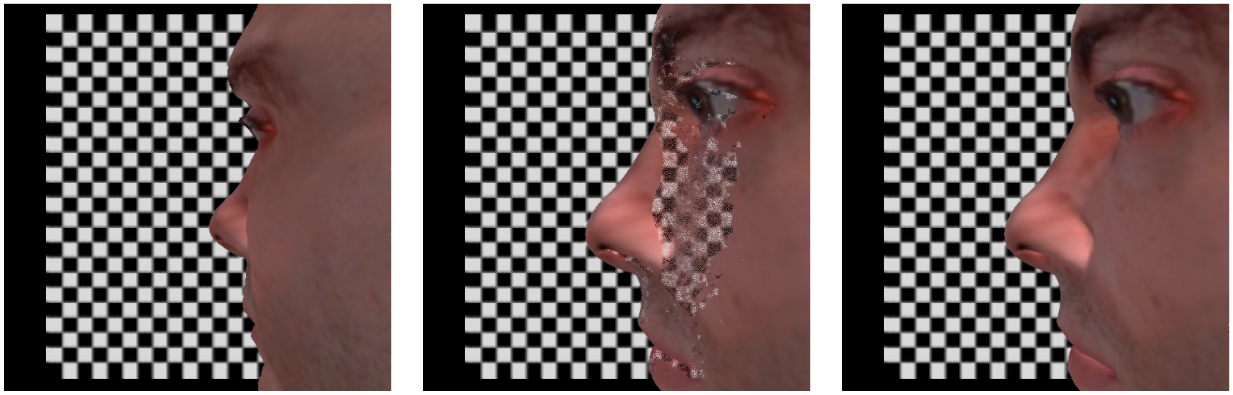
- Les objets potentiellement visibles sont numérotés par ordre décroissant du nombre de points qui les composent. Cela ne change pas l'ordre de tracé. Appelons n_i le nombre de points représentant l'objet numéro i.
- Deux schémas de répartition des bits représentés par k_0 et k_1 sont déterminés en prenant tout d'abord pour k_0 le plus petit entier tel que $2^{k_0} \geq n_0$. Rappelons qu'un bit de l'identificateur doit être réservé pour stocker le schéma utilisé (k_0 ou k_1). Le schéma basé sur k_0 est donc utilisé pour les 2^{31-k_0} plus gros objets, et k_1 est choisi au plus juste pour

représenter le maximum des plus petits objets, c'est-à-dire, k_1 est le plus petit entier tel que $2^{k_1} \geq n_{2^{31-k_0}}$.

- Bien que très peu probable en pratique, il se pourrait que le nombre total d'objets soit supérieur à $2^{31-k_0}+2^{31-k_1}$. Cela signifie que deux schémas ne sont pas suffisants. Nous pouvons alors passer à quatre schémas différents représentés par les quatre entiers k_0, k_1, k_2 et k_3. Ces valeurs sont calculées comme précédemment, à la différence près que seulement 30 bits sont disponibles puisque deux bits sont alors nécessaires au codage du schéma utilisé : nous prenons les plus petits entiers tel que $2^{k_0} \geq n_0$, $2^{k_1} \geq n_{2^{30-k_0}}$, $2^{k_2} \geq n_{2^{30-k_0}+2^{30-k_1}}$ et $2^{k_3} \geq n_{2^{30-k_0}+2^{30-k_1}+2^{30-k_2}}$.

L'inconvénient de cette approche est de nécessiter un tri des objets. Ce tri peut généralement être évité en fixant à priori deux schémas de répartition de bits. Par exemple, pour de vastes paysages comme notre forêt de test, nous utilisons $k_0 = 20$ et $k_1 = 17$. Cela permet d'avoir 2048 objets avec entre 131k et 1M de points et plus de 16k petits objets comportant moins de 131k points.

4.1.3 Cohérence spatio-temporelle



(a) (b) (c)

FIG. 4.4 – Illustration de l'impact et de la correction de la cohérence temporelle.
(a) Image initiale avec deux objets basés points : un échiquier et une tête. (b) Image suivante, après une large rotation de la tête, sans la seconde passe de splatting de visibilité. (c) Même image qu'en (b) mais avec la seconde passe de visibilité activée.

Après la passe de sélection expliquée à la section précédente, nous sommes capables de réaliser la passe de splatting des attributs uniquement sur les splats réellement visibles. Le coût de cette passe devient ainsi indépendant de la complexité de la scène. Cependant, la passe de splatting de visibilité est aussi relativement coûteuse. Dans cette section, nous montrons comment cette passe peut également être accélérée en prenant avantage de la cohérence spatio-temporelle entre deux images successives.

En effet, dans le cadre de la navigation intéractive dans une scène animée (ou non), le temps (réel) écoulé entre deux images succesives est très faible et donc à la fois le déplacement de la caméra et des objets est également faible. Ce cette manière, les différences entre deux images consécutives sont minimes, et la majeure partie des points visibles à l'image i sont donc toujours visibles à l'image suivante. L'idée est alors de tracer uniquement les points visibles à l'image précédente lors de la passe de splatting de visibilité. Cependant, l'ensemble des points non

visibles à l'image $i-1$ et devenant visibles à l'image courante i sont manquants. Le tampon de profondeur ainsi calculé est donc incomplet et peut comporter des trous. En pratique, ce tampon de profondeur est tout de même suffisant pour notre passe de sélection des points visibles, seuls quelques points non visibles seront inutilement sélectionnés.

Le problème est que ces trous potentiels dans le tampon de profondeur risquent de générer des artefacts dans l'image finale puisque des points non visibles seront accumulés dans les tampons d'attributs (figure 4.4-b). Ces artefacts peuvent être acceptables dans certaines applications où la navigation n'est pas un but mais juste un moyen pour obtenir un nouveau point de vue (pendant la phase de modélisation par exemple). En effet, une fois la caméra et la scène fixées, l'ensemble des points visibles ne varie plus ($B_{i-1} = B_i$) et il suffit alors d'un rendu supplémentaire pour que les artefacts soient supprimés. En fait, il faut bien comprendre que les artefacts ne s'accumulent pas d'une image à l'autre mais, au contraire, les artefacts d'une image sont corrigés à l'image suivante pour, éventuellement, laisser place à de nouveaux artefacts.

Correction du tampon de profondeur

Il est cependant possible de corriger ces artefacts à relativement peu de frais. Après avoir calculé B_i, il suffit de réaliser une passe de splatting de visibilité supplémentaire avec l'ensemble des points nouvellement visibles : $B_i - B_{i-1}$. Le challenge est alors de calculer le plus rapidement possible la différence $B_i - B_{i-1}$. Son calcul exact n'est malheureusement pas envisageable en pratique car beaucoup trop coûteux. Nous proposons donc d'en calculer une approximation de manière conservative. Pour cela, nous commençons par approcher l'ensemble B_{i-1} en le représentant par un tableau de booléen C_{i-1} où chaque élément est associé à un groupe de m points d'indices consécutifs : le k^{me} élément de ce tableau $C_{i-1}[k]$ correspond aux points d'indices de $m*k$ à $m*(k+1)-1$ (voir figure 4.5). Chaque élément indique l'appartenance à l'ensemble B_{i-1} d'au moins un point du groupe, ce tableau satisfait donc les deux propositions suivantes :

$$\begin{aligned} j \in B_{i-1} \quad &\Rightarrow \quad C_{i-1}[j/m] \\ \neg C_{i-1}[k] \quad &\Rightarrow \quad j \notin B_{i-1}, \forall\, j \in [m*k..m*(k+1)-1] \end{aligned} \tag{4.1}$$

où ici "/" dénote la division entière et "¬" l'opérateur de négation. Nous pouvons donc déterminer l'appartenance d'un point de B_i à B_{i-1} immédiatement (i.e. en $O(1)$). En pratique, la différence $B_i - B_{i-1}$ est calculée lors du tri du résultat de la passe de sélection, voir section précédente.

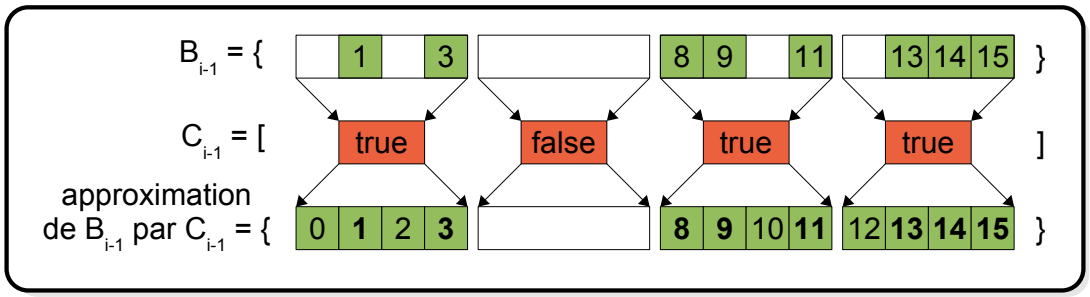


FIG. 4.5 – Approximation de l'ensemble d'indices B_{i-1} par le tableau de booléens C_{i-1}. Dans cet exemple nous avons pris $m = 4$.

Avec cette approche, nous supposons que tous les points représentés par l'approximation C_{i-1} ont été tracés lors de la première passe de splatting de visibilité. Afin de ne pas calculer une différence $B_i - B_{i-1}$ qui soit érronée, nous devons donc effectivement tracer, lors de la

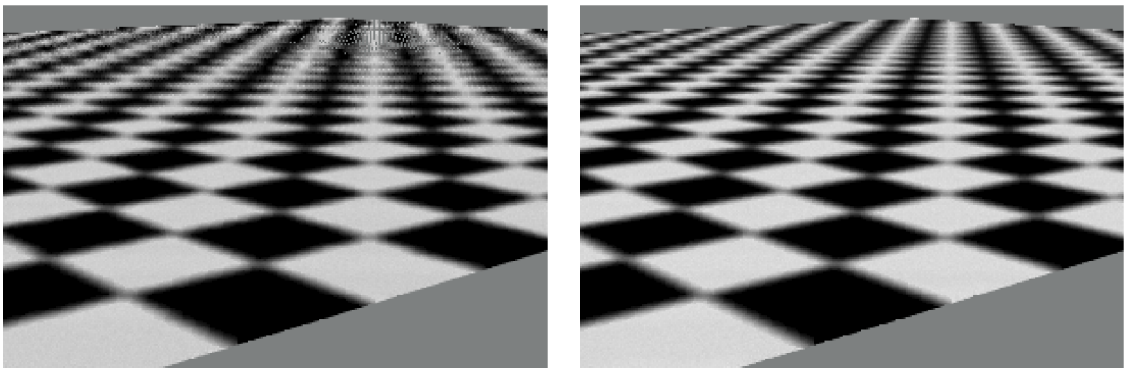
première passe de splatting de visibilité, l'ensemble des points représenté par C_{i-1} et non B_{i-1} exactement. Bien sûr, d'un coté, cela augemente légèrement le temps de rendu de cette passe puisque plus de points doivent être tracés. Mais d'un autre coté, puisque plus de points sont tracés, le risque de trous dans le tampon de profondeur devient moindre et ainsi, la différence $B_i - B_{i-1}$ devient plus petite et moins éparpillée entre les différents objets. Finalement, la seconde passe de splatting de visibilité corrigeant le tampon de profondeur devient donc quasiment négligeable (en terme de temps de rendu).

Le choix de la taille m des groupes est donc un compromis entre la précision et le coût mémoire. Pour des raisons d'efficacité m doit être une puissance de deux puisqu'ainsi l'indice du groupe de pixels contenant le point d'indice j est simplement obtenu en prenant les $32 - log_2(m)$ bits de poids forts de j. Dans notre implantation nous avons pris des groupes de $m = 32$ points, ce qui nécessite 16Mo de mémoire vive pour représenter un peu plus de 4 milliards de points.

Finalement, il est important de bien noter que cette approche est parfaitement compatible avec une scène animée et/ou contenant des objets déformables (figure 4.9-a). En effet, durant la première passe de splatting de visibilité, nous utilisons les matrices de transformation des objets ainsi que les attributs (position, normale, ...) des points correspondants à l'image courante i. Nos ensembles B_i ne stockent que les indices des points. La robustesse est principalement assurée par la correction du tampon de profondeur effectuée par la passe de splatting de visibilité supplémentaire.

4.2 Effets de bord du *deferred splatting*

4.2.1 Aliasing



(a) (b)

FIG. 4.6 – Illustration de la perte d'information de textures.
(a) Aliasing causé par une texture avec des hautes fréquences. (b) Lissage par interpolation des différents niveaux de texture.

Notre algorithme n'autorise qu'un seul point par pixel lors de la passe de sélection. Cela correspond en quelque sorte à une sélection de type "niveaux de détails" puisque les points superflus sont ainsi éliminés. De plus, cela résout également le problème d'*overflow* du splatting lorsque la précision des tampons est limitée à 8 bits par composante (c'est le cas pour les cartes graphiques ne supportant pas le blending avec des tampons en précision flottante) .

En revanche, en cas de fort sur-échantillonnage, de nombreux points superflus mais visibles seront écartés par notre algorithme de sélection. Cela signifie une perte d'informations sur la texture des objets introduisant de l'aliassage en cas de hautes fréquences dans la texture. Cela peut également générer des scintillements dans le cas de scènes animées ou de mouvements de caméra. En principe, ce problème devrait être corrigé par un algorithme de sélection des niveaux de détails. Il est toutefois possible de compenser l'imprécision ou l'absence d'un système de gestion des niveaux de détails en utilisant une technique similaire au mip-mapping présenté par Pfister et al. dans [PZvG00]. Le principe est de stocker pour chaque splat plusieurs niveaux de texture correspondant à un préfiltrage de la texture pour différents rayons du splats. Au moment du rendu, la couleur du splat est interpolée entre les deux niveaux de texture les plus proches par le *vertex shader* de la passe de splatting des attributs. Le coefficient d'interpolation est déduit de la mise à l'échelle des rayons par le filtre passe-bas :

$$\alpha = \sqrt{\frac{\lambda_1 + 1}{\lambda_1}} \tag{4.2}$$

Prenons l'hypothèse que chaque point stocke trois niveaux de texture de valeur c^0, c^1 et c^2 correspondant respectivement aux rayons de préfiltrage r, $2r$, et $4r$ (r est de rayon du splat). La couleur c du splat est alors interpolée par l'algorithme suivant :

---

si $\alpha > 2$ **alors**
    $t = \frac{\alpha-2}{2}$;
    $c = c_1 * t + (1-t) * c_2$;
sinon si $\alpha > 1$ **alors**
    $t = \alpha - 1$;
    $c = c_0 * t + (1-t) * c_1$;
sinon
    $c = c_0$;
finsi

4.2.2 Visibilité éronnée

Notre méthode est basée sur l'hypothèse qu'un point est visible ou non. Or, cette hypothèse n'est pas valide dans notre cas puisque nous ne considérons pas de simples points mais des splats, c'est-à-dire des disques orientés. En effet, pour un point de vue donné, un splat peut être partiellement visible alors que son centre ne l'est pas. Comme nous testons uniquement la visibilité des centres des splats, il se peut que quelques splats visibles soient malencontreusement éliminés. Cela peut se produire uniquement pour les splats des objets situés en arrière plan et proches de la silhouette d'un autre objet situé au premier plan. La question cruciale est donc de savoir quand un splat peut être réduit à son simple centre. La réponse dépend en fait de la position et de la résolution de la caméra. Les calculs de visibilité étant réalisés à la précision des pixels par le z-buffer, il est clair qu'un splat ayant une taille inférieure au pixel dans l'espace image peut être considéré comme un simple point. En pratique, principalement grâce au recouvrement des splats dans l'espace image, les *petits* splats, c'est-à-dire ceux ayant une projection inférieure à 10 pixels environ, peuvent être considérés comme de simples points sans grands artefacts dans l'image finale.

Finalement, la sélection effectuée par notre technique de *deferred splatting* ne doit être réalisée que pour les points ayant une projection suffisement petite. Les *gros* splats doivent être tracés quoi qu'il arrive. En pratique, cette classification est effectuée grossièrement en amont par un algorithme de sélection des niveaux de détails classique. Notons que cela n'est pas une forte limitation de notre approche puisque si un splat est *gros* alors, par définition, cela signifie qu'il est proche de l'observateur et a donc une forte chance d'être visible.

De plus, rappelons que dans le cadre du rendu de scènes très complexes, les meilleurs résultats, en terme de performances, ont été obtenus avec des techniques de rendu hybrides points/polygones (voir section 2.3.3). Dans ces méthodes, les points ne sont utilisés que lorsque ceux-ci deviennent réellement plus efficaces que les triangles, c'est-à-dire lorsque la projection de ces derniers est de l'ordre de quelques pixels. Autrement dit, si localement la taille des points dans l'espace image dépasse quelques pixels, il est alors plus intéressant d'utiliser localement une représentation polygonale si celle-ci est disponible. Ainsi notre approximation d'un splat par son simple centre est toujours vérifiée pour ce type d'approche.

4.3 Extensions

4.3.1 Occlusion culling

Notre approche de *deferred splatting* permet de réaliser des tests d'occlusions efficace, très simplement. En effet, les cartes graphiques actuelles permettent de tester la visibilité de n'importe quelle primitive vis-à-vis du tampon de profondeur courant par un simple processus de rastérisation. Cette fonctionnalité est accessible en OpenGL via l'extension `ARB_occlusion_query`. En général, une requête de visibilité est réalisée sur la boîte englobante d'un groupe de primitives afin d'éventuellement les rejeter d'un seul coup. Le tampon de profondeur doit néanmoins être initialisé le plus précisement possible en traçant les objets ayant une forte chance d'être visibles. Cette sélection pose un problème difficile et requiert généralement un tri des objets du plus proche au plus éloigné (voir section 2.3.1.3).

Avec notre méthode, nous disposons déjà d'un tampon de profondeur très précis obtenu très rapidement par la première passe de splatting de visibilité. Ainsi, n'importe quelle structure de données et algorithme de sélection de haut niveau peuvent être utilisés pour réaliser des tests d'occlusion sur ce tampon de profondeur. Le choix de ces structures et algorithmes est généralement dépendant du type d'application.

En supposant une structure de données classique basée sur une hiérarchie de volumes englobants (e.g. un octree), nous proposons d'intégrer des tests d'*occlusion culling* de la manière suivante (voir figure 4.7). Tout d'abord, nous effectuons la première passe de splatting de visibilité. Notons que cette passe est entièrement réalisée par le GPU ; nous pouvons donc commencer la sélection de haut niveau en même temps. La hiérarchie de volumes englobants est alors parcourue en réalisant les classiques tests de fenêtrage et de niveaux de détails. Nous proposons d'effectuer les requêtes d'occlusions uniquement sur les noeuds sélectionnés pour deux raisons principales. La première est qu'il est généralement inutile de réaliser un test d'occlusion sur un objet ou sur une partie d'un objet dont le volume englobant représente une grande partie de l'image puisqu'il a de grandes chances d'être visible. Réaliser plusieurs petits tests est bien plus précis que d'effectuer un seul test plus important et en pratique pas plus coûteux puisque le coût d'une telle requête dépend principalement du nombre de pixels rastérisés. Deuxièmement, les requêtes d'occlusion culling sont réalisées de manière asynchrone par le GPU et peuvent même être mises en attente sans bloquer le CPU si la première passe de splatting de visibilité n'est pas finie. Il est donc astucieux de ne pas attendre le résultat d'une requête. Le noeud est alors inséré

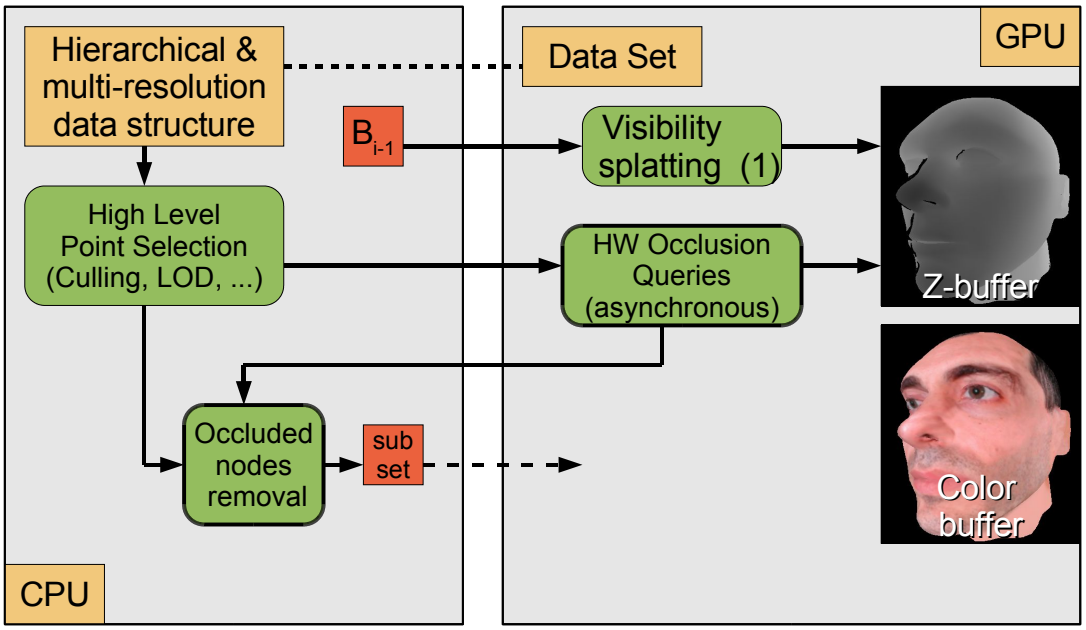


FIG. 4.7 – Apercu du fonctionnement des tests d'occlusion culling avec notre algorithme de *deferred splatting*.

dans la liste des noeuds à tracer et la sélection de haut niveau est continuée classiquement. À la fin du parcours récursif des hiérarchies de volumes englobants, la liste des noeuds potentiellement visibles est parcourue et pour chaque noeud le résultat de la requête d'occlusion est enfin récupéré permettant éventuellement de supprimer le noeud de la liste.

Le tri des indices des points visibles par objet effectué après la passe de sélection (section 4.1.2), nous permet de connaître très précisement le pourcentage de visibilité de chaque objet. Une autre optimisation possible du parcours de la structure de donnée multi-résolution, est alors d'effectuer les calculs de visibilité uniquement sur les objets ayant un pourcentage de visibilité à l'image précédente inférieur à un seuil donné.

4.3.2 Sequential Point Trees

Le *sequential point tree* (SPT) présenté dans la section 2.3.2.2 est une structure de données permettant une sélection fine des niveaux de détails qui est réalisée par le GPU. Un inconvénient de cette approche est que beaucoup trop de points sont envoyés au GPU : entre 10% et 40% suivant les cas. Utiliser le SPT avec un algorithme de splatting multi-passe n'est alors pas vraiment efficace car les coûteux *vertex shaders* utilisés pour les deux passes de splatting devront traiter plus de points qu'avec une méthode classique. Notons, que les cartes graphiques les plus récentes permettent de résoudre en parti ce problème grâce au support du branchement dynamique au niveau des *vertex shaders* : le code coûteux relatif au splatting ne sera pas exécuté pour les points superflus. Pour les cartes graphiques de la génération des GeForce5, notre technique de *deferred splatting* permet de résoudre ce problème puisque les points superflus relatifs à l'utilisation des SPT ne seront tracés que lors de notre passe extrêmement rapide de sélection (passe numéro 2).

Dans la version originale des SPT, les auteurs proposent de décomposer le SPT représentant un objet en un tableau de SPT. Chaque SPT ne contient que les points ayant la même normale

quantifiée (ils utilisent 128 normales quantifiées). L'intérêt est de permettre une élimination des faces arrières très rapide. Cependant, dans le cas de scènes complexes contenant de larges objets, nous proposons de remplacer la classification par normale par une classification spatiale permettant d'effectuer des tests de visibilité de haut niveau (fenêtrage et occlusion) beaucoup plus précis. En effet, une telle classification permet de travailler sur une hiérarchie de volumes englobants au niveau de l'objet au lieu d'un seul volume englobant par objet.

4.3.3 Approximation du *deferred splatting* 100% GPU

Sur les cartes graphiques actuelles, la lecture de données situées en mémoire vidéo n'est absolument pas optimisée. Les nouveaux bus PCI express sensés avoir un taux de transfert identique dans un sens comme dans l'autre n'améliorent en fait que légèrement les taux de transfert de la mémoires vidéo vers la mémoire centrale. La lecture du tampon d'identificateurs effectuée après la passe de sélection (passe numéro 2) prend donc un temps non négligeable (24ms pour un tampon de 1024x1024 [7]). De plus, le tri des identificateurs par objet et le calcul de la différence $B_i - B_{i-1}$ consomme du temps CPU. Sous certaines conditions et en négligeant la correction du tampon de profondeur (passe 3)[8], il est alors possible d'avoir une implémentation 100% GPU du *deferred splatting*. L'idée principale est d'utiliser directement le buffer de destination contenant les identificateurs des points visibles comme un *index buffer*. L'API OpenGL ne permet malheureusement pas encore de faire cela directement [9]. Il est néanmoins possible de simuler ce comportement via une copie du tampon de couleur vers un *vertex buffer object* (ou VBO). Un VBO est un tampon en mémoire vidéo pouvant être utilisé à la fois comme tableau d'indices ou tableau d'attributs des sommets. Cette copie est donc très rapide puisqu'elle est effectuée de la mémoire vidéo vers la mémoire vidéo. Le rendu de l'ensemble des points réellement visibles B_i est alors effectué d'un seul coup pour tous les objets via la méthode OpenGL `glDrawElements`.

Une première implication de cela est qu'il est alors indispensable que les objets soient stockés dans un unique *vertex buffer object*. L'identificateur d'un point devient simplement l'indice du point dans cet unique VBO. Chaque objet doit pourtant pouvoir avoir sa propre matrice de transformation ainsi que ses propres propriétés de matériaux. Les matrices de transformations et propriétés de matériaux seront choisies dynamiquement pour chaque point par les *vertex shader* des passes de splatting. Ces informations sont fournies au *vertex shader* par l'intermédiaire d'un tableau *uniform*. Cependant, les variables *uniform* ne sont pas stockées en mémoire vidéo mais dans des registres dont le nombre est relativement faible, par exemple seulement 96x4 registres vectoriels sont disponibles sur une GeForce5. Une matrice de transformation nécessitant 4 registres vectoriels et en comptant 4 scalaires par matériaux, le nombre d'objets dans la scène est limité à une douzaine pour une GeForce5, car quelques registres doivent être réservés pour la matrice de projection, les paramètres des sources lumineuses ainsi que les constantes utilisées pour le splatting. Les *vertex shaders* doivent également être capables de retrouver le numéro de l'objet du point traité. L'indice du point traité n'étant pas accessible à partir du vertex programme il est impossible d'en déduire son numéro d'objet. Nous proposons alors de stocker pour chaque point le numéro de l'objet auquel il appartient. Pour des raisons d'efficacité et de coût mémoire, ce numéro est stocké sous la forme d'un seul octet dans la composante alpha de la

[7] Mesure effectuée avec une GeForceFX5900 en AGP 8X sous GNU/Linux.

[8] Les impacts d'une absence de correction du tampon de profondeur ont déjà été discutés section 4.1.3.

[9] Plusieurs propositions d'extentions généralisant l'utilisation des buffers en OpenGL n'ont malheureusement jamais vu le jour.

couleur du point (en supposant qu'il n'y a pas d'objets semi-transparents).

Les indices invalides issus des pixels n'ayant pas été atteints par un point visible doivent cependant être supprimés. Pour cela, nous utilisons l'extension `GL_NV_primitive_restart` qui permet à l'utilisateur de définir une valeur d'index qui sera interprétée comme une paire `glEnd() ; glBegin(`<*current primitive*>`) ;`. Utilisée avec la primitive *point*, cette extention a pour seul effet de sauter les indices ayant cette valeur, ce qui est exactement ce que nous voulons ! La valeur de l'index utilisée pour le `GL_NV_primitive_restart` doit donc être la valeur utilisée pour vider le tampon de couleur avant la passe de sélection, c'est-à-dire : `0xFFFFFFFF` (ce choix a été discuté section 4.1.2).

4.3.3.1 Proposition d'extensions matérielles pour le deferred splatting

Comme nous venons de le voir, nos implantations du *deferred splatting*, qu'elles soient hybrides CPU/GPU ou 100% GPU, sont limitées pour des raisons techniques. Aussi, nous proposons ici une extension relativement simple qui permettrait une implantation vraiment efficace du principe du *deferred splatting*.

Le principe général serait d'introduire un mode de rendu spécial, sans mise à jour des tampons de destination mais avec écriture des indices des primitives passant les tests de visibilité dans un *index buffer*. Appelons ce mode rendu "sélection". Plutôt que de spécifier de manière précise une extension matérielle, nous allons commenter, pas à pas, un scénario typique d'utilisation :

1. Effectuer la première passe de splatting de visibilité.
2. Activer deux tampons d'indices (*index buffer object*) comme tampons de destination du mode de rendu "sélection". Ceux-ci doivent avoir le même nombre d'éléments que le nombre de pixels du tampon de profondeur. Le premier permet de stocker les indices des primitives visibles, c'est-à-dire B_i, tandis que le second permet de calculer les primitives nouvellement visibles, c'est-à-dire $B_i - B_{i-1}$. Appelons les respectivement T_0 et T_1.
3. Mettre à zéro les compteurs de primitives, nous avons un compteur par tampon d'indice. Il s'agit simplement de deux variables entières gérées par la carte graphique. Leur valeurs doivent être accessibles par l'application entre chaque opération de tracé. Appelons ces variables c_0 (associée au tampon T_0) et c_1 (associée au tampon T_1).
4. Nous allons maintenant tracer tous les objets dans les tampons d'indices en mode "sélection". Afin d'associer les éléments des tampons d'indices avec les objets nous allons utiliser deux listes L_0 et L_1 (une pour chaque tampon). Pour chaque liste, chaque élément est un triplet associant le numéro de l'objet aux premier et dernier élément du tampon d'indices respectif. Pour chaque objet k :
   (a) Récupérer les valeurs courantes des deux compteurs : (c_0^d, c_1^d).
   (b) Tracer les primitives (via des intervalles ou des listes d'indices). Cette partie est détaillée dans la suite.
   (c) Récupérer les valeurs courantes des deux compteurs : (c_0^f, c_1^f).
   (d) Ajouter (k, c_0^d, c_0^f) à la liste L_0 et (k, c_1^d, c_1^f) à la liste L_1.
5. Mettre à jour le tampon de profondeur en traçant $B_i - B_{i-1}$, c'est-à-dire simplement en activant le tampon d'indice T_1 et en parcourant la liste L_1.

6. Effectuer la passe de splatting des attributs en traçant B_i, c'est-à-dire simplement en activant le tampon d'indice T_0 et en parcourant la liste L_0.

Lorsqu'une primitive est tracée en mode "sélection", celle-ci est rastérisée le plus rapidement possible, et la profondeur de chaque fragment généré est comparée au tampon de profondeur courant. Dès qu'un fragment est visible, alors le processus de rastérisation peut être stoppé et l'indice de la primitive courante est écrit dans le tampon d'indice T_0. Le calcul de la différence $B_i - B_{i-1}$ peut paraître assez problématique. En fait, cette différence nous sert juste à corriger le tampon de profondeur courant qui est susceptible de contenir des trous. Une alternative est donc de détecter les trous lors du rendu en mode "sélection", et d'utiliser cette détection pour sélectionner les primitives que l'on doit tracer à nouveau pour corriger le tampon de profondeur. Détecter un trou est trivial : il suffit de comparer la profondeur du fragment à celle du tampon de profondeur, si celle-ci dépasse ϵ, alors nous avons un trou. L'algorithme de tracé des primitives en mode "sélection" est donc le suivant :

```
pour chaque primitive d'indice j faire
    pour chaque fragment de la primitive faire // rastérisation
        si z_fragment ≤ z_buffer alors // test du z-buffer réussi
            stopper la rastérisation de la primitive;
            T0[c0] = j;
            c0 += 1;
            si z_fragment + ε < z_buffer alors
                T1[c1] = j;
                c1 += 1;
            finsi
        finsi
```

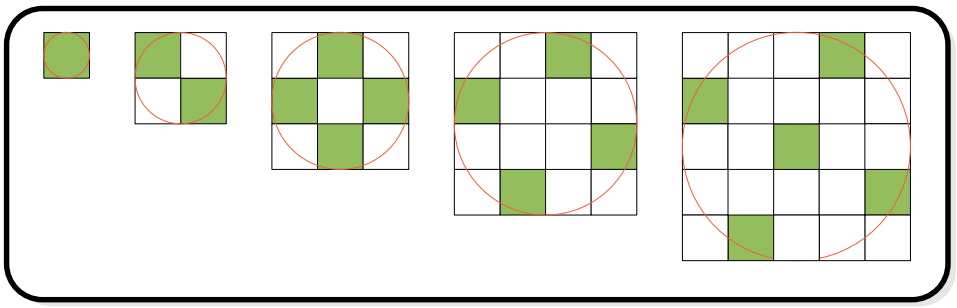


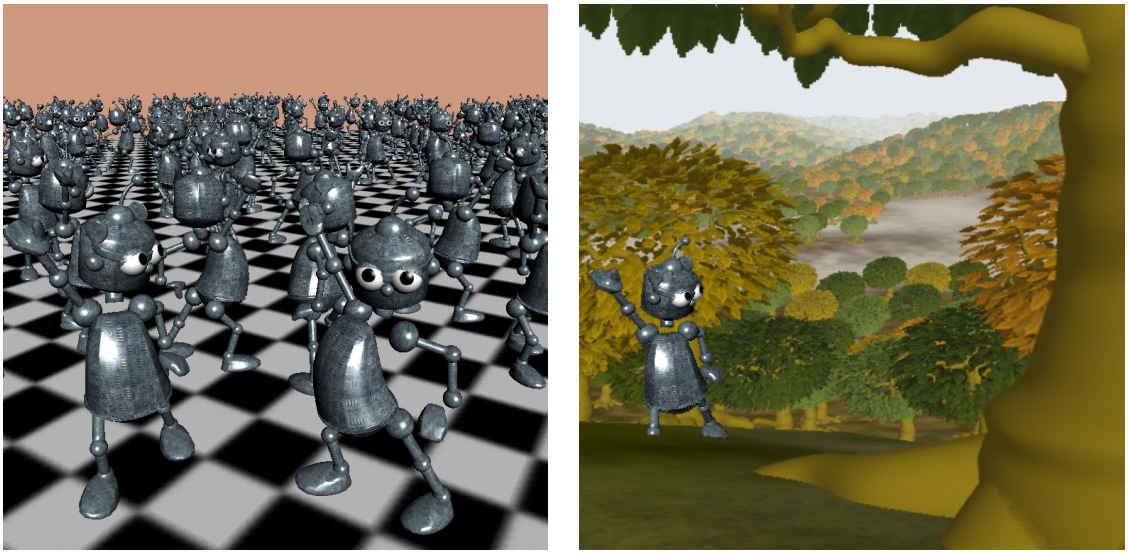
FIG. 4.8 – Rastérisation rapide via un échantillonnage par la méthode des n-reines.

Bien sûr, pour que cette sélection soit efficace, il est nécessaire que la rastérisation utilisée pour ce mode de rendu soit vraiment plus rapide que la rastérisation d'un vrai splat. De plus, afin de ne pas être limité aux petits splats seulement, nous devons tester l'ensemble de la projection d'un splat. Approcher la projection d'un splat par un carré aligné aux axes, comme le permet la primitive point standard des cartes graphiques actuelles, est suffisant et rapide, mais le nombre de fragment généré peut être excessif dans certain cas. Cependant, statistiquement,

nous n'avons pas besoins de tester l'ensemble des points, mais simplement de choisir suffisement d'échantillons soigneusement répartis au travers de l'approximation de la projection du splat (voir figure 4.8). Ces répartitions d'échantillons peuvent être précalculées pour chaque taille de point.

Une telle implantation du *deferred splatting* possèderait de nombreux avantages par rapport à nos implantations actuelles : pas de problème de calcul des identificateurs, pas de limite sur le nombre d'objets, pas de limite sur la taille des points et aucun travail pour le CPU. Techniquement, l'approche que nous venons de décrire n'apporte pas de réelle complication. Le plus problématique, par rapport aux cartes graphiques actuelles, est certainement la pseudo-rastération partielle des splats et le fait de devoir stopper cette rastérisation.

4.4 Résultats



(a) (b)

FIG. 4.9 – (a) La dance des Hugo : scène dynamique contenant 200 instances du modèle de Hugo (450k points) rendue à 33 fps. (Modèle fourni par Laurence Boissieux INRIA 2003)
(b) Un paysage avec 6800 arbres de 750k points chacun. Dans cette image 1500 arbres sont visible et 95% des points sélectionnés par les algorithmes de haut niveau sont supprimés par la sélection du deferred splatting. Dans cette scène, le sol est un maillage polygonal et illustre l'utilisation de notre algorithme avec une technique de rendu traditionnel.

Nous avons implanté notre algorithme de *deferred splatting* en utilisant les extensions standard OpenGL `ARB_vertex_program` et `ARB_fragment_program` supportées par la majorité des cartes graphiques actuelles. Afin d'obtenir des performances optimales, les objets sont stockés en mémoire vidéo en utilisant l'extension standard OpenGL `ARB_vertex_buffer_object`. Nous stockons également en mémoire vidéo un tableau d'indices consécutifs partagé par tous les objets et utilisé lors de la passe de sélection, i.e. du rendu des indices des points. Les résultats que nous présentons ici ont été obtenus avec une carte graphique NVidia GeForce FX 5900 et un processeur AMD Athlon 2GHz sous Linux.

Pour ces tests, tous nos modèles sont stockés dans un octree permettant une sélection locale des niveaux de détails. Des tests de view-frustum et occlusion culling sont réalisés au niveau des objets tout entier. Nous avons évalué notre algorithme sur des scènes de complexité différente, de la plus simple, la tête figure 4.4-c, à la plus complexe, une forêt figure 4.9-b de 6800 arbres composés de 750 000 points chacun. Cette forêt contient virtuellement plus de 5000 millions de points. En fait, seulement 4 modèles d'arbres sont explicitement stockés, les autres étant obtenus par simple instanciation. Nous avons également testé notre algorithme sur des scènes dynamiques (figure 4.9-a). Les performances de notre algorithme en images par seconde pour une résolution de 512×512 sont montrées dans la table 4.1. La première colonne expose les performances de l'algorithme d'EWA splatting sans sélection des points, la seconde colonne montre le pourcentage de points éliminés par notre sélection, la troisième colonne, les performances de notre algorithme complet.

Scène	EWA splatting (fps)	Points éliminés	EWA splatting + Deferred Splatting (fps)
Tête(285k pts)	34	70%	39
Arbre(750k pts)	8.6	88%	32
Danse des Hugos (200 x 450k pts)	11.5	90%	33.5
Forêt (6800 arbres)	0.8-1.5	90-98%	11-18

TAB. 4.1 – Performances du deferred splatting.

Même pour un modèle relativement simple comme la tête, le surcoût du deferred splatting inhérent aux passes que nous avons ajoutées est compensé par le nombre moins important de points à traiter. Lorsque les scènes deviennent très complexes, le gain devient très important puisque nous augmentons jusqu'à 10 fois les performances globales du rendu par EWA splatting.

Afin d'évaluer plus précisément le temps des différentes étapes de notre algorithme, nous présentons table 4.2 le nombre moyen de points par seconde ainsi que le temps moyen de chaque passe pour les scènes précédentes.

Passe	primitives/seconde	temps(ms)
1 - Splatting de visibilité	11.5-13.5 M/s	7-11
2 - Passe de sélection		
Rendu des indices	90-110 M/s	-
Lecture du tampon d'indices	-	6.6
Tri des indices par objet	-	1.7-2.2
3 - Splatting de visibilité (correction)	11.5-13.5 M/s	0.8-1.2
4 - Reconstruction/Accumulation	11-13 M/s	9-11.5
5 - Normalisation	-	0.7

TAB. 4.2 – Performances par passe de l'algorithme du deferred splatting.

Nous ne pouvons pas donner de temps moyen pour le rendu des indices de la passe de sélection car cela dépend grandement de la complexité de la scène (voir figure 4.10) et des algorithmes utilisés pour la sélection de haut niveau. Comme le montre le graphique 4.10, c'est cette passe de rendu qui devient la plus coûteuse. Le seul moyen de réduire ces temps de calcul est de diminuer le nombre de primitives à tester, c'est-à-dire d'améliorer la précision des algorithmes de sélection de haut niveau. Cette sélection pouvant être réalisée en parallèle de la première

passe de splatting de visibilité, le temps nécessaire à une sélection plus précise sera en partie masqué par celle-ci.

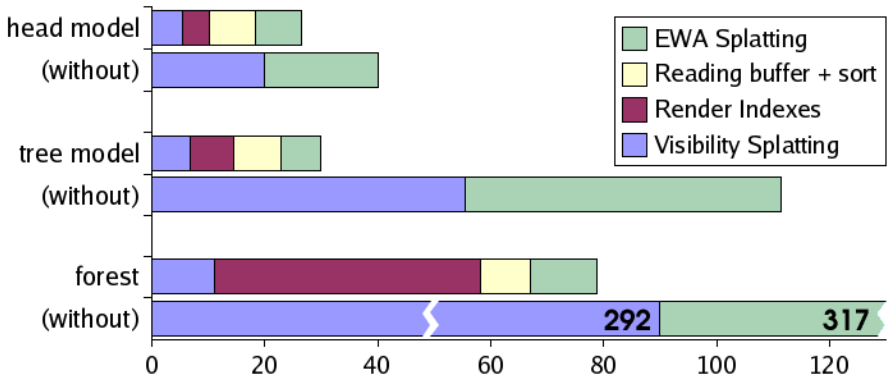


FIG. 4.10 – Temps de rendu par passe. Les passes négligeables n'apparaissent pas.

4.5 Discussions et conclusion

À propos des techniques de rendu basé images

La première passe de splatting de visibilité consiste à précalculer un tampon de profondeur en traçant uniquement les points visibles à l'image précédente. On peut alors se demander si une approche par reprojection d'image de profondeur (image warping) [BCD+99] ne permettrait pas d'obtenir un résultat similaire. En effet, étant donné les faibles différences de point de vue entre deux images successives, une méthode par reprojection d'image inverse devrait permettre une reconstruction du tampon de profondeur d'une qualité suffisante pour ne pas avoir à le corriger par une passe supplémentaire. Cependant, une telle approche aurait aussi quelques inconvénients importants. Tout d'abord, il n'est pas possible de prendre en compte une scène dynamique, seule la caméra peut bouger. De plus, le tampon de profondeur doit tout de même être recalculé de temps en temps, si ce n'est à chaque image, par une passe de splatting de visibilité, ce qui annule les éventuels gains apportés.

À propos des objets semi-transparents

L'utilisation d'objets non opaques dans une scène ne pose pas de réel problème avec notre nouvelle technique de rendu. Simplement, le *deferred splatting* ne permet aucune optimisation pour ce type d'objets. En effet, il pourrait être intéressant de tester la visibilité des points des objets semi-transparents vis à vis du z-buffer des objets opaques. Malheureusement, le deferred splatting ne permet de sélectionner qu'un seul point par pixel, ce qui pose évidement problème pour ce type d'objets puisque par définition un objet semi-transparent n'occulte pas les objets qui sont derrières et donc plusieurs surfaces différentes se projettent sur un même pixel. En revanche, le *deferred splatting* peut être utilisé par les passes de rendu des objets opaques sans affecter les autres passes effectuées pour les objets non opaques (voir section 3.3.2 pour les détails du rendu des objets semi-transparents).

Quel avenir pour le deferred splatting ?

Une implantation de l'EWA splatting en deux passes, une passe de visibilité suivie d'une passe de reconstruction, est bien sûr considérée comme un inconvénient, puisqu'un simple test de profondeur flou permettrait un splatting en une seule passe, doublant ainsi les performances. Avec le *deferred splatting* nous avons tourné cet inconvénient en un avantage puisque notre algorithme se base complètement sur le fait que le splatting est implanté en deux passes.

Maintenant, que se passerait-il si la prochaine génération de cartes graphiques permettait une implantation en une seule passe du splatting ? En fait, l'idée du *deferred splatting* serait toujours applicable via un algorithme en quatre passes :

1. Splatting des points B_{i-1}.
2. Passe de sélection, on obtient B_i.
3. Splatting des points $B_i - B_{i-1}$ (passe de correction)
4. Normalisation et *deferred shading*.

Cependant, la principale accélération du *deferred splatting* est due au fait que tracer un point sous la forme d'un seul pixel est actuellement considérablement plus rapide que le rendu complet d'un splat. Si une carte graphique venait à supporter nativement le rendu et mélange de splats, les différences deviendraient sans doute assez minimes et l'accélération par *deferred splatting* ne serait notable que pour des scènes très complexes. D'un autre côté, nous avons vu qu'une approche par *deferred splatting* apporte d'autres avantages permettant d'optimiser les algorithmes de sélection de haut niveau. En particulier, nous sommes capable d'obtenir très rapidement, et avant même le processus de sélection de haut niveau, un tampon de profondeur presque parfait. Ceci est extrêmement important puisque ce tampon de profondeur est très utilile pour effectuer des tests d'occlusion culling de haut niveau très efficace. Aussi, nous pensons que le gain principale du *deferred splatting* pour les scènes complexes ne viendra pas de la sélection du *deferred splatting* elle même, mais plutôt des tests d'occlusion culling efficaces qu'il permet de mettre en place.

Conclusion

Grâce à une sélection de bas niveau des primitives visibles et de la prise en compte de la cohérence spatio-temporelle, notre méthode de *deferred splatting* permet de réduire les calculs coûteux du splatting aux seuls points visibles. Le nombre de points visibles pour un point de vue donné étant borné par la taille de l'image, les temps de rendu des passes de splatting de visibilité et de reconstruction deviennent ainsi indépendant de la complexité de la scène. Bien que suffisement générique pour être utilisée dans tous types d'applications et avec tous types de structures de données de haut niveau, nous avons vu que notre technique est capable de tirer profit des méthodes de sélection de haut niveau et inversement. Par exemple, nous avons montré que notre approche permet de réaliser très simplement et efficacement des tests d'occlusion culling de haut niveau et permet aussi d'accélérer le parcours hiérarchique de la scène. Inversement la complexité de notre passe de sélection est linéaire avec le nombre de points sélectionnés par les algorithmes de haut niveau. Deferred splatting et algorithme de sélection de haut niveau ne sont donc pas en compétition, mais au contraire sont très complémentaires.

Chapitre 5

Le raffinement des géométries basées points [10]

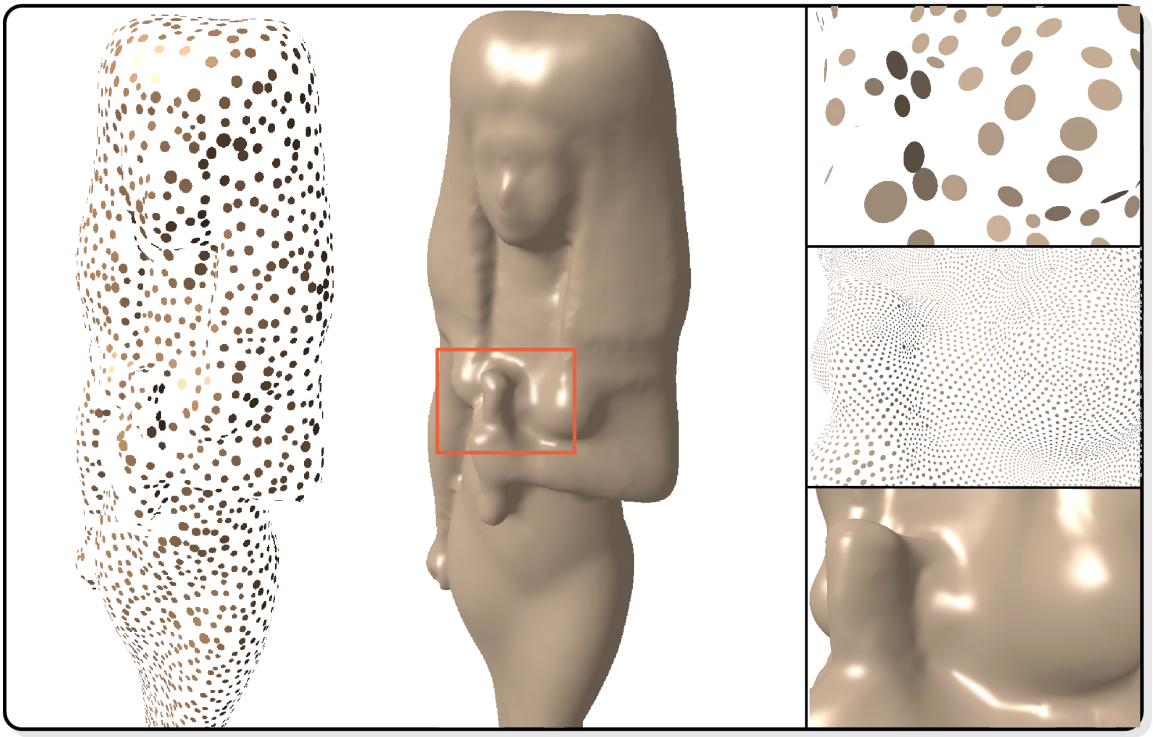


FIG. 5.1 – Illustration de notre raffinement $\sqrt{3}$ sur le modèle d'Isis irrégulièrement échantillonné par 3500 points (gauche). La colonne de gauche montre un détail sur une zone particulièrement sous-échantillonnée.

[10] Ces travaux ont été publiés dans : [GBP04c, GBP04b, GBP05].

Introduction

Comme nous l'avons déjà vu tout au long de ce mémoire, l'EWA splatting permet un très bon compromis qualité/vitesse du rendu. En cas de réduction, la qualité est assurée par un filtrage anisotrope performant. Théoriquement, la définition de surface utilisée par l'opération de splatting est censée reconstruire une fonction continue pour chaque attribut. Une fois appliquée au splatting, la paramétrisation globale utilisée pour mélanger les splats est dépendante du point de vue. En cas de large agrandissement, la qualité de la reconstruction se dégrade rapidement au niveau des régions où le morphisme 2D/2D définissant le passage des paramétrisations locales des splats à la paramétrisation globale subit les plus fortes distorsions. Ces régions correspondent principalement aux silhouettes de l'objet où la géométrie, c'est-à-dire les disques orientés se chevauchant, devient visible (voir figure 5.2).

Pour résoudre ce problème majeur, nous proposons d'augmenter dynamiquement, au moment du rendu, la densité du nuage de points là où c'est nécessaire. D'un point de vue théorique, en plus d'augmenter la densité de l'échantillonnage un tel algorithme de raffinement devrait idéalement fournir les fonctionnalités suivantes :

- Régulariser l'échantillonnage. La qualité du splatting est également sensible à la régularité (locale) de la répartition des points.
- Converger vers une surface lisse (G^1 continue : continuité visuelle des normales).
- Assurer le remplissage de larges trous dans la géométrie.
- Assurer la prise en compte des bords et des arêtes.
- Être multi-résolution.

D'un point de vue pratique, celui-ci devrait en plus :

- Être rapide, afin de permettre son utilisation au sein d'un *pipe-line* de rendu temps-réel.
- Ne pas nécessiter une longue phase de précalcul (inférieure à quelques milli-secondes) afin de ne pas perdre la flexibilité du splatting.
- Permettre le raffinement local, seules les régions sous-échantillonnées par rapport au point de vue courant doivent être raffinées.

Satisfaire toutes ces conditions en même temps n'est pas simple. Dans le cadre des représentations par maillage polygonal, les surfaces de subdivision permettent globalement d'assurer tous ces besoins. Nous nous sommes donc logiquement orientés vers une approche de raffinement de nuages de points s'inspirant des principes des surfaces de subdivision. Avec une telle approche nous pouvons en plus espérer viser d'autres types d'applications que le rendu de haute qualité, comme par exemple la modélisation multi-résolution, domaine où les surfaces de subdivision excellent.

Ce chapitre est organisé de la façon suivante. Après un aperçu de notre algorithme de raffinement itératif, nous présenterons notre approche de reconstruction locale de surface, une nouvelle méthode de calcul de voisinage, les différentes stratégies de raffinement que nous proposons, le mécanisme de contrôle de l'échantillonnage et la prise en compte des bords et arêtes franches. Ensuite, nous verrons le côté implantation de notre algorithme de rendu, c'est-à-dire les structures de données mises en jeux pour la recherche rapide des voisins et l'intégration au sein de notre *pipe-line* de rendu.

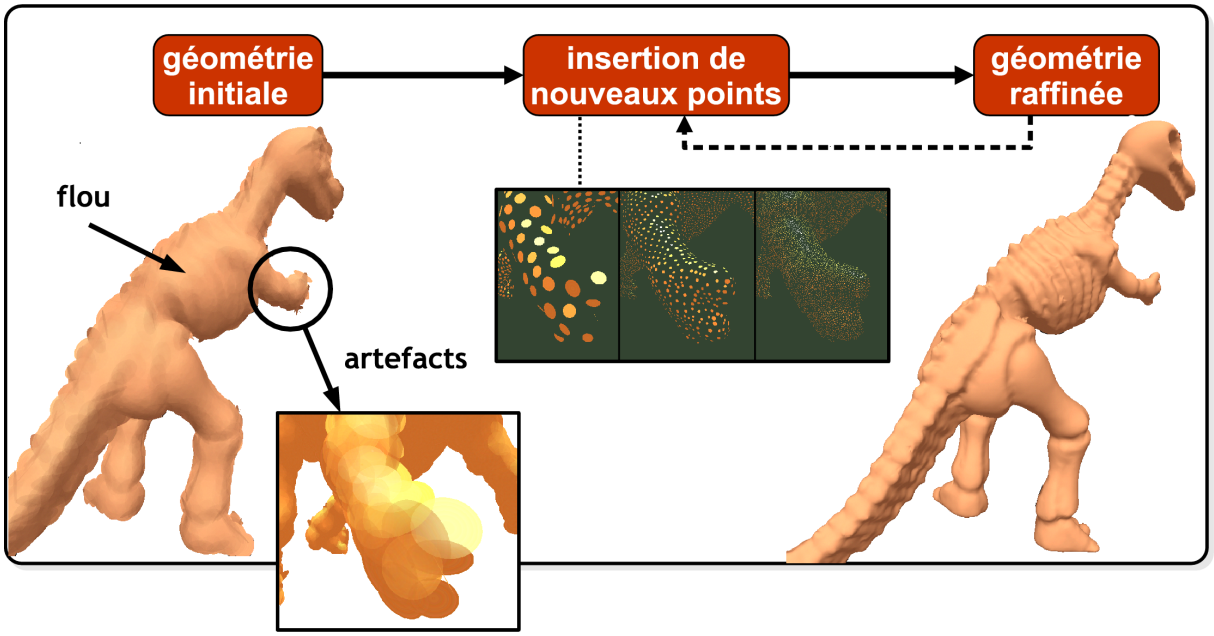


FIG. 5.2 – Illustration de notre approche par raffinement itératif.

5.1 Aperçu de l'algorithme de raffinement

Notre algorithme de raffinement prend en entrée un nuage de points $P^0 = \{\mathbf{p}_i\}$ définissant une surface lisse et *manifold* où chaque point $\mathbf{p}_i \in P^0$ est équipé d'une normale $\mathbf{n}_i$ et d'un rayon r_i. Le rayon d'un point $\mathbf{p}_i$ doit être choisi de sorte que le plus loin des voisins de $\mathbf{p}_i$ soit à une distance inférieure à r_i.

En nous inspirant des surfaces de subdivisions, nous proposons une approche de raffinement itérative générant une séquence $P^0, P^1, \ldots, P^l, \ldots$ de nuages de points de plus en plus denses. Puisque nous voulons interpoler les points, nous avons : $P^l \subset P^{l+1}$ (bien que le rayon des points diminue à chaque pas de raffinement). Le nuage de points raffiné P^{l+1} est donc l'union de l'ensemble P^l et de l'ensemble des points résultant du raffinement local de chaque point $\mathbf{p} \in P^l$.

Le raffinement local d'un point $\mathbf{p} \in P^l$ se déroule en plusieurs étapes :

- La première étape consiste à sélectionner parmi les points de P^l un voisinage N_p formant un premier anneau autour de $\mathbf{p}$. Ce voisinage forme implicitement un *triangle fan* autour du point $\mathbf{p}$. Il est important de bien remarquer que ces *triangles* ne sont construits que localement, autour du point courant et sont *oubliés* immédiatement après le raffinement du point. De plus, l'union de ces *triangles fan* ne forme pas un maillage global consistant.
- Un ensemble L_p de candidats pour l'insertion des nouveaux points est extrait du *triangle fan* précédent. Nous proposons dans la section 5.4 deux stratégies de raffinement : un raffinement diadique dont le principe est d'insérer un point pour chaque paire de voisins et un raffinement $\sqrt{3}$ insérant un point au centre de chaque *triangle*. Les points issus des règles de raffinement sont ensuite lissés grâce à des opérateurs de lissage ϕ_k, défini à partir de k points. Ces opérateurs sont présenté dans la section 5.2.1. Ce sont les points lissés qui forment l'ensemble des candidats L_p.

- La dernière étape consiste à sélectionner parmi L_p les points à insérer autour de $\mathbf{p}$, afin d'augmenter la densité de points tout en optimisant l'uniformité de l'échantillonnage (dans les zones suffisement denses, aucun point n'est inséré).

5.2 Reconstruction locale de surface par cubiques de Bézier

Afin de permettre la génération d'une surface lisse, nous devons disposer d'un mécanisme de lissage des points insérés. En d'autres termes, les points définis par la stratégie de raffinement doivent être déplacés de manière à former une surface lisse. Comme nous le verrons par la suite, les positions des points fournies par les stratégies de raffinement correspondent aux centres de gravité de groupes de points de cardinalité variable et formant des polygones convexes.

Aussi, l'approche que nous avons choisie est de projeter les points sur un morceau de surface lisse interpolant les positions et normales des points du groupe à partir duquel ils ont été créés. Un premier avantage de cette approche est de garantir la localité des calculs.

Pour la reconstruction de morceaux de surface lisse interpolant positions et normales, nous avons opté pour l'utilisation de cubiques de Bézier, très rapide à construire. En fonction du nombre de points-normales à interpoler, nous proposons différents opérateurs de lissage ϕ définis par la construction d'une courbe de Bézier, d'un triangle de Bézier ou d'un carreau de Bézier.

Cette reconstruction n'est utilisée que pour interpoler les attributs géométriques de la surface, c'est-à-dire positions et normales. Pour les autres attributs tels que la texture, nous nous contentons d'une interpolation linéaire.

Après avoir présenté ces opérateurs locaux de lissage, nous présenterons aussi quelques outils de mesure basés sur cette reconstruction locale.

5.2.1 Les opérateurs de lissage

5.2.1.1 Interpoler entre deux point-normales

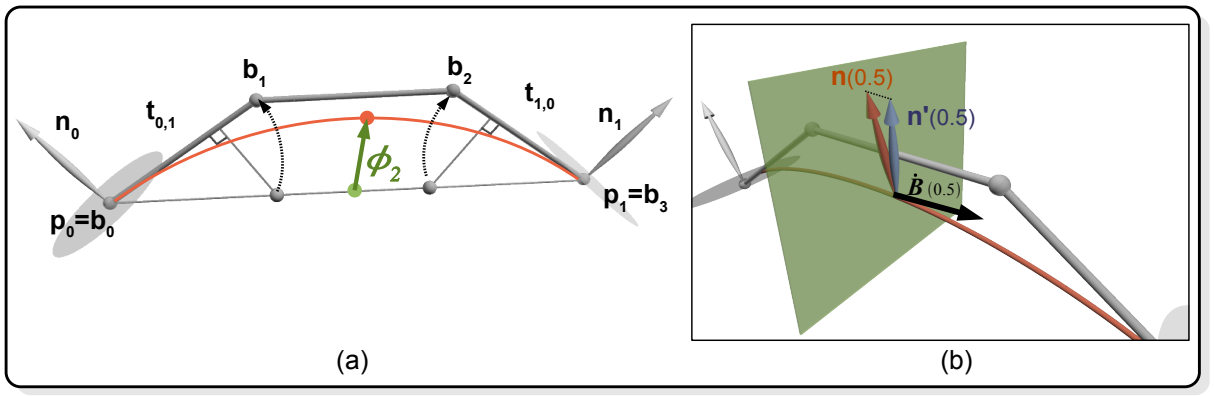


FIG. 5.3 – (a) Construction d'une courbe de Bézier interpolant deux points orientés. (b) Interpolation des normales.

Nous effectuons l'interpolation entre deux points orientés $\mathbf{p}_0$ et $\mathbf{p}_1$ par la construction d'une courbe cubique de Bézier $B(u)$ définie par les quatre points de contrôles $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$ et $\mathbf{b}_3$. Les deux extrémités $\mathbf{b}_0$ et $\mathbf{b}_3$ de la courbe sont respectivement $\mathbf{p}_0$ et $\mathbf{p}_1$. Les deux points $\mathbf{b}_1$ et $\mathbf{b}_2$

contrôlant les tangentes de la courbe doivent être pris dans les plans tangents respectifs des points $\mathbf{p}_0$ et $\mathbf{p}_1$. Étant donné qu'il y a une infinité de solutions, nous devons en choisir une d'une forme raisonnable et relativement simple à calculer. L'idée est de prendre $\mathbf{b}_1$ sur la demi-droite allant du point $\mathbf{p}_0$ vers la projection de $\mathbf{p}_1$ sur le plan tangent de $\mathbf{p}_0$, de sorte que la distance entre $\mathbf{p}_0$ et $\mathbf{b}_1$ soit égale au tiers de la distance entre les deux extrémités $\mathbf{p}_0$ et $\mathbf{p}_1$. Le point $\mathbf{b}_1$ est obtenu de façon symétrique. Soit $Q_i(\mathbf{x})$ l'opérateur projetant orthogonalement le point $\mathbf{x}$ sur le plan tangent du point $\mathbf{p}_i$:

$$Q_i(\mathbf{x}) = \mathbf{x} + (\mathbf{p}_i - \mathbf{x}) \cdot \mathbf{n}_i * \mathbf{n}_i \tag{5.1}$$

$$\tag{5.2}$$

Nous définissons aussi, d'une manière générale, le pseudo-vecteur tangent $\mathbf{t}_{ij}$, allant du point $\mathbf{p}_i$ vers le point $\mathbf{p}_j$, de la manière suivante :

$$\mathbf{t}_{i,j} = \frac{\|\mathbf{p}_j - \mathbf{p}_i\|}{3} \frac{Q_i(\mathbf{p}_j) - \mathbf{p}_i}{\|Q_i(\mathbf{p}_j) - \mathbf{p}_i\|} \tag{5.3}$$

Les points de contrôle $\mathbf{b}_1$ et $\mathbf{b}_2$ sont alors simplement donnés par :

$$\mathbf{b}_1 = p_0 + \mathbf{t}_{0,1}$$
$$\mathbf{b}_2 = p_1 + \mathbf{t}_{1,0}$$

Opérateur de lissage Insérer un point entre deux points orientés revient donc à insérer un point au milieu de la courbe de Bézier précédemment construite (figure 5.3-a). L'opérateur de lissage ϕ_2 correspond donc au déplacement du milieu du segment $[\mathbf{p}_0\mathbf{p}_1]$ sur le milieu de la courbe de Bézier, c'est-à-dire aux trois-huitièmes de la somme des pseudo-vecteurs tangents :

$$\phi_2(\mathbf{p}_0, \mathbf{p}_1) = \frac{3}{8} (\mathbf{t}_{0,1} + \mathbf{t}_{1,0}) \tag{5.4}$$

Lors de l'insertion d'un point sur cette courbe, nous devons également être capable de calculer sa normale. Cependant, un point sur une courbe n'a pas de normale. Nous savons uniquement que sa normale doit être orthogonale au vecteur tangent de la courbe en ce point et que la normale d'un point sur la courbe doit varier de manière continue de $\mathbf{n}_0$ à $\mathbf{n}_1$. Ainsi, une solution raisonnable est d'interpoler linéairement les normales des deux extrémités et de prendre le vecteur orthogonal au vecteur tangent le plus proche, c'est-à-dire, la projection de la normale interpolée sur le plan de normale $\dot{B}(u)$ (figure 5.3-b) :

$$\begin{aligned} \mathbf{n}'(u) &= \mathbf{n}_0 * (1-u) + \mathbf{n}_1 * u \\ \mathbf{n}(u) &= \mathbf{n}'(u) - \mathbf{n}'(u) \cdot \dot{B}(u) * \dot{B}(u) \end{aligned} \tag{5.5}$$

La normale $\mathbf{n}_{new}$ du point inséré est donc :

$$\mathbf{n}_{new} = \frac{\mathbf{n}(0.5)}{\|\mathbf{n}(0.5)\|} \tag{5.6}$$

5.2.1.2 Interpoler entre 3 point-normales

À partir de trois points orientés, nous pouvons construire un carreau triangulaire de Bézier cubique $B(u, v)$ interpolant les trois points et leurs normales.

$$B(u,v) = \sum_{i+j+k=3} \mathbf{b}_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k \,,\ w = 1-u-v \tag{5.7}$$

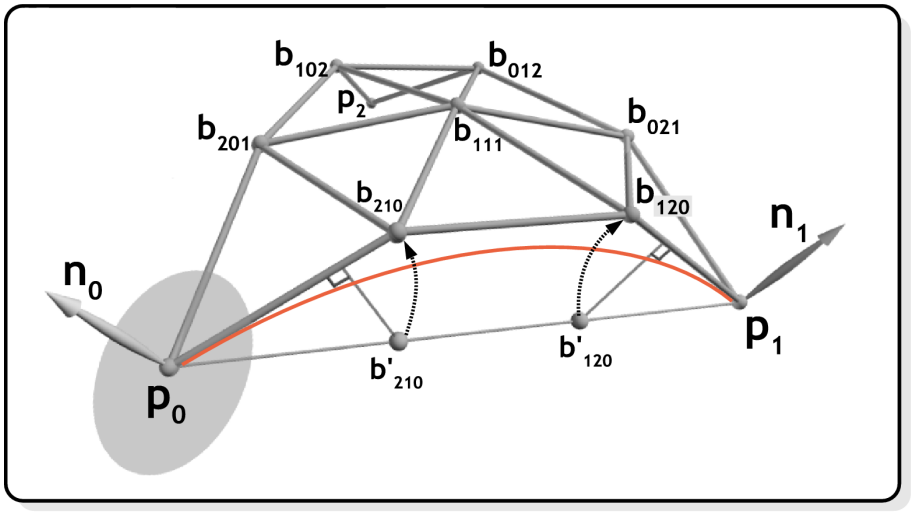


FIG. 5.4 – Construction d'un triangle de Bézier entre trois points orientés. Les côtés du triangle correspondent à des courbes de Bézier interpolant deux points orientés.

La détermination des neuf points de contrôle $\mathbf{b}_{ijk}$ est très similaire à la construction précédente :

- Les trois coins $\mathbf{b}_{300}$, $\mathbf{b}_{030}$, $\mathbf{b}_{003}$ sont respectivement $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$.
- Les positions des six points de contrôle à la bordure du carreau ($\mathbf{b}_{ijk}$, $i+j+k=3$, $i \neq j \neq k$) dépendent uniquement de leur bordure respective et sont calculés exactement comme précédemment. Par exemple :

$$\begin{aligned} \mathbf{b}_{210} &= \mathbf{p}_0 + \frac{\|\mathbf{p}_0\mathbf{p}_1\|}{3} * \frac{Q_0(\mathbf{p}_1) - \mathbf{p}_0}{\|Q_0(\mathbf{p}_1) - \mathbf{p}_0\|} \\ &= \mathbf{p}_0 + \mathbf{t}_{0,1} \end{aligned}$$

- Le point de contrôle central est choisi pour reproduire des polynômes quadratiques :

$$b_{111} = \mathbf{c} + \frac{3}{2}(\mathbf{a} - \mathbf{c}) \tag{5.8}$$

où $\mathbf{c}$ est le centre de gravité des trois extrémités et $\mathbf{a}$ est la moyenne des six points de contrôle à la bordure du carreau.

$$\mathbf{c} = \frac{1}{3}(\mathbf{b}_{300} + \mathbf{b}_{030} + \mathbf{b}_{003})$$

$$\mathbf{a} = \frac{1}{6}(\mathbf{b}_{210} + \mathbf{b}_{120} + \mathbf{b}_{021} + \mathbf{b}_{012} + \mathbf{b}_{102} + \mathbf{b}_{201})$$

Notons que cette construction est très proche de celle utilisée par Vlachos et al. dans leur technique de rendu de maillages triangulaire par PN-triangles (Point-Normal Triangle) [VPBM01]. Leur construction varie simplement pour le calcul des six points de contrôles à la bordure du carreau. Par exemple, dans leur cas, le point $\mathbf{b}_{210}$ est obtenu par la projection orthogonale du tiers de $\mathbf{p}_0\mathbf{p}_1$ sur le plan tangent de $\mathbf{p}_0$. Dans notre cas, cette projection est en plus déplacée de manière à ce que la longueur $\|\mathbf{b}_{210} - \mathbf{p}_0\|$ soit égale au tiers de la distance entre $\mathbf{p}_0$ et $\mathbf{p}_1$. Cette remise à l'échelle a comme avantage d'éviter l'introduction de zones plates dans les régions de forte courbure.

Opérateur de lissage Nous définissons l'opérateur de lissage ϕ_3 comme le déplacement du centre de gravité des trois points initiaux sur le centre du triangle de Bézier interpolant ces trois points orientés. La position finale du point est donc $B(\frac{1}{3}, \frac{1}{3})$ et l'opérateur de lissage ϕ_3 peut être exprimé comme la moyenne des six pseudo-vecteurs tangents :

$$\phi_3(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2) = \frac{1}{6} \sum_{i=0}^{2} \mathbf{t}_{i,i+1} + \mathbf{t}_{i,i+2} \tag{5.9}$$

Bien sûr, les indices $i+1$ et $i+2$ doivent être pris modulo 3.

La normale du point ainsi inséré est donc simplement la normale du centre du carreau triangulaire de Bézier et est obtenue par le produit vectoriel des deux vecteurs tangents :

$$\begin{aligned} B^u(\frac{1}{3}, \frac{1}{3}) &= 7(\mathbf{p}_1 - \mathbf{p}_0) + \mathbf{b}_{120} - \mathbf{b}_{102} + \mathbf{b}_{012} - \mathbf{b}_{210} + 2(\mathbf{b}_{021} - \mathbf{b}_{201}) \\ B^v(\frac{1}{3}, \frac{1}{3}) &= 7(\mathbf{p}_2 - \mathbf{p}_0) + \mathbf{b}_{102} - \mathbf{b}_{120} + \mathbf{b}_{021} - \mathbf{b}_{201} + 2(\mathbf{b}_{012} - \mathbf{b}_{210}) \\ \mathbf{n}_{new} &= \frac{B^u(\frac{1}{3}, \frac{1}{3}) \wedge B^v(\frac{1}{3}, \frac{1}{3})}{\|B^u(\frac{1}{3}, \frac{1}{3}) \wedge B^v(\frac{1}{3}, \frac{1}{3})\|} \end{aligned} \tag{5.10}$$

5.2.1.3 Interpoler entre 4 point-normales

Nous reconstruisons un morceau de surface interpolant quatre points orientés en construisant un carreau de Bézier bi-cubique (figure 5.5-a). Un tel carreau est défini par seize points de contrôle calculés comme suit :

- Les quatre coins $\mathbf{b}_{00}$, $\mathbf{b}_{03}$, $\mathbf{b}_{33}$, $\mathbf{b}_{30}$ sont respectivement $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$.
- Les positions des huit points de contrôle aux bordures du carreau sont construits comme précédemment.
- Les quatre points de Bézier internes sont déterminés de manière à générer des vecteurs twists nuls. Par exemple, pour le point de contrôle b_{11} nous avons :

$$\begin{aligned} b_{00} &= p_0 \\ b_{01} &= p_0 + \mathbf{t}_{0,1} \\ b_{10} &= p_0 + \mathbf{t}_{0,3} \\ b_{11} &= p_0 + \mathbf{t}_{0,1} + \mathbf{t}_{0,3} \end{aligned}$$

Opérateur de lissage La solution des twists nuls permet de simplifier le calcul de la position du centre du carreau $B(0.5, 0.5)$ tout en évitant de vriller la surface interne des carreaux. Finalement nous avons pour l'opérateur de lissage :

$$\phi_4(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = \frac{3}{16} \sum_{i=0}^{3} \mathbf{t}_{i,i+1} + \mathbf{t}_{i,i+3} \tag{5.11}$$

La normale du point lissé est simplement obtenue par le produit vectoriel des deux vecteurs tangents :

$$\mathbf{n}_{new} = \frac{B^u(0.5, 0.5) \wedge B^v(0.5, 0.5)}{\|B^u(0.5, 0.5) \wedge B^v(0.5, 0.5)\|}$$

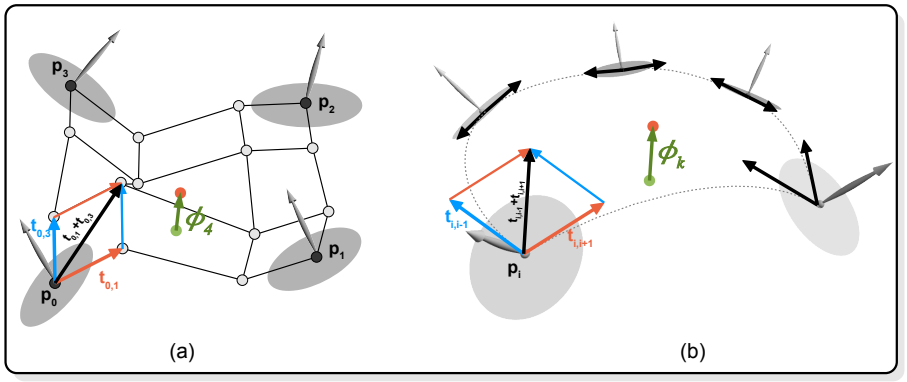


FIG. 5.5 – (a) Construction d'un carreau de Bézier interpolant quatre points orientés. (b) Opérateur de lissage généralisé.

5.2.1.4 Interpoler entre $k \geq 5$ point-normales

Nous pouvons remarquer que les trois opérateurs de lissage précédents sont tous définis comme une somme pondérée des vecteurs tangents définis le long de la bordure du polygone de contrôle. Cependant, la somme des poids n'est pas toujours égale à 1 et dépend indirectement du nombre de sommets du polygone. En effet, considérons le cas de polygones réguliers, plus le nombre de sommets est grand et plus l'angle entre deux côtés consécutifs est important. Ainsi, la norme de la somme des deux vecteurs tangents partant d'un sommet diminue et doit donc en quelque sorte être *re-normalisée*. Nous définissons le facteur de normalisation β_k suivant :

$$\beta_k = \frac{3}{4cos^2\left(\pi\frac{k-2}{2k}\right)} \tag{5.12}$$

La valeur de β_k ainsi définie est inversement proportionnelle à l'angle entre deux côtés consécutifs et est en accord avec les opérateurs de lissage précédents. Nous pouvons donc définir notre opérateur de lissage généralisé de la façon suivante (figure 5.5-b) :

$$\phi_k(\mathbf{p}_0, ..., \mathbf{p}_k) = \frac{\beta_k}{2k}\sum_{i=1}^{k} \mathbf{t}_{k,k+1} + \mathbf{t}_{k,k-1} \tag{5.13}$$

Reste le difficile problème du calcul de la normale du point inséré. Remarquons que le polygone de contrôle forme autour du point inséré un *triangle fan*. Aussi, un choix raisonnable pour cette normale est de prendre la moyenne des normales des k faces triangulaire de ce *triangle fan*.

5.2.2 Outils pour l'analyse locale de la surface

5.2.2.1 Distance géodésique locale

Lors de l'analyse d'un voisinage, il est souvent utile de mesurer les distances relatives entre les différents voisins potentiels. Cependant, lorsqu'il s'agit de points sur une surface, la simple distance euclidienne n'est généralement pas suffisante. Conformément à notre méthode de reconstruction locale de surface, nous définissons la distance géodésique locale $\widetilde{G}(\mathbf{p}_0, \mathbf{p}_1)$ entre deux points orientés relativement proches $\mathbf{p}_0$, $\mathbf{p}_1$ comme étant la longueur de la cubique de Bézier interpolant ces deux points.

D'un point de vue pratique, le calcul exact de la longueur d'une telle courbe est beaucoup trop coûteux. Dans notre cadre d'utilisation, une approximation suffisante est de prendre la longueur du polygone de contrôle :

$$\widetilde{G}(\mathbf{p}_0, \mathbf{p}_1) = \frac{2}{3}\|\mathbf{p}_0\mathbf{p}_1\| + \|b_1 b_2\| \tag{5.14}$$

5.2.2.2 "Angle-courbe"

Mesurer l'angle entre deux points $\mathbf{p}_0$, $\mathbf{p}_1$ par rapport à un troisième point $\mathbf{p}$ est particulièrement utile pour analyser le voisinage du point $\mathbf{p}$. De la même manière que pour la distance euclidienne, l'angle géométrique classique n'est pas assez précis lorsque les points sont liés à une surface. Nous définissons donc "l'angle-courbe" $\widetilde{A}_{\mathbf{p}}(\mathbf{p}_0, \mathbf{p}_1)$ comme étant "l'angle le long de la géodésique allant du point $\mathbf{p}_0$ au point $\mathbf{p}_1$" (voir figure 5.6). Mathématiquement, cette angle est défini par l'intégrale suivante :

$$\widetilde{A}_{\mathbf{p}}(\mathbf{p}_0, \mathbf{p}_1) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \widehat{B(\frac{i}{n})\mathbf{p}B(\frac{i+1}{n})} \tag{5.15}$$

où $B(u)$ représentante la géodésique allant du point $\mathbf{p}_0$ au point $\mathbf{p}_1$. Dans notre cas il s'agit donc d'une cubique de Bézier construite comme précédemment. Comme pour l'approximation de notre distance géodésique locale, nous approchons "l'angle-courbe" ainsi défini par la somme des trois angles formés par le polygone de contrôle de la courbe interpolant $\mathbf{p}_0$ et $\mathbf{p}_1$ et le point $\mathbf{p}$:

$$\widetilde{A}_{\mathbf{p}}(\mathbf{p}_0, \mathbf{p}_1) = \widehat{\mathbf{p}_0\mathbf{p}b_1} + \widehat{b_1\mathbf{p}b_2} + \widehat{b_2\mathbf{p}\mathbf{p}_1} \tag{5.16}$$

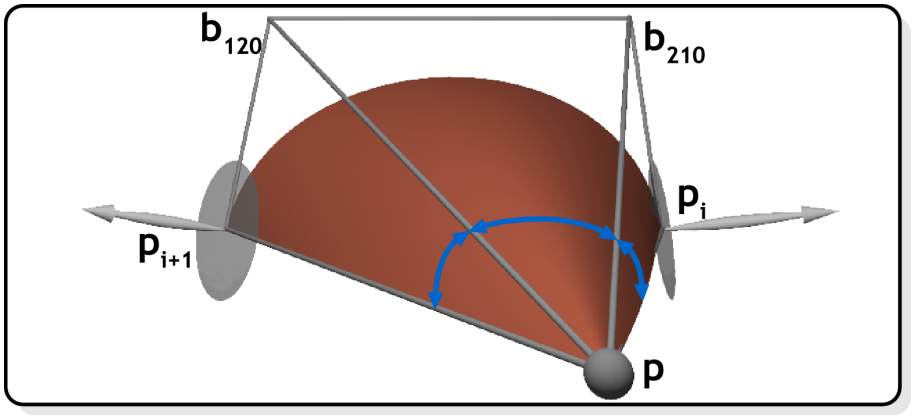


FIG. 5.6 – Définition d'un "angle courbe" entre deux points orientés $\mathbf{p}_i$, $\mathbf{p}_{i+1}$ relativement à un troisième point $\mathbf{p}$.

5.2.2.3 Conditions nécessaires / Limites de la construction

Notre construction des points de contrôle d'une cubique interpolante par projection sur les plans tangents n'est cependant pas toujours consistante. En effet, comme illustré sur la figure 5.7, certaines configurations des positions et normales respectives des deux extrémités produisent une inconsistance de la reconstruction vis-à-vis de l'orientation des normales (intérieur/extérieur).

Cette situation apparaît lorsque le point $\mathbf{p}_1$ est à l'intérieur du cône infini d'apex $\mathbf{p}_0$ et d'axe $\mathbf{n}_0 + \mathbf{n}_1$ (figure 5.7).

Dans ce cas, un traitement spécifique pourrait être appliqué afin de rétablir la cohérence des normales. Cependant, plusieurs possibilités sont envisageables. Cela signifie que la surface peut soit faire une vrille sur elle-même, soit faire une large vague (un des vecteurs tangents doit être inversé). Déterminer la solution la plus raisonnable parmi les trois possibilités, nécessiterait une analyse globale du nuage de points. De plus, cela signifierait que nous essayons de reconstruire une surface très largement sous-échantillonnée[11]. Finalement, une dernière solution plus raisonnable, est de considérer que les deux points $\mathbf{p}_0$ et $\mathbf{p}_1$ ne sont pas voisins. Ainsi, nous pouvons établir la condition nécessaire pour que deux points soient voisins de la manière suivante :

$$C_{cne}(\mathbf{p}_0, \mathbf{p}_1) \Leftrightarrow \left| (\mathbf{n}_0 + \mathbf{n}_1) \cdot \frac{\mathbf{p}_0\mathbf{p}_1}{\|\mathbf{p}_0\mathbf{p}_1\|} \right| > 1 + \mathbf{n}_0 \cdot \mathbf{n}_1 \tag{5.17}$$

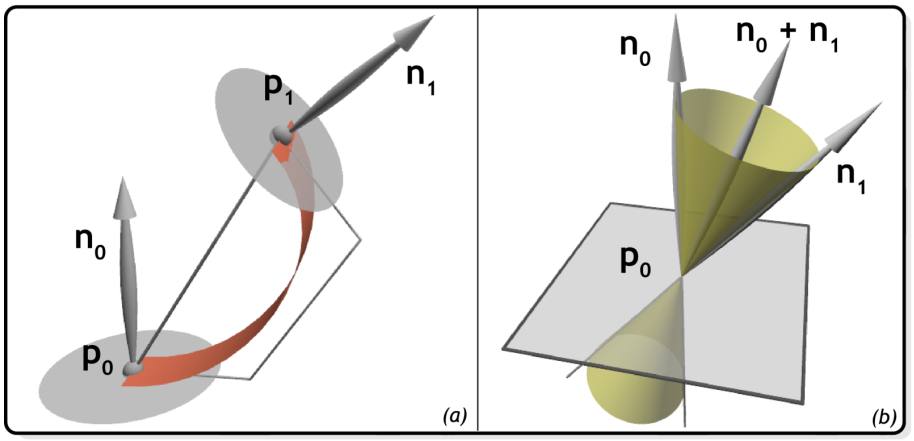


FIG. 5.7 – Limites de la construction des courbes de Bézier. (a) Les positions et orientations relatives des points $\mathbf{p}_0$ et $\mathbf{p}_1$ sont telles que la construction des vecteurs tangents par projection sur les plans tangents est inconsistante. (b) En fixant la position de $\mathbf{p}_0$ et les deux normales $\mathbf{n}_0$ et $\mathbf{n}_1$, le point $\mathbf{p}_1$ doit être à l'extérieur du cône jaune.

5.3 Sélection d'un premier anneau de voisinage

La première étape dans le raffinement d'un point consiste à sélectionner les points formant un premier anneau de voisinage qui soit le plus pertinent possible. Cette étape est l'étape-clé de notre algorithme, et de la qualité de cette sélection dépend grandement la robustesse du raffinement puisque c'est à partir de ce voisinage que seront insérés les nouveaux points.

La plupart des algorithmes travaillant sur des nuages de points sont basés sur la détermination d'un voisinage et de nombreuses méthodes ont déjà été proposées. Cependant, aucune n'est suffisement robuste pour notre cas.

[11] Par exemple, le dernier cas signifierait que P^0 est un r-sampling de la surface S avec $r > 2$, voir [ABK98] pour la définition d'un r-sampling.

5.3.1 Les méthodes existantes

Voisinage euclidien Le voisinage euclidien N_p^ϵ est l'ensemble des points contenu dans une boule de centre $\mathbf{p}$ et de rayon ϵ :

$$N_p^\epsilon = \{\mathbf{x} \in P \mid \|\mathbf{p} - \mathbf{x}\| < \epsilon\} \tag{5.18}$$

Le problème d'un tel voisinage réside bien sûr dans le choix du ϵ.

Les k plus proches voisins Comme son nom l'indique, la méthode des k plus proches voisins consiste à prendre les k points dans P les plus proches du point $\mathbf{p}$. Ainsi, N_p^k est l'ensemble des k points $\{p_0, \ldots, p_{k-1}\} \subset P$ tel que :

$$\forall p_j \in P - N_p^k et p_i \in N_p^k, \|p - p_j\| > \|p - p_i\| \tag{5.19}$$

Cette méthode, très largement utilisée, a l'avantage sur la définition précédente de s'adapter à la densité locale du nuage de points P. Cependant, elle ne permet absolument pas de sélectionner un voisinage cohérent dès que la répartition des points n'est pas suffisement régulière. Une analyse des limites d'un tel voisinage peut être trouvée dans [AGPS04].

Dans [LP02b], Linsen propose une extension de la méthode des k plus proches voisins consistant à remplacer le plus éloigné des voisins sélectionnés par le plus proche voisin suivant tant qu'un critère d'angle n'est pas satisfait, c'est-à-dire tant que l'angle entre les projections sur le plan tangent de $\mathbf{p}$ de deux voisins consécutifs est supérieur à $\frac{\pi}{2}$. Bien que cette méthode ait été proposée dans le but de construire des *triangles fan* autour des points, elle reste très limitée en pratique, puisque quoi qu'il arrive, k points seront sélectionnés, alors que le nombre de voisins "naturels" varie d'un point à l'autre.

Le voisinage BSP Soit B_i le demi-espace défini par :

$$B_i = \{\mathbf{x} \mid (\mathbf{x} - \mathbf{q}_i) \cdot (\mathbf{p} - \mathbf{q}_i) \geq 0\} \tag{5.20}$$

où $\mathbf{q}_i$ est la projection de $\mathbf{p}_i$ sur le plan tangent de p.

À partir des k plus proches voisins N_p^k, le filtrage par BSP (binary space partition) consiste à supprimer tous les voisins $\mathbf{p}_i \in N_p^k$ situés *derrière* un autre voisin (figure 5.8), c'est-à-dire pour lesquels il existe $\mathbf{p}_j \in N_p^k$ tel que $\mathbf{p}_i \notin B_j$. Le voisinage BSP de $\mathbf{p}$ est alors défini par l'ensemble d'indices N_p^B :

$$N_p^B = \{i \in N_p^k \mid \mathbf{p}_i \in \bigcap_{j \in N_p^k} B_j\} \tag{5.21}$$

Le problème de cette sélection de voisinage est qu'elle est trop sélective et peut engendrer des trous comme illustré sur la figure 5.8-c. En effet, si nous considérons les *triangles fans* formés par les voisinages de l'ensemble des points du nuage de point, alors l'intersection de ces triangles fan peut ne pas recouvrir complètement la surface sous jacente.

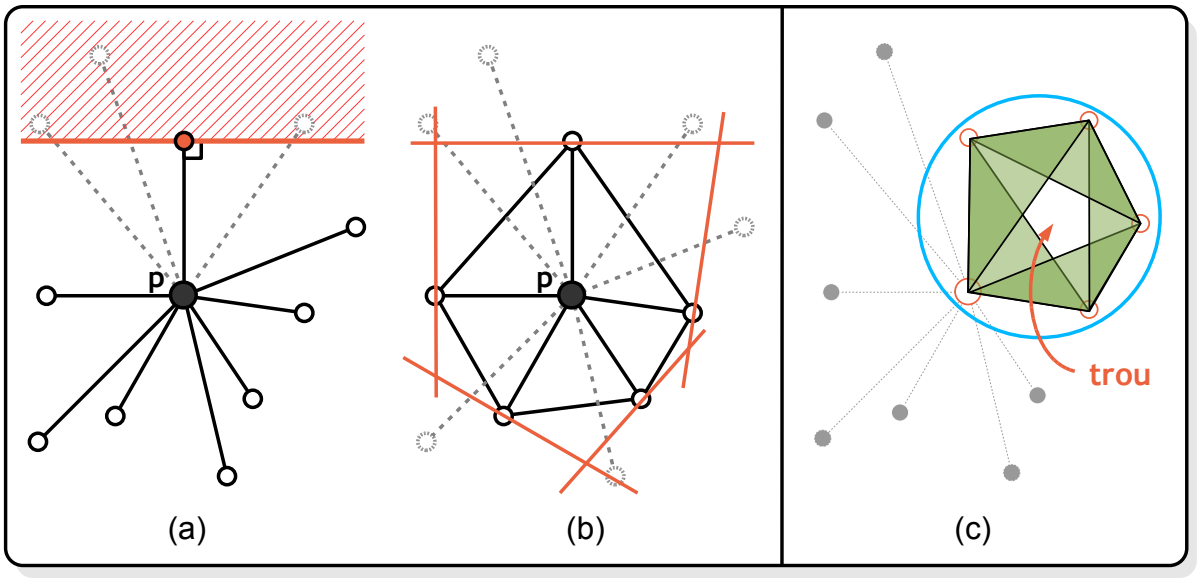


FIG. 5.8 – (a) et (b) Illustration du voisinage BSP. (c) Le voisinage BSP comme le voisinage au sens de Voronoï peuvent générer des trous.

Voisinage au sens de Voronoï Soit V le diagramme de Voronoï des projections $\mathbf{q}_i$, $i \in N_p^k$. La cellule de Voronoï V_i du point $\mathbf{q}_i$ est définie par :

$$V_i = \{\mathbf{x} \in T_p \,|\, \|\mathbf{x} - \mathbf{q}_i\| \leq \|\mathbf{x} - \mathbf{q}_j\| \quad \forall j \in N_p^k,\, j \neq i\} \tag{5.22}$$

où ici T_p dénote le plan tangent du point $\mathbf{p}$. Soit V_p la cellule de Voronoï contenant le point $\mathbf{p}$. Les voisins au sens de Voronoï de $\mathbf{p}$ sont alors l'ensemble des points $\mathbf{p}_i$, $i \in N_p^k$ dont la cellule de Voronoï est adjacente à V_p et sont définis par l'ensemble d'indices N_p^V :

$$N_p^V = \{i \in N_p^k \,|\, V_i \cap V_p \neq \emptyset\} \tag{5.23}$$

Principalement à cause de la projection orthogonale, cette sélection de voisinage peut générer des trous exactement comme pour le cas du voisinage BSP précédent (figure 5.8-c).

Bilan L'analyse des méthodes existantes nous permet de tirer les conclusions suivantes pour le design d'une méthode de sélection d'un premier anneau de voisinage :

- Sélectionner k voisins strictement implique nécessairement de sélectionner des points superflus ou au contraire d'oublier des voisins pertinents.
- La projection orthogonale des voisins potentiels sur le plan tangent local permet d'ordonner les voisins par angle croissant autour du point $\mathbf{p}$ courant. Cependant, énormément d'informations sur les positions relatives des voisins sont perdues durant cette étape, ce qui peut conduire à des aberrations comme sur la figure 5.10-a. De plus, cette projection empêche la relation de voisinage d'être symétrique.
- Afin d'éviter la création de trous, il faut privilégier la souplesse.
- Aucune méthode ne prend en compte l'orientation des points lorsque celle-ci est disponible.

5.3.2 Le voisinage BSP flou

Suite aux remarques précédentes, nous proposons ici une nouvelle méthode de sélection de voisinage inspirée de la méthode de sélection par BSP. Garantir la propriété de symétrie de la relation de voisinnage est particulièrement important pour l'analyse du raffinement (cf. 5.10.1) et implique de s'affranchir de tout type de projection. Cependant, sélectionner de manière robuste un premier anneau pertinent en trois dimensions est une tâche particulièrement difficile. Nous proposons donc deux versions de notre méthode de sélection de voisinage : la première garantissant la propriété de symétrie et la seconde, simple variante, permettant de prendre en compte des configurations géométriques plus complexes.

Pour définir nos relations de voisinage, les idées principales sont d'une part de considérer les distances géodésiques entre les points, ce qui permet indirectement de prendre en compte les normales des points, et d'autre part de rendre la sélection plus souple via l'introduction de la notion de plan discriminant flou.

5.3.2.1 Calcul du voisinage

Le calcul du voisinage N_p du point $\mathbf{p}$ se déroule en plusieurs étapes :

Sélection grossière. Afin d'accélérer le reste des calculs, nous commençons par rechercher l'ensemble des points potentiellement voisins $\widetilde{N}_p$. Nous commençons par considérer le voisinage euclidien N_p^ϵ de $\mathbf{p}$ en prenant pour ϵ le rayon r du point $\mathbf{p}$. Ainsi, N_p^ϵ est l'ensemble des indices de tous les points $\mathbf{p}_i$ inclus dans la boule de centre $\mathbf{p}$ et de rayon r :

$$N_p^\epsilon = \{i \,|\, \mathbf{p}_i \in P^l,\, \mathbf{p}_i \neq \mathbf{p},\, \|\mathbf{p} - \mathbf{p}_i\| < r\} \tag{5.24}$$

Vis-à-vis de la section 5.2.2.3, tous les points $\mathbf{p}_i$, $i \in N_p^\epsilon$ ne satisfaisant pas la condition 5.17 doivent être éliminés.

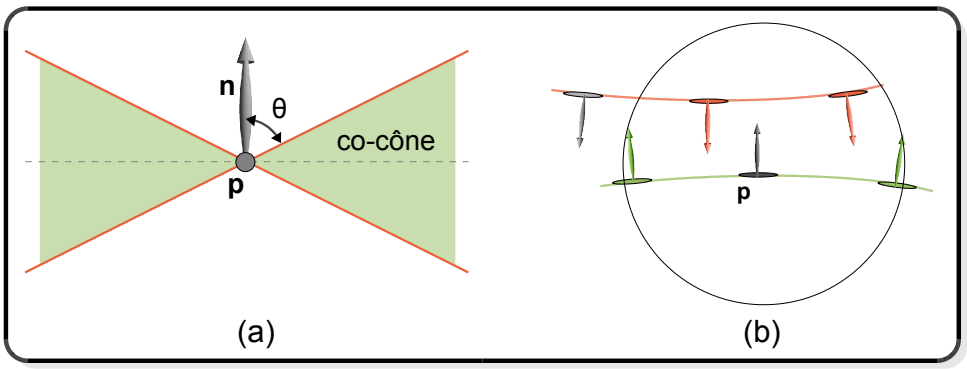


FIG. 5.9 – (a) Définition d'un co-cône. (b) Lorsque deux morceaux de surface sont très proches, et que le nuage de points satisfait certains critères d'échantillonnage par rapport à la courbure local, un test sur les normales permet de séparer simplement les deux surfaces.

De plus, en fonction des connaissances à priori sur le nuage de points considéré, d'autres conditions peuvent être appliquées. Nous pouvons, par exemple, ajouter un filtrage par co-cône

comme suggéré dans [ABK98]. Le principe du filtrage par co-cône est que deux points $\mathbf{p}_0$, $\mathbf{p}_1$ ne peuvent être voisins que si $\mathbf{p}_1$ (resp. $\mathbf{p}_0$) appartient au complémentaire du double cône d'apex $\mathbf{p}_0$ (resp. $\mathbf{p}_1$), d'axe $\mathbf{n}_0$ (resp. $\mathbf{n}_1$) et d'angle θ_{co-cne} (voir figure 5.9-a) :

$$\begin{aligned} C_{co-cne}(\mathbf{p}_0, \mathbf{p}_1) \Leftrightarrow \quad & acos(\left| \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|} \cdot \mathbf{n}_0 \right|) < \theta_{co-cne} \\ \text{et} \quad & acos(\left| \frac{\mathbf{p}_0 - \mathbf{p}_1}{\|\mathbf{p}_0 - \mathbf{p}_1\|} \cdot \mathbf{n}_1 \right|) < \theta_{co-cne} \end{aligned} \tag{5.25}$$

Un choix typique pour l'angle θ_{co-cne} est $\frac{\pi}{4}$. Une autre heuristique est d'inclure un critère d'angle maximal θ_{normal} entre les normales :

$$C_{normal}(\mathbf{p}_0, \mathbf{p}_1) \Leftrightarrow acos(\mathbf{n}_0 \cdot \mathbf{n}_1) > \theta_{normal} \tag{5.26}$$

Ce critère permet dans certains cas de séparer deux morceaux de surface proches (figure 5.9-b). Remarquons que dans le cas précis de la figure 5.9-b, notre test C_{cne} permet également de séparer les deux surfaces.

Finalement, une première approximation $\widetilde{N}_p$ de notre voisinage est résumée par :

$$\widetilde{N}_p = \{i \in N_p^\epsilon \,|\, C_{cne}(\mathbf{p}, \mathbf{p}_i) \text{ et } C_{co-cne}(\mathbf{p}, \mathbf{p}_i) \text{ et } C_{normal}(\mathbf{p}, \mathbf{p}_i) \text{ et } \ldots\} \tag{5.27}$$

Notons bien que les conditions C_{co-cne} et C_{normal} ne doivent pas être appliquées si le nuage de points ne les satisfait pas.

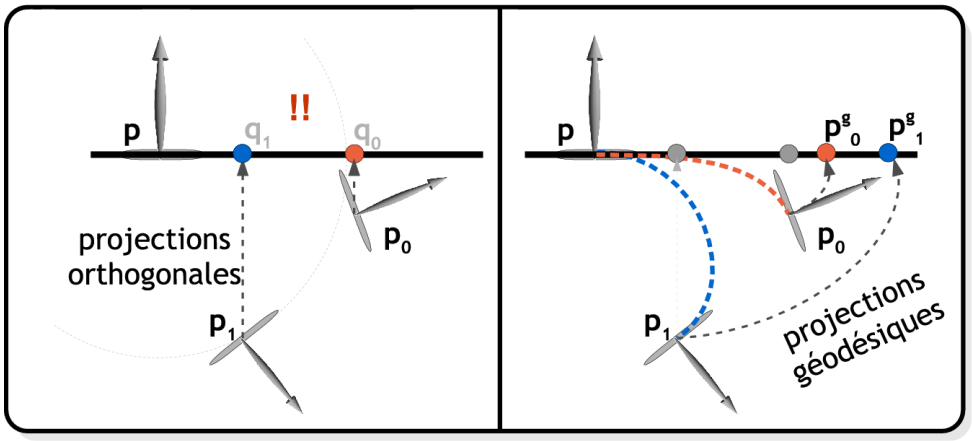


FIG. 5.10 – Projection géodésique.

"Projection géodésique". Cette étape consiste à calculer la "projection géodésique" $\mathbf{p}_i^g$ de chacun des voisins $\mathbf{p}_i$, $i \in \widetilde{N}_p$ sur le plan tangent de $\mathbf{p}$. Le point $\mathbf{p}_i^g$ est la projection orthogonale de $\mathbf{p}_i$ déplacée de sorte que la distance entre $\mathbf{p}$ et $\mathbf{p}_i^g$ soit égale à la distance géodésique entre $\mathbf{p}$ et $\mathbf{p}_i$ (équation 5.14) :

$$\mathbf{p}_i^g = \mathbf{p} + \widetilde{G}(\mathbf{p}, \mathbf{p}_i) * \frac{Q_p(\mathbf{p}_i) - \mathbf{p}}{\|Q_p(\mathbf{p}_i) - \mathbf{p}\|} \tag{5.28}$$

La projection des voisins sur un plan 2D permet de simplifier l'étape de sélection suivante, tandis que conserver les distances géodésiques entre $\mathbf{p}$ et les projections de ses voisins permet d'ordonner correctement les voisins en cas de forte courbure comme illustré sur la figure 5.10.

Filtrage par "BSP flou". Cette dernière étape consite à appliquer le filtrage par BSP flou sur l'ensemble des points $\mathbf{p}_i^g$, $i \in \widetilde{N}_p$. Intuitivement, l'idée est de supprimer tous les voisins situés fortement derrière un autre voisin, ainsi que ceux situés entre et légèrement derrières deux autres voisins.

Pour cela, nous introduisons la notion de plan discriminant flou en définissant la valeur $w_{i,j}$ exprimant à quel point le point $\mathbf{p}_i^g$ est derrière le point $\mathbf{p}_j^g$ relativement au point courant $\mathbf{p}$ (voir figure 5.11). La valeur de $w_{i,j}$ varie de 0 à 1 lorsque l'angle $a_{i,j} = \widehat{\mathbf{p}\mathbf{p}_j^g\mathbf{p}_i^g}$ varie de θ_0 à θ_1 :

$$w_{i,j} = \frac{a_{i,j} - \theta_0}{\theta_1 - \theta_0} \tag{5.29}$$

En pratique, afin d'éviter le calcul d'un arc-cosinus pour obtenir la valeur de $a_{i,j}$, nous utilisons pour $w_{i,j}$ l'approximation suivante (voir figure 5.11-b) :

$$w_{i,j} \approx \frac{cos(a_{i,j}) - cos(\theta_0)}{cos(\theta_1) - cos(\theta_0)} \tag{5.30}$$

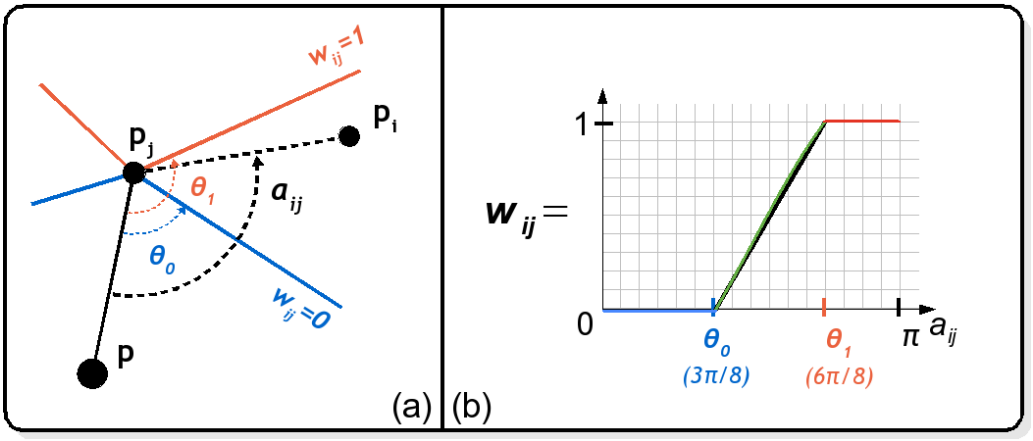


Fig. 5.11 – Plan discriminant flou.
(a) Un plan discriminant flou est défini par deux angles θ_0 et θ_1. (b) La courbe verte correspond à notre approximation.

La valeur $w_{i,j}$ définie donc un champ implicite où chaque iso-valeur $v \in]0,1[$ correspond à un cône d'angle $\theta_0(v-1) + \theta_1 v$. Prendre $\theta_0 = \theta_1 = \frac{\pi}{2}$ est équivalent à prendre un simple plan. En pratique, ces deux angles doivent être choisis de manière à ce que l'isovaleur 0.5 correspond à un cône d'angle supérieur à $\frac{\pi}{2}$, c'est-à-dire que $\frac{\theta_0+\theta_1}{2}$ soit légèrement supérieur à $\frac{\pi}{2}$. Cela permet d'assurer une certaine souplesse dans la sélection au sens où plus de points seront sélectionnés, évitant ainsi de laisser des trous comme nous l'avions vue pour les autres méthodes figure 5.8-c. Un choix typique est par exemple de prendre $\theta_0 = \frac{3\pi}{8}$ et $\theta_1 = \frac{6\pi}{8}$.

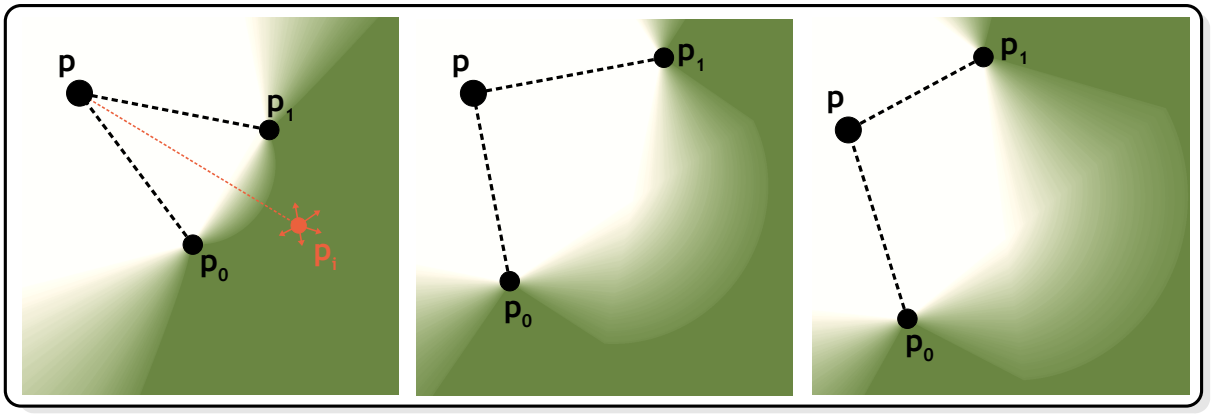


FIG. 5.12 – Illustration du mélange de deux plans discriminants flous pour différentes configurations. Le gradient indique la valeur de pénalité $w_i = w_{i,0} + w_{i,1}$ induite par les deux points $\mathbf{p}_0$ et $\mathbf{p}_1$ pour tous les points de l'espace (en fait, cette valeur n'est valide que pour les points $\mathbf{p}_i$ situés dans le secteur angulaire $\mathbf{p}_0\mathbf{p}\mathbf{p}_1$).

Soit $Succ_i$ (resp. $Pred_i$) l'ensemble des successeurs (resp. prédécesseurs) du point $\mathbf{p}_i$, $i \in N_p^2$ tel que $Succ_i = \{j \in N_p^\epsilon \,|\, 0 < \widehat{\mathbf{p}_i^g\mathbf{p}\mathbf{p}_j^g} < \pi\}$ (resp. $Pred_i = \{j \in N_p^2 \,|\, -\pi < \widehat{\mathbf{p}_i^g\mathbf{p}\mathbf{p}_j^g} < 0\}$). Ces définitions sont illustrées figure 5.13-a.

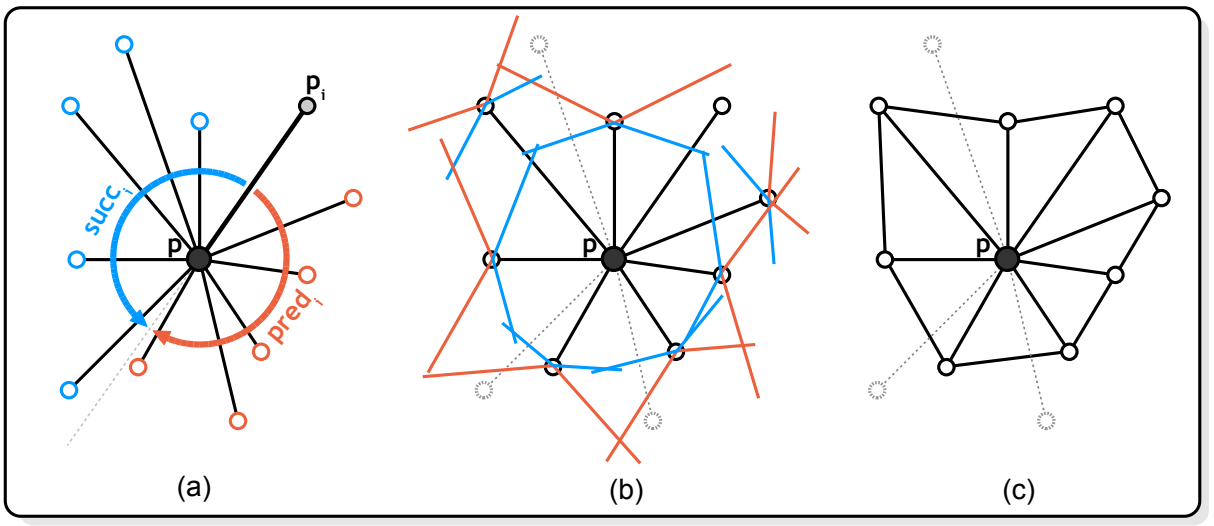


FIG. 5.13 – (a) Définition des successeurs et prédécesseurs du point $\mathbf{p}_i$. (b) Application de notre filtrage par BSP flou sur le voisinage du point $\mathbf{p}$. (c) Le voisinage du point $\mathbf{p}$ défini implicitement un *triangle fan*.

Nous pouvons maintenant combiner nos plans discriminants flous en associant à chaque candidat $\mathbf{p}_i^g$ une valeur de pénalité w_i signifiant à quel point $\mathbf{p}_i^g$ est derrière le voisinage de $\mathbf{p}$ tout entier (figure 5.13). w_i est la somme des deux valeurs de pénalité maximale induites par les

prédécesseurs et successeurs de p_i :

$$w_i = \max_{j \in Succ_i} (w_{i,j}) + \max_{j \in Pred_i} (w_{i,j}) \tag{5.31}$$

Ce mélange est illustré figure 5.12. Finalement, un voisin $\mathbf{p}_i$ est supprimé du voisinage dès qu'il a atteint une pénalité w_i supérieure ou égale à 1 et :

$$N_p^f = \{i | i \in N_p^2, w_i < 1\} \tag{5.32}$$

Les figures 5.13-b et 5.13-c illustrent l'application de notre filtrage BSP flou sur le même exemple que celui de la figure 5.8 illustrant le filtrage BSP "dur".

5.3.2.2 Version symétrique du voisinage BSP flou

La symétrie de la relation de voisinage est une propriété importante permettant de garantir quelques bonnes propriétés de notre algorithme de raffinement (section 5.10.1) et est également requit par notre stratégie de raffinement diadique (section 5.4.1). Un voisinage N^s est dit symétrique si et seulement si :

$$\forall\, (\mathbf{p}_i, \mathbf{p}_j) \in P \times P\,,\ i \in N_{p_j}^s \Leftrightarrow j \in N_{p_i}^s \tag{5.33}$$

En fait, rendre une relation symétrique, que ce soit une relation de voisinage ou non, est théoriquement très facile. Par exemple, nous pouvons établir le voisinage symétrique N^s à partir de notre voisinage non symétrique N, de manière conjonctive :

$$\forall\, \mathbf{p}_i \in P\,,\ N_{p_i}^s = \left\{ j \,|\ \in j \in N_{p_i} \text{ et } i \in N_{p_j} \right\} \tag{5.34}$$

ou bien de manière disjonctive :

$$\forall\, \mathbf{p}_i, \in P\,,\ N_{p_i}^s = \left\{ j \,|\ \in j \in N_{p_i} \text{ ou } i \in N_{p_j} \right\} \tag{5.35}$$

Cependant, en pratique cela implique d'évaluer et de stocker l'ensemble des relations de voisinage N_{p_i} pour tous les points $\mathbf{p}_i \in P$, puis de les combiner entre elle. Ceci est à la fois extrêmement coûteux en temps de calcul et en mémoire et invaliderait une utilisation purement locale.

Aussi nous proposons plutôt de rendre notre voisinage symétrique naturellement. Ceci est relativement facile puisque qu'il suffit de s'affranchir de la deuxième étape de projection géodésique, c'est-à-dire de prendre simplement $\mathbf{p}_i^g = \mathbf{p}_i$. En revanche, l'absence de projection ainsi que l'impossibilité de prendre en compte les distances géodésiques diminue significativement la robustesse de la sélection en cas de fortes irrégularités dans le nuage de points.

5.3.2.3 Finalisation

Finalement, les voisins $\mathbf{p}_i$, $i \in N_p^f$ sont triés par angle croissant de leur projection $\mathbf{p}_i^g$ sur le plan tangent de $\mathbf{p}$:

$$N_p = \{i_0, i_1, \ldots, i_j, \ldots\} \text{ tel que } \widehat{\mathbf{p}_{i_0}^g \mathbf{p} \mathbf{p}_{i_j}^g} < \widehat{\mathbf{p}_{i_0}^g \mathbf{p} \mathbf{p}_{i_{j+1}}^g}\ \forall\, i_j \in N_p \tag{5.36}$$

Trié de la sorte, le voisinage N_p forme implicitement un *triangle fan* autour du point $\mathbf{p}$ (figure 5.13-c).

5.4 Les stratégies de raffinement

Nous appelons "stratégie de raffinement", l'ensemble des règles qui, à partir du *triangle fan* sélectionné à l'étape précédente, fournissent l'ensemble L_p des positions où il sera possible d'insérer un nouveau point. Puisque basées sur le seul premier anneau de voisinage, ces règles sont par définition extrêmement locales. Afin d'obtenir une surface lisse, les positions des points de L_p doivent être lissées, c'est-à-dire être le résultat de l'un de nos opérateurs de lissage précédemment définis. Les règles de raffinement et les opérateurs de lissage sont donc extrêmement liés.

Ainsi, ces règles de raffinement sont très importantes puisqu'elles influent à la fois sur la qualité/régularité de l'échantillonnage généré, sur la vitesse de raffinement (i.e. le facteur d'augmentation de la densité) mais aussi sur la lisseur/qualité de la surface générée. Nous proposons ici deux stratégies de raffinement différentes : la première pouvant être qualifiée de *diadique* et la seconde étant inspirée du raffinement $\sqrt{3}$.

5.4.1 Raffinement diadique

Par définition, avec une stratégie de raffinement diadique le nombre de points est multiplié par 2 dans chacune des directions à chaque pas de subdivision, c'est-à-dire par quatre sur les surfaces. Pour les maillages triangulaires, cela correspond par exemple aux surfaces de subdivisions du Loop [Loo87] ou du Butterfly [ZSS96] dans lesquels un nouveau point est inséré pour chaque arête du maillage.

Application au raffinement des nuages de points Afin de reproduire un raffinement *diadique*, la règle de base est d'insérer un nouveau point pour chaque paire de voisins. L'ensemble L_p des positions possibles pour l'insertion de nouveaux points est alors l'ensemble des milieux des segments $cog(\mathbf{p}, \mathbf{p}_i$ lissés par notre opérateur de lissage ϕ_2 :

$$L_p = \{cog(\mathbf{p}, \mathbf{p}_i) + \phi_2(\mathbf{p}, \mathbf{p}_i) \,|\, i \in N_p\} \tag{5.37}$$

Cependant, il peut arriver que les relations de voisinage se croisent, générant des ambiguïtés d'un point de vue global, comme dans l'exemple de la figure 5.14-a. Sur quels critères privilégier l'une ou l'autre paire de voisins ? Afin de rendre notre algorithme de raffinement le moins sensible possible à toute connectivité arbitraire, nous proposons de détecter les groupes de points étant tous voisins les uns des autres et de n'insérer qu'un seul point au centre de ces groupes en prenant en compte de manière équitable chacun des points.

La procédure de construction de l'ensemble L_p est alors la suivante. Considérons le voisin $\mathbf{p}_{i_j}$, $i_j \in N_p$. Avant d'insérer dans L_p le centre de la paire $\mathbf{p}$, $\mathbf{p}_{i_j}$ nous devons détecter l'ensemble des relations de voisinage *croisant* cette paire. Pour cela, nous allons construire l'ensemble H_k initialisé avec la paire $\mathbf{p}$, $\mathbf{p}_{i_j}$. Ensuite, si le prédécesseur $\mathbf{p}_{i_{j-1}}$ et le successeur $\mathbf{p}_{i_{j+1}}$ satisfont la relation de voisinage R entre eux ainsi qu'avec le point $\mathbf{p}_{i_j}$, alors ces deux points sont insérés dans H_k. Puis, les successeurs $\mathbf{p}_{i_{j+2}}$, ... sont itérativement insérés dans H_k tant que la relation de voisinage symétrique R est satisfaite entre tous les points de H_k. À la fin, nous avons donc deux possibilités pour H_k :

$$\begin{aligned} H_k \;=\; & \{\mathbf{p}, \mathbf{p}_{i_j}\} \\ & \{\mathbf{p}, \mathbf{p}_{i_{j-1}}, \mathbf{p}_{i_j}, \ldots, \mathbf{p}_{i_l}\} \text{ tel que } \mathbf{p}_m \, R \, \mathbf{p}_n \;\forall\, \mathbf{p}_m \in C_j, \mathbf{p}_n \in C_j \end{aligned}$$

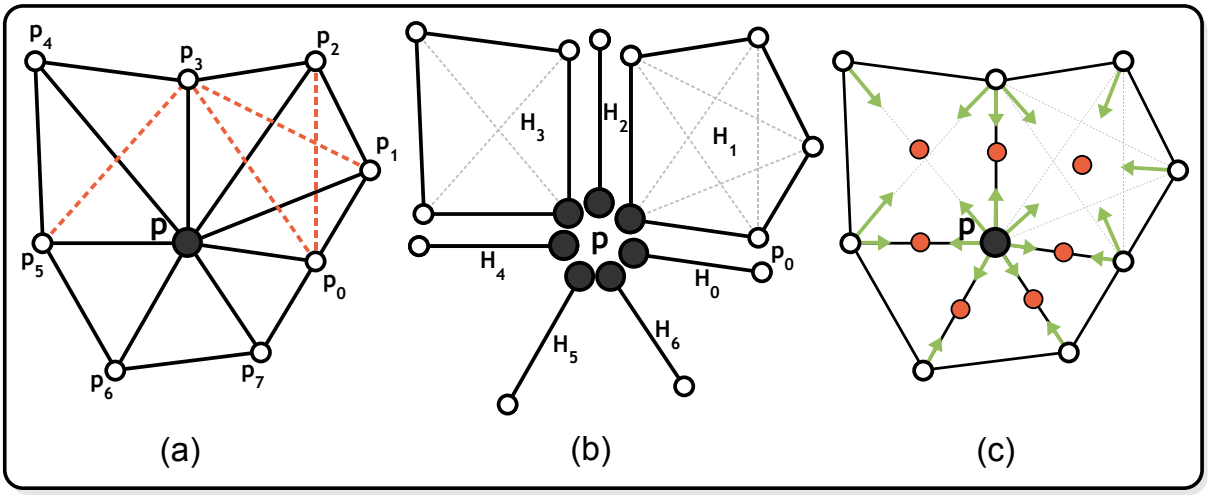


FIG. 5.14 – Notre procédure de raffinement diadique.
(a) Le voisinage du point courant $\mathbf{p}$. Les lignes rouges montrent les relations de voisinage qui existent entre les voisins de $\mathbf{p}$. (b) Formation des groupes de points H_k. (c) Règles d'insertion des nouveaux points : l'ensemble des points rouges défini l'ensemble L_p.

Le centre de gravité de H_k lissé par notre opérateur de lissage est inséré dans L_p :

$$L_p \leftarrow L_p \cup \{cog(H_k) + \phi_{|H_k|}(H_k)\} \tag{5.38}$$

Nous recommençons ce processus en construisant H_{k+1} à partir du dernier voisin $\mathbf{p}_{i_l}$ inséré dans H_k ou bien à partir du voisin suivant $\mathbf{p}_{i_{j+1}}$ si aucun croisement n'a été détecté (c'est-à-dire si $|H_k| = 2$). Ce processus est illustré figure 5.14.

Cette construction est dépendante du voisin de départ puisque seul le premier prédécesseur est considéré, ce qui peut poser problème si ce point participe à un groupe d'au moins cinq points voisins. Par exemple, sur l'exemple de la figure 5.14 il ne faudrait pas commencer par le point $\mathbf{p}_2$. Une heuristique simple, mais efficace, est de commencer avec le plus proche des voisins puisque celui-ci a très peu de chances de participer à un groupe de plus de 4 points voisins.

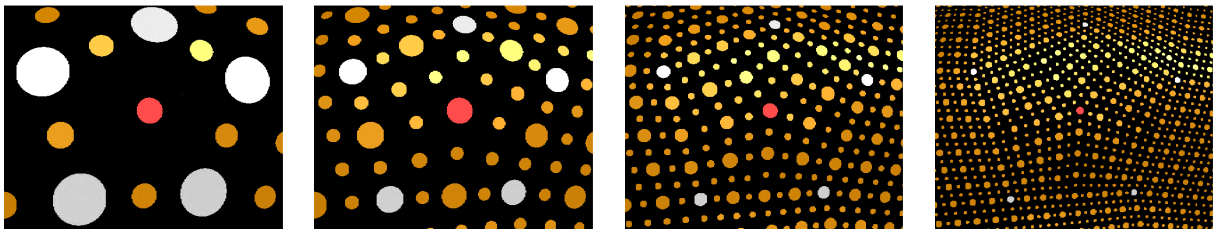


FIG. 5.15 – Illustration de la procédure de raffinement diadique. Chaque image illustre un pas de raffinement, les petits points représentant les nouveaux points insérés.

Afin de garantir une cohérence globale, cette stratégie de raffinement diadique par regroupement nécessite l'utilisation d'une relation de voisinage symétrique. La version symétrique de notre voisinage par BSP flou, que nous avons présenté à la section 5.3.2.2 est donc ici extrêmement utile. Cependant, cette version ne prend pas en compte les orientations des splats et est

donc moins robuste en cas de fort sous-échantillonnage. En ce sens, nous pouvons déjà affirmer que cette stratégie de raffinement diadique sera moins robuste que la suivante, qui elle autorise l'utilisation de la version robuste de notre voisinage par BSP flou.

5.4.2 Raffinement $\sqrt{3}$

Le raffinement $\sqrt{3}$ a été proposé en 2000 par Kobbelt [Kob00]. À partir d'un maillage triangulaire, le principe est d'insérer un nouveau point au centre de chaque face. Cette insertion est ensuite suivie d'un remaillage (voir figure 5.16). Comme son nom l'indique, le nombre de points est multiplié par $\sqrt{3}$ dans chacune des directions à chaque pas de subdivision, soit par 3 sur la surface.

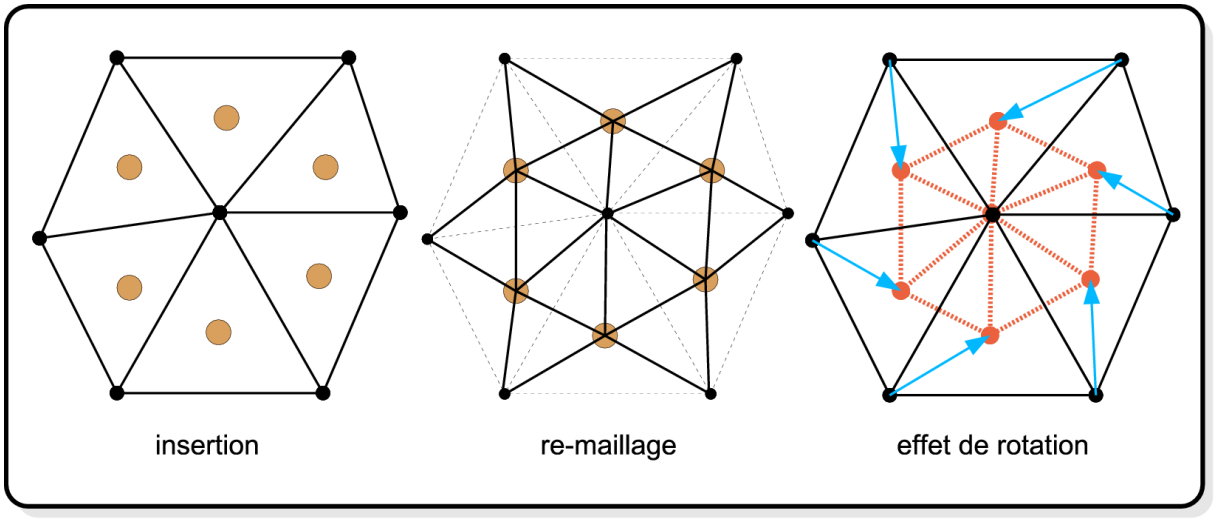


FIG. 5.16 – Le raffinement $\sqrt{3}$.

Le principal avantage du raffinement $\sqrt{3}$ est d'avoir une règle d'insertion des nouveaux points extrêmement locale, puisque seuls trois points sont nécessaires, tout en générant une surface lisse C^2. La lisseur de la surface limite, malgré une très forte localité du masque de subdivision, s'explique intuitivement par l'effet de rotation de ce schéma (figure 5.16).

Application au raffinement des nuages de points Avec une stratégie de raffinement $\sqrt{3}$, l'ensemble des locus L_p pour l'insertion des nouveaux points est le résultat de l'application de notre opérateur de lissage sur le centre de gravité de chaque triangle du *triangle fan* formé par le voisinage N_p :

$$L_p = \{cog(\mathbf{p}, \mathbf{p}_i, \mathbf{p}_{i+1}) + \phi_3(\mathbf{p}, \mathbf{p}_i, \mathbf{p}_{i+1}) \,|\, i \in N_p\} \tag{5.39}$$

En plus de fournir une règle unique et simple pour l'insertion des nouveaux points, le raffinement $\sqrt{3}$ devrait également permettre un lissage supplémentaire de la surface générée comme illustré par la figure 5.17. En effet, il est important de remarquer que par construction, les carreaux de Bézier triangulaires construits autour d'un point ne sont que C^0 continus aux niveaux des jointures de deux carreaux (figure 5.17-a). Cependant, au prochain pas de raffinement deux nouveaux points devraient être insérés au niveau de chacune des discontinuités précédentes à partir de carreaux lisses, lissant ainsi les discontinuités. (figure 5.17-b).

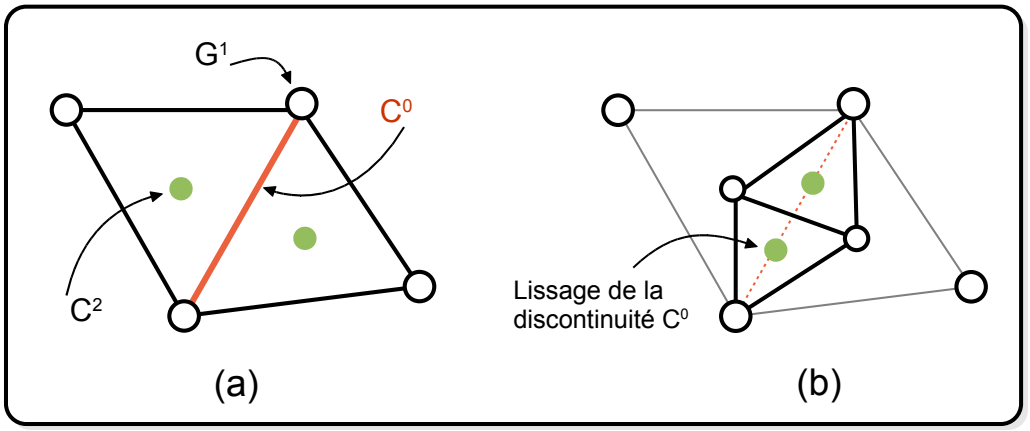


FIG. 5.17 – Lissage du raffinement $\sqrt{3}$. (a) La jonction de deux triangles de Bézier n'est que C^0 continue. (b) Au pas de raffinement suivant, des carreaux de Bézier lisses sont construits au-dessus la discontinuité précédente.

5.5 Raffinement local et contrôle de l'échantillonnage

Dans cette section nous allons voir comment raffiner de manière locale le voisinage du point courant $\mathbf{p}$ tout en évitant la redondance (i.e. ne pas insérer deux fois le même point) et le sur-échantillonnage local (i.e. ne pas insérer des points trop près). À cette étape du raffinement du point $\mathbf{p}$ nous disposons :

- du nuage de points en cours de raffinement P^l,
- de l'ensemble des points de $P^{l+1} - P^l$ créés lors du raffinement des points précédant $\mathbf{p}$,
- du premier anneau de voisinage N_p^f de $\mathbf{p}$,
- ainsi que de l'ensemble L_p des locus possibles pour l'insertion de nouveaux points.

L'ensemble L_p dépend de la stratégie de raffinement utilisée (section précédente).

Pour cette dernière étape les principaux challenges sont :

- Augmenter la densité de points autour du point $\mathbf{p}$ en insérant suffisement de nouveaux points de façon à ne pas laisser ou créer de trous.
- Régulariser le voisinage de $\mathbf{p}$, et par conséquent éviter de créer un sur-échantillonnage local en insérant trop de points.

Afin de satisfaire au mieux ces deux contraintes antagonistes, nous allons créer itérativement un nouveau voisinage N'_p autour du point $\mathbf{p}$. Ce nouveau voisinage doit être plus petit que N_p au sens où le plus éloigné des nouveaux voisins doit être plus proche que le plus éloigné des anciens voisins. N'_p doit également être le plus régulier possible. De plus, durant sa construction, nous devons prendre en compte tous les points déjà insérés et insérer un nouveau point uniquement si cela est réellement nécessaire.

Nous commençons par initialiser N'_p avec l'ensemble des points de P^{l+1} "proche" de $\mathbf{p}$ (figures 5.18-a,b) :

$$N'_p = \{i | \|\mathbf{p}_i - \mathbf{p}_i\| < \lambda r \, , \, \mathbf{p}_i \in P^{l+1}\} \tag{5.40}$$

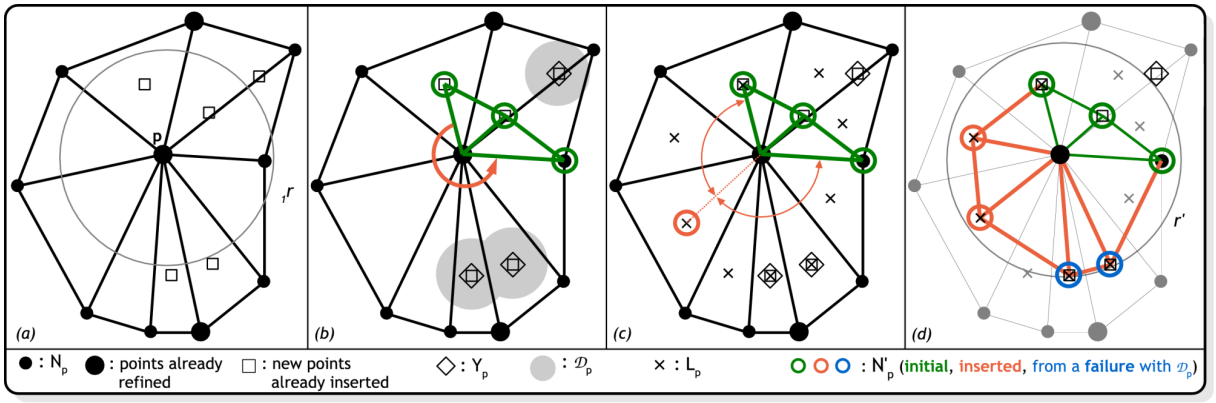


FIG. 5.18 – Illustration du contrôle de l'échantillonnage dans le cas du $\sqrt{3}$. (a) Le voisinage du point courant $\mathbf{p}$ avant son raffinement. Deux de ses voisins ont déjà été raffinés : les nouveaux points insérés lors de leur raffinement sont matérialisés par les carrés. (b) Initialisation du nouveau voisinage N'_p (en vert). (c) Sélection d'un point à insérer. (d) Le nouveaux voisinage N'_p est complet, le rayon du point $\mathbf{p}$ est mis à jour.

où r est le rayon du point $\mathbf{p}$ et λ est défini de sorte que λr corresponde au rayon de $\mathbf{p}$ après raffinement si son voisinage était parfaitement régulier. λ dépend donc de la stratégie de raffinement, et par définition nous avons :

- $\lambda = \frac{1}{\sqrt{3}}$ pour le raffinement $\sqrt{3}$.
- $\lambda = \frac{1}{2}$ pour le raffinement diadique.

Nous considérons que le raffinement de $\mathbf{p}$ est complet dès que "l'angle courbe" maximal entre deux voisins consécutifs de N'_p est inférieur à un seuil $\theta_c = \frac{\pi}{2}$. Ainsi, tant que N'_p n'est pas complet, un nouveau point parmi ceux de L_p doit être inséré.

Cependant, un point ne doit être définitivement inséré que si celui-ci est suffisement éloigné des autres points. En pratique, de par la nature de nos règles d'insertions, un point de l'ensemble L_p est forcément bien positionné par rapport aux autres points du nuage P^l. De plus, les points de $P^{l+1} - P^l$ qui ont déjà été insérés dans N'_p seront pris en compte naturellement par la procédure d'insertion. Aussi nous définissons Y_p, l'ensemble des points déjà insérés qui sont suffisement proche de $\mathbf{p}$ mais qui n'ont pas déjà été sélectionné dans N'_p (figure 5.18-b) :

$$Y_p = \{i \mid \mathbf{p}_i \in P^{l+1} - P^l,\ \lambda r < \|\mathbf{p}_i - \mathbf{p}\| < r\}, \tag{5.41}$$

À partir de Y_p nous définisons l'espace $\mathcal{D}_p$ représentant l'ensemble des positions interdites pour l'insertion d'un nouveaux points. Cet espace est l'union des sphères de rayon $\frac{1}{2}\lambda r$ centrées sur les points de Y_p (figure 5.18-b) :

$$\mathcal{D}_p = \{x \mid i \in Y_p,\ \|x - \mathbf{p}_i\| < \frac{1}{2}\lambda r\} \tag{5.42}$$

Finalement, la procédure d'insertion est la suivante :

Tant que le voisinage N'_p n'est pas complet **répéter**

1. Sélectionner la paire de points consécutifs $\mathbf{p}_j$, $\mathbf{p}_{j+1}$ dans N'_p ayant "l'angle courbe" maximal. (figure 5.18-b).

2. Sélectionner le nouveau point dans L_p qui optimise au mieux l'échantillonnage de points (figure 5.18-c). Un bon candidat est le point $\mathbf{p}_k \in L_p$ tel que le minimum des deux angles $\widehat{\mathbf{p}_j\mathbf{p}\mathbf{p}_k}$ et $\widehat{\mathbf{p}_k\mathbf{p}\mathbf{p}_{j+1}}$ soit maximal.

3. Si ce point n'est pas trop proche d'un point déjà inséré, i.e. $\mathbf{p}_k \notin \mathcal{D}_p$, alors ce point est inséré dans P^{l+1} et dans N'_p. Dans le cas contraire, le candidat $\mathbf{p}_k$ est simplement oublié, aucun point n'est inséré dans P^{l+1} et le point de P^{l+1} le plus proche de $\mathbf{p}_k$ est inséré dans N'_p (figure 5.18-d). Ainsi, si les points sont suffisement denses localement, aucun point n'est inséré.

Lorsque ce processus est terminé, le rayon du point $\mathbf{p}$ est mis à jour en fonction de son nouveau voisinage N'_p. Le nouveau radius r' est égal à la distance maximale entre $\mathbf{p}$ et ses nouveaux voisins :

$$r' = \max_{j \in N'_p}(\|\mathbf{p} - \mathbf{p}_j\|) \tag{5.43}$$

Le rayon r_j de chaque point p_j, $j \in N'_p$ est mis a jour en prenant le maximum des quatre valeurs suivantes : r_j, $\|p_j - p\|$, $\|p_j - p_{j-1}\|$ et $\|p_j - p_{j+1}\|$.

Uniformité locale versus uniformité globale

Avec la stratégie de raffinement qui vient d'être présentée, une zone plus dense que le reste de l'objet restera également plus dense après un ou plusieurs pas de raffinement. Ce comportement peut être souhaité surtout si la densité du nuage de points initial a déjà été adaptée à la courbure locale de la surface sous-jacente. Toutefois, il peut parfois être utile de tendre vers un échantillonnage globalement régulier. Ce comportement peut facilement être obtenu en utilisant un rayon global pour l'initialisation du voisinage N'_p. Rappelons que cette initialisation est effectuée en sélectionnant tous les points dans un rayon de λr où r était le rayon du point courant. Une variante consiste donc à prendre pour ce r le rayon du plus large splat du nuage de points courant. De cette manière, les zones les plus denses ne seront raffinées que lorsque les zones les moins denses auront été suffisement raffinées, ce qui garanti une certaine uniformité globale de l'échantillonnage du nuage de points. Cette stratégie a par exemple été utilisée pour l'exemple de reconstruction d'un large trou de la figure 5.27. Dans cet exemple, les points au bord du trou ont un rayon au moins égal à la largeur du trou et donc à peu près 50 fois supérieur aux autres points du modèle. Une stratégie d'uniformité globale est donc ici indispensable.

5.6 Raffinement des bords et arêtes

Jusqu'à présent nous avons fait l'hypothèse que le nuage de points à raffiner représentait une surface lisse et fermée, c'est-à-dire sans aucune discontinuité. Dans cette section, nous allons voir comment notre méthode peut être étendue au raffinement des bords et arêtes.

5.6.1 Représentation des bords, arêtes et coins

Comme nous l'avons vu dans l'état de l'art, de telles discontinuités peuvent être représentées en associant aux splats des lignes de découpe définies dans leur plan tangent. Afin de faciliter leur raffinement, nous ajoutons comme contrainte que les lignes de découpe doivent en plus

passer par le centre du splat ou, autrement dit que les points soient positionnés précisement sur les courbes de discontinuité.

De cette façon, une arête peut être représentée plus simplement et de manière plus compacte, par un seul point ayant deux normales différentes (voir des exemples sur la figure 5.19). Lors du raffinement, le produit vectoriel de ces deux normales fournit directement un vecteur tangent de la courbe définissant la discontinuité. Ce produit vectoriel permet aussi, au moment du rendu, de déterminer la ligne de découpe du splat. Un tel point est donc rendu deux fois en inversant l'ordre des normales.

Un splat sur une bordure est représenté de la même façon, la seule différence est qu'il ne doit être tracé qu'une seule fois. En fait, la seconde normale permet juste de définir la ligne de découpe du splat ; ainsi les "splats de bordure" et "splats d'arête" peuvent être traités de la même manière. Par extension, un coin est représenté par un point ayant trois normales différentes. Lors du rendu, ce point doit donc être tracé trois fois.

5.6.2 Détection et interpolation des arêtes

Quelle que soit la stratégie de raffinement utilisée, nos opérateurs d'interpolation et de lissage sont basés sur le calcul de vecteurs tangents allant d'un point vers un autre. Rappelons que le vecteur tangent allant du point $\mathbf{p}_0$ vers le point $\mathbf{p}_1$ est obtenu par projection du point $\mathbf{p}_1$ sur le plan tangent de $\mathbf{p}_0$. Comme les points peuvent maintenant avoir plusieurs normales, nous devons distinguer trois cas possibles pour le calcul du vecteur tangent $\mathbf{t}_{0,1}$ allant du point $\mathbf{p}_0$ vers le point $\mathbf{p}_1$:

1. $\mathbf{p}_0$ n'a qu'une seule normale : pas de changement, $\mathbf{t}_{0,1}$ est obtenue par projection de $\mathbf{p}_1$ sur le plan tangent de $\mathbf{p}_0$.

2. $\mathbf{p}_0$ a deux normales $\mathbf{n}_0^1$, $\mathbf{n}_0^2$ et $\mathbf{p}_1$ satisfait la condition suivante :

$$\left| \mathbf{v}_0 \cdot \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|} \right| > cos(\theta_{arte}) \tag{5.44}$$

   où $\mathbf{v}_0$ correspond au vecteur tangent de la courbe représentant l'arête : $\mathbf{v}_0 = \frac{\mathbf{n}_0^1 \wedge \mathbf{n}_0^2}{\|\mathbf{n}_0^1 \wedge \mathbf{n}_0^2\|}$. Cette condition définit un cône d'apex $\mathbf{p}_0$, d'axe $\mathbf{v}_0$ et d'angle θ_{arte}. Cela permet de tester si le point $\mathbf{p}_1$ est ou non sur la discontinuité. Le choix de l'angle θ_{arte} dépend en fait du type du point $\mathbf{p}_1$. Si $\mathbf{p}_1$ représente également une arête alors l'angle de tolérance peut être assez large afin de favoriser la connexion des arêtes ($\frac{3\pi}{8}$ est un choix raisonnable). D'un autre côté, si $\mathbf{p}_1$ est un simple splat alors la discontinuité ne doit être prolongée que si $\mathbf{p}_1$ est très proche de la discontinuité, ce qui implique un angle de tolérance faible (par exemple, $\frac{\pi}{6}$). Dans ce dernier cas, l'arête sera progressivement mélangée avec une surface lisse (figure 5.19).
   Finalement, dans ce cas nous prenons pour $\mathbf{t}_{0,1}$:

$$\begin{aligned} &\frac{1}{3}\|\mathbf{p}_1 - \mathbf{p}_0\|\mathbf{v}_0 \quad \text{si} \quad \mathbf{v}_0 \cdot (\mathbf{p}_1 - \mathbf{p}_0) > 0 \\ -&\frac{1}{3}\|\mathbf{p}_1 - \mathbf{p}_0\|\mathbf{v}_0 \quad \text{sinon} \end{aligned} \tag{5.45}$$

3. Si $\mathbf{p}_0$ a deux normales et que la condition 5.44 n'est pas satisfaite, alors le vecteur tangent $\mathbf{t}_{0,1}$ est obtenu en projetant $\mathbf{p}_1$ sur les deux plans tangents de $\mathbf{p}_0$ et en conservant la projection la plus proche de $\mathbf{p}_1$.

Dans le cas où $\mathbf{p}_0$ représente un coin (i.e. $\mathbf{p}_0$ a 3 normales), la normale associée au plan tangent le plus éloigné du point $\mathbf{p}_1$ est simplement oubliée pour se ramener aux cas 2 ou 3. Nous avons donc six combinaisons possibles pour la construction d'une courbe interpolante entre deux points. Parmi ces six combinaisons seules trois impliquent une arête (ou un bord) entre les deux points considérés. Il s'agit des combinaisons 2-2, 1-2 et 2-3 qui sont résumées sur la figure 5.19. Lorsqu'une arête est détectée entre deux points, celle-ci est reconstruite par une courbe cubique de Bézier exactement comme pour notre opérateur de lissage ϕ_2 (section 5.2.1.1), grâce aux vecteurs tangents calculés comme nous venons de le voir.

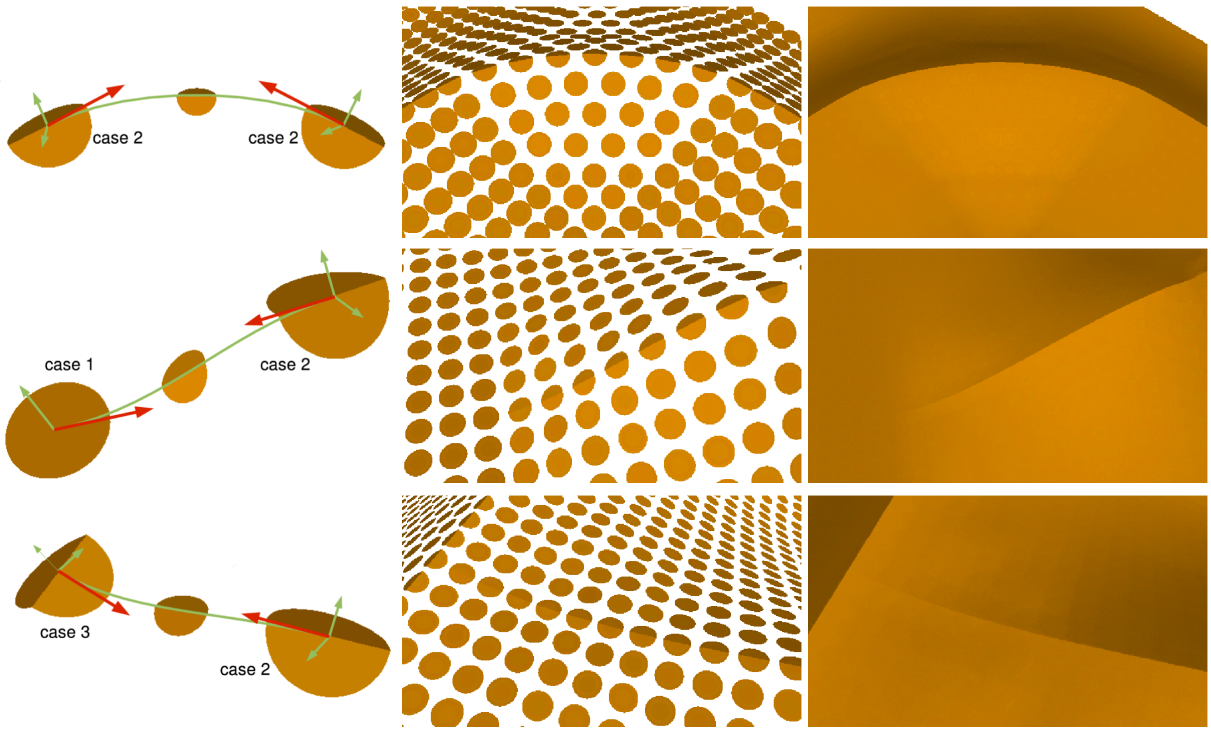


Fig. 5.19 – Illustration du raffinement des arêtes.

5.6.3 Détection et interpolation des bords

Puisque nous utilisons une représentation uniforme pour le marquage des bords et des arêtes nous pouvons réutiliser, à quelques différences près, le même principe qu'exposé précédemment. La première différence est qu'un bord est détecté entre deux voisins uniquement s'il s'agit de deux splats de bordure, c'est-à-dire uniquement dans le cas 2-2.

Lorsqu'une bordure est détectée entre deux points, celle-ci est reconstruite par une courbe cubique de Bézier comme pour notre opérateur de lissage ϕ_2 (section 5.2.1.1). Pour les vecteurs tangents de cette courbe nous pouvons au choix :

- Utiliser les vecteurs tangents obtenus par projection sur les plans tangents (comme pour deux splats normaux).

- Utiliser, comme pour les arêtes, les vecteurs définissant les lignes de découpe. Cela permet en plus un lissage de la discontinuité dans le plan de la surface.

5.6.4 Application au raffinement diadique

La prise en compte des arêtes (et bords) avec une stratégie de raffinement diadique est très facile puisque la règle de base est d'insérer un nouveau point entre chaque voisin. Cependant, après le regroupement des voisins nous devons vérifier pour chaque groupe s'il n'y a pas d'arête traversant le polygone associé au groupe. Si tel est le cas, l'arête étant prioritaire, seul un nouveau point sur l'arête sera inséré pour le groupe.

Lorsque qu'un nouveau point est inséré entre deux points formant une arête, celui-ci étant situé sur une arête doit avoir deux normales. Ses deux normales sont obtenues de la même manière qu'avec notre opérateur de lissage ϕ_2 (voir équation 5.5) en considérant les normales des extrémités de l'arête deux à deux.

5.6.5 Application au raffinement $\sqrt{3}$

La prise en compte des arêtes (et bords) avec une stratégie de raffinement $\sqrt{3}$ est par contre beaucoup plus problématique puisque les nouveaux points ne sont jamais directement insérés entre deux points.

Dans le cas du raffinement $\sqrt{3}$ des maillages [Kob00], la solution de Kobbelt est d'utiliser un traitement spécial pour les bords et arêtes franches :

- De telles arêtes ne sont jamais modifiées lors de la phase de remaillage.
- Tous les deux pas de subdivision (les pas pairs), deux points sont insérés pour chaque arête du bord et arête franche.

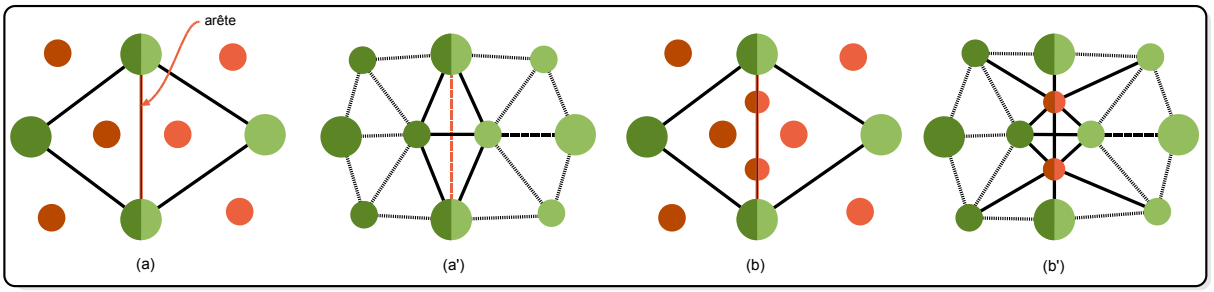


FIG. 5.20 – Problème du raffinement des arêtes avec le raffinement $\sqrt{3}$. (a) Pas de raffinement impaire : une arête est détectée mais aucun point n'est inséré sur l'arête. (a') Pas de raffinement suivant : l'arête n'est plus détectée. (b) Pas de raffinement impaire : une arête est détectée et deux points supplémentaires sont insérés en avance sur l'arête. (b') Pas de raffinement suivant : les relations de voisinage sont affectés par l'insertion prématurée des points de l'arête.

Le problème avec une représentation par points est qu'il n'y a pas de lien de connectivité. Après un pas de raffinement, deux points voisins formant une arête ont de fortes chances de ne plus être détectés comme voisins et aucun traitement spécial ne pourra être effectué entre eux (figure 5.20-a). L'arête sera donc lissée. Dans le cas d'un bord, celui-ci sera creusé de manière fractale (figure 5.22-a). Comme nous ne pouvons pas attendre un pas pair pour raffiner les arêtes, l'idée est de raffiner les arêtes en avance, c'est-à-dire lors des pas impairs (figure 5.20-b).

Cependant, comme le montre la figure 5.20-b', afin de ne pas affecter la sélection des voisins lors du pas de raffinement suivant (pas pair), les points ainsi calculés en avance ne doivent pas être pris en compte pour le calcul des voisinages.

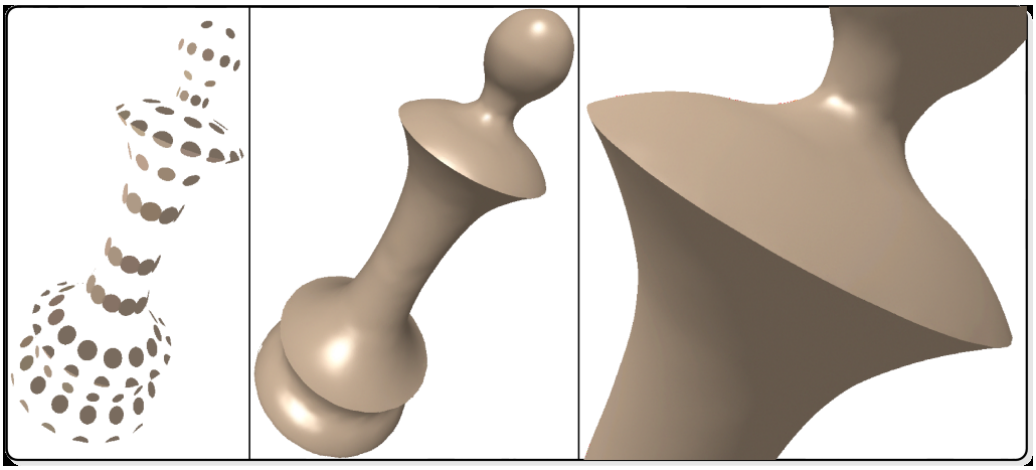


FIG. 5.21 – Illustration du raffinement des arêtes avec le $\sqrt{3}$.

5.6.6 Marquage automatique des bords et arêtes

Les arêtes franches sont généralement issues d'outils de modélisation tels que les opérations CSG [PKKG03, AD03] et peuvent donc être connues *à priori*. Lorsque cela n'est pas le cas il peut être utile de les détecter automatiquement. Pour cela, Fleishman et al. [FCOS05] ont récemment proposé une méthode pour détecter très précisement des arêtes au sein d'un nuage de points qui peut même être bruité. Leur méthode est cependant assez lourde à mettre en oeuvre et requiert un nuage de points suffisement dense.

Lorsque les informations de normales sont connues et que le nuage de points n'est pas bruité, une heuristique simple pour détecter les arêtes est de définir un angle maximal entre les plans tangents de deux points voisins. Concrètement, si l'angle entre les normales de deux points voisins $\mathbf{p}_0$ et $\mathbf{p}_1$ est supérieur à un seuil donné (e.g. $\frac{\pi}{2}$), alors un nouveau point est inséré sur l'arête qui vient d'être détectée, c'est-à-dire sur l'intersection des deux plans tangents et de manière à ce qu'il soit le plus proche de $\mathbf{p}_0$ et $\mathbf{p}_1$.

Le marquage des bords est particulièrement important pour le raffinement $\sqrt{3}$ qui, le cas échéant, génère des bords fractals (figure 5.22-a). Pour le raffinement diadique cette détection automatique n'est pas crucial excepté pour obtenir un rendu propre de ceux-ci. La notion de bords est en fait directement liée à la définition du voisinage utilisé : un point $\mathbf{p}$ est un "splat de bordure" s'il existe deux voisins dans N_p consécutifs $\mathbf{p}_i$ et $\mathbf{p}_{i+1}$ qui ne sont pas voisins entre eux, c'est-à-dire si l'angle $\widehat{\mathbf{p}_i\mathbf{p}\mathbf{p}_{i+1}}$ est supérieur à l'angle θ_1 utilisé par notre relation de voisinage (voir section 5.3.2 et figure 5.22-b).

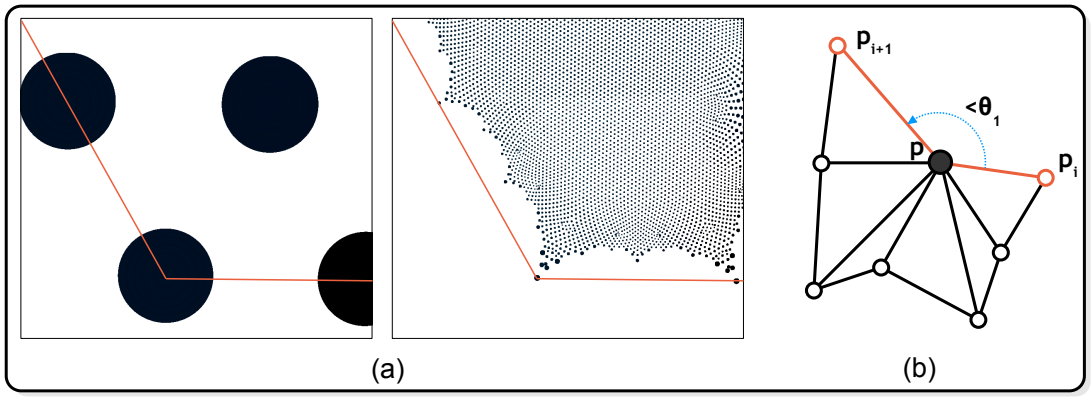


FIG. 5.22 – (a) Raffinement naïf d'un bord avec une stratégie $\sqrt{3}$. (b) Détection d'un bord dans le voisinage du point **p**.

5.7 Structures de données

5.7.1 Recherche des plus proches voisins

Le calcul du premier anneau de voisinage, et plus spécifiquement la recherche des plus proches voisins, est une partie critique de notre algorithme du point de vue du coût de calcul. Cette recherche est généralement accélérée par une structure de données partitionnant finement l'espace telle qu'une simple grille 3D ou un *kd*-tree. Cependant, aucune de ces structures de données n'est réellement adaptée à notre situation. Pour satisfaire nos besoins, une telle structure de données devrait avoir les propriétés suivantes :

- Fin partitionnement de l'espace : environ 10 points par cellule.
- Dynamique : mise à jour (insertion et suppression) rapide lors du raffinement.
- Multi-résolution : séparation claire des différents niveaux. Ce point est important car cela permet un raffinement purement local ainsi que la recherche des nouveaux points déjà insérés (nécessaire à l'étape de contrôle de l'échantillonnage).

Nous proposons de construire dynamiquement une hiérarchie de sphères englobantes en associant à chaque point $\mathbf{p} \in P^l$ l'ensemble des points de P^{l+1} insérés durant son propre raffinement. Ces points sont appelés fils du point **p**. Chaque noeud de la hiérarchie est donc simplement un point du nuage de points : le centre de la sphère est directement la position du point tandis que son rayon est déduit du rayon du point. L'analyse de algorithme de raffinement (section 5.10.1) montre que prendre pour le rayon de la sphère le double du rayon du point associé au noeud garantit que tous les fils seront bien contenus dans la sphère. Le coût de stockage de notre hiérarchie de sphère englobante est alors quasi nul puisque, si nous faisons l'hypothèse que les points sont insérés de manière séquentielle dans un tableau, alors uniquement 4 octets supplémentaires par point sont nécessaires : 3 octets pour stocker l'indice du premier fils et un octet pour stocker le nombre de fils.

Grâce à notre algorithme de contrôle de l'insertion des nouveaux points, le nombre de fils d'un point ne peut excéder 8, un codage plus optimal mais également légèrement plus coûteux à manipuler, consisterait à prendre 29 bits pour l'indice du premier fils et 3 bits pour le nombre de fils. Cela permet de porter le nombre maximal de points par objet de 16 millions (2^{24}) à plus de 518 millions ! Il est également important de remarquer que le coût de construction de notre

structure de données est nul.

La recherche des nouveaux points déjà insérés est triviale, nous avons juste à tester les fils des voisins du point courant. La recherche des plus proches voisins dans le niveau l, c'est-à-dire des points de P^l dans une boule de centre $\mathbf{p}$ et de rayon r, est réalisée en parcourant récursivement la hiérarchie de la sphère englobante :

```
List<Indice> N_p^ε = ∅;

procedure rechercher(Indice i, Entier niveau_courant)
{
    si niveau_courant+1 == l alors
        pour chaque fils p_j du point p_i faire
            si ‖ p − p_j ‖ < r alors
                insérer(N_p^ε, j);
            finsi
    sinon
        pour chaque fils p_j du point p_i faire
            si ‖ p − p_j ‖ − 2 r_j < r alors
                rechercher( j,niveau_courant + 1 );
            finsi
    finsi
}
```

Cependant, nous avons besoin d'une structure de données additionnelle afin de trouver les plus proches voisins dans le niveau initial ainsi que pour initialiser le processus de recherche dans la hiérarchie de sphères englobantes. Pour cela, n'importe quelle structure de données *classique* peut être utilisée. Pour notre part, nous utilisons une simple grille 3D très rapide à construire bien qu'un *kd*-tree soit plus efficace lorsque la densité du nuage de points varie de manière importante.

Du point de vue de l'efficacité de la recherche des plus proches voisins, notre hiérarchie de sphères englobantes n'est pas optimale. En effet, les cellules sont loin d'être ajustées au plus juste (centre et rayon des sphères) et les cellules d'un niveau donné se recouvrent largement. De plus, le nombre de fils par noeud n'est pas équilibré puisque les premiers points raffinés ont en moyenne 7 fils tandis que les derniers n'ont qu'un fils, eux-mêmes, car leur voisinage a été raffiné durant le raffinement de leur voisins. Malgré cela, notre hiérarchie de sphères englobantes reste très intéressante par rapport aux autres solutions existantes et présente trois avantages majeurs : coût de construction nul, coût de stockage additionnel très faible et multi-résolution.

5.8 Application au rendu temps-réel

L'objectif initial de notre algorithme de raffinement était d'améliorer la qualité du rendu par splatting en augmentant dynamiquement la densité de points lorsque cela est nécessaire. Bien évidement, les performances de notre algorithme ne permettent pas de raffiner à chaque image le nuage de points dans son entier et jusqu'à ce que la taille des points soit inférieure au pixel. Nous

devons au contraire ne raffiner que les zones visibles et réellement sous-échantillonnées pour le point de vue courant. La sélection des points à raffiner est donc équivalente à un processus de rendu multi-résolution tel que présenté à la section 2.3.2.

La hiérarchie de sphères englobantes présentée à la section précédente nous fournit déjà une structure multi-résolution dynamique de notre nuage de points qui pourrait être utilisée dans ce but. Cependant, plusieurs travaux ont déjà montré qu'une structure de données trop fine n'est pas adaptée pour ce type d'applications puisque les meilleures performances sont généralement atteintes pour un nombre d'environ 1000 points par cellule.

Nous avons donc choisi d'utiliser une seconde structure de données pour cette étape de sélection des noeuds à raffiner. Octree ou *kd*-tree sont classiquement utilisés. Si l'octree est particulièrement bien adapté à un raffinement diadique, aucune structure de données classique n'est adaptée au raffinement $\sqrt{3}$. Nous proposons donc une nouvelle structure de données plus flexible et permettant de s'adapter à n'importe quelle stratégie de raffinement.

Finalement, les points générés lors du raffinement doivent être stockés temporairement dans un cache.

5.8.1 Subdivision spatiale adaptative et dynamique

Le point de départ de notre structure de données multi-résolution dynamique est un partitionnement du nuage de points initial sous la forme d'un ensemble de boîtes englobantes alignées aux axes. Optionnellement, ces volumes englobants peuvent correspondre aux feuilles d'une structure de données hiérarchique permettant la gestion des niveaux de détails en cas de réduction et d'effectuer des tests de visibilité hiérarchiques sur les faibles niveaux. Cette structure de données peut être un octree, un *kd*-tree ou encore la hiérarchie de boîtes englobantes que nous allons décrire.

Notre hiérarchie de boîtes englobantes étant construite dynamiquement lors du processus de raffinement, nous allons décrire son principe en décrivant le raffinement d'un noeud.

Afin d'être compatible avec notre algorithme de raffinement et de permettre des tests de visibilités efficaces ainsi qu'une sélection des niveaux de détail locale, notre hiérarchie de volumes englobants doit satisfaire les contraintes suivantes :

- Le nombre de points par noeud doit être constant. Appelons M cette constante. Un ordre de grandeur est 1000 points par cellule.
- Par souci de compacité et de rendu efficace, le stockage des points d'un noeud devrait être séquentiel.
- Les points doivent être stockés en mémoire vidéo dans un VBO partagé par tous les noeuds.
- Chaque niveau dans la hiérarchie correspond à un niveau de raffinement différent.
- Les fils d'un point devant être stockés séquentiellement, ceux-ci doivent appartenir au même noeud.
- Le volume englobant associé à un noeud doit être ajusté au plus juste tout en contenant les splats entièrement (pas seulement leur centre).

Afin de garantir un nombre constant de points par noeud (ou cellule), lorsqu'un noeud est raffiné celui-ci est subdivisé en un nombre de sous-cellules qui est fonction du nombre total de points N' de ces sous-cellules. Bien sûr, nous ne pouvons connaître ce nombre total avant d'avoir raffiné les points du noeud. Il est toutefois possible d'en connaître une bonne approximation à partir du nombre de points N du noeud courant et de la stratégie de raffinement utilisée. Lorsque le nuage de points est localement assez uniforme, alors le nombre de points est multiplié à chaque pas de raffinement par quatre pour le raffinement diadique et par trois pour le raffinement $\sqrt{3}$.

Le nombre de sous-cellules k est donc donné par :

$$k = \left\lfloor \frac{N'}{M} + \frac{1}{2} \right\rfloor \tag{5.46}$$

Si $k \leq 1$ alors le noeud n'est pas subdivisé et a un seul fils. Si $k = 2$ ou $k = 3$ alors la cellule est subdivisée uniformément en deux ou trois le long de sa plus grande dimension. Si $k \geq 4$ alors la cellule est subdivisée en deux le long de ses deux plus grandes dimensions. En fait, les cas où k serait supérieur à quatre sont très improbables avec nos schémas de raffinement.

Afin de limiter les coûts de stockage et de permettre une transmission rapide au GPU des points à tracer, nous avons choisi de stocker les points d'un noeud de manière séquentielle dans un tampon unique partagé par tous les noeuds. Afin d'accélérer les temps de rendu, une copie de ce tampon est stockée en mémoire vidéo.

En pratique, un noeud de notre hiérarchie est donc assez compact puisqu'il est composé de :

- Une boîte englobante alignée aux axes (deux points 3D).
- Un scalaire représentant l'espacement moyen entre les points du noeud (utilisé au moment du rendu pour évaluer si la densité est suffisante ou non).
- Les indices du premier et du dernier point.
- Quatre pointeurs sur les fils.

Finalement, afin de satisfaire le stockage séquentiel des points, un noeud est raffiné de la manière suivante :

```
procedure raffiner_noeud ( Noeud n )
{
    subdiviser le noeud n en k sous-cellules;

    pour chaque sous-cellule S_j faire
        pour chaque point p_i du noeud n faire
            si p_i ∈ S_j alors
                raffiner_point(p_i);
            finsi
        mettre à jour le volume englobant de S_j;
}
```

Comme indiqué dans cet algorithme, les volumes englobants des sous-cellules doivent être mis à jour, c'est-à-dire contractés et/ou dilatés après l'insertion des nouveaux points afin de garantir que la boite englobante contient tous les splats de la cellule dans leur totalité, tout en étant minimale.

5.8.2 L'algorithme de rendu progressif

Notre algorithme de rendu progressif est initialement basé sur un algorithme de rendu multi-résolution classique tel que celui que nous avions présenté dans l'état de l'art (voir l'algorithme présenté section 2.3.2). La principale différence est que maintenant, si un noeud est une feuille de l'arborescence mais n'est pas suffisement dense, alors le noeud est raffiné et le parcours récursif est appliqué sur ses nouveaux fils.

Nous proposons en plus un certain nombre d'optimisations du parcours de la hiérarchie :

- Afin de garantir une visualisation temps-réel, nous proposons de borner le temps alloué au processus de rendu. Ceci implique que le parcours de la hiérarchie doit pouvoir être stoppé à tout moment tout en fournissant une image complète. Le parcours doit donc être réalisé en largeur d'abord.
- Il est inutile d'effectuer les tests de visibilité sur la descendance d'un noeud entièrement ou presque entièrement visible.
- Les intervalles d'indices sont assemblés au fur et à mesure afin de limiter le coût de stockage de la liste des points à tracer et d'accélérer le transfert de cette liste vers le GPU.

Une version simplifiée de l'algorithme de rendu global est finalement :

```
FIFO<Noeud> file;

// Initialisation de la pile avec la/les racines de la hiérarchie
pour chaque noeud racine n faire
    insérer_en_queue(file, n);

tant que file n'est pas vide faire
    Noeud n = dépiler(file);

    si n est potentiellement visible faire
        si le temps alloué est écoulé
        ou la densité des points de n est suffisante alors
            tracer les points de n;
        sinon
            si n est une feuille alors
                raffiner_noeud(n);
            finsi

            pour chaque fils f du noeud n faire
                insérer_en_queue(file, f);
        finsi
    finsi
```

5.8.3 Gestion du cache

Jusqu'à présent nous n'avons vu que l'aspect insertion de notre algorithme dynamique. Lors du raffinement d'un noeud, de nouveaux points sont créés et stockés séquentiellement dans un cache. Cependant, la mémoire disponible n'étant pas infinie, il est nécessaire de mettre en place un mécanisme pour supprimer du cache les points qui ne sont plus nécessaires.

Pour cela, tous les noeuds créés par raffinement sont aussi stockés dans une liste. En pratique, chaque élément de la liste stocke une référence vers un noeud et chaque noeud stocke une référence sur un élément de la liste. Lors du parcours de la hiérarchie, à chaque fois qu'un noeud est tracé (ou qu'une de ces descendances est tracée) alors celui-ci est déplacé en fin de

liste. Lorsqu'un noeud doit être raffiné et que le cache est plein, alors les points du noeud situé en tête de la liste sont supprimés du cache. Le noeud est également supprimé de la hiérarchie et de la liste. Bien sûr, s'il ne s'agit pas d'une feuille de l'arborescence, alors toute sa descendance doit également être supprimée.

Ce mécanisme est la fois très simple à mettre en oeuvre et très efficace puisque toutes les opérations requises s'effectuent en temps constant ($O(1)$).

5.8.4 Simplifications - Optimisations

La technique de raffinement que nous avons présentée, a été conçue pour permettre la prise en compte de nuages de points sous-échantillonnés et irréguliers. Cependant, lorsque le nuage de points est assez bien échantillonné, ce qui est toujours le cas après un ou deux pas de raffinement car les points se rapprochent du plan tangent du point interpolé (section 5.10.1)), il est possible d'utiliser quelques simplifications augmentant significativement les performances :

1. Approcher "l'angle courbe" par l'angle géométrique classique :
   $\widetilde{A}_{\mathbf{p}}(\mathbf{p}_0, \mathbf{p}_1) = \widehat{\mathbf{p}_0\mathbf{p}\mathbf{p}_1}$

2. Approcher la distance géodésique par la distance euclidienne :
   $\widetilde{G}(\mathbf{p}_0, \mathbf{p}_1) = \|\mathbf{p}_0\mathbf{p}_1\|$

3. Utiliser la version symétrique de notre voisinage qui évite l'étape de "projection géodésique".

4. Approcher la position des nouveaux points par les simples centres de gravité (lors du calcul de L_p). L'opérateur de lissage est donc appliqué uniquement si le point doit réellement être inséré.

En pratique, ces quelques simplification permettent de presque doubler les performances.

5.9 Résultats

Qualité visuelle

La première stratégie que nous avons mise en oeuvre est la stratégie de raffinement diadique. Bien qu'améliorant considérablement la qualité d'un rendu par splatting, comme le montre la figure 5.24, nous avons assez vite abandonné cette approche à cause des quelques artefacts apparaissant sur les surfaces générées comme par exemple sur la figure 5.23-c. Le raffinement $\sqrt{3}$ offre en effet un raffinement de bien meilleure qualité, avec une surface très lisse visuellement comme le montre la figure 5.23. Sur cette figure, nos deux schémas de raffinement sont également comparés au butterfly modifié qui est une surface de subdivision interpolante C^1 [ZSS96] :

- Bien qu'étant garanti C^1, le butterfly génère une surface avec de larges oscillations.
- Notre raffinement diadique est globalement plus lisse (moins d'oscillation) mais présente des artefacts de hautes fréquences.
- Bien que nous n'ayons pû montrer que notre raffinement $\sqrt{3}$ soit au moins G^1 continu (nous n'avons pas non plus montré le contraire, voir section 5.10.1), la surface générée est visuellement parfaitement lisse.

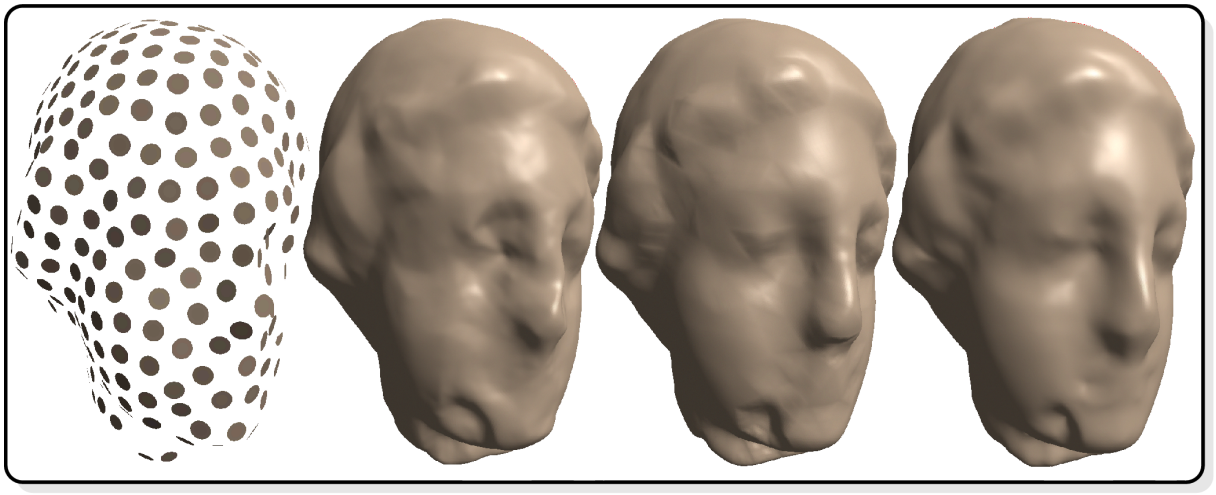


FIG. 5.23 – Comparaison de la qualité de l'interpolation. De gauche à droite : Le visage de Igea échantillonné par 600 points. Résultat de l'interpolation avec le Butterfly (après triangulation du nuage de points). Notre raffinement diadique et enfin notre raffinement $\sqrt{3}$.

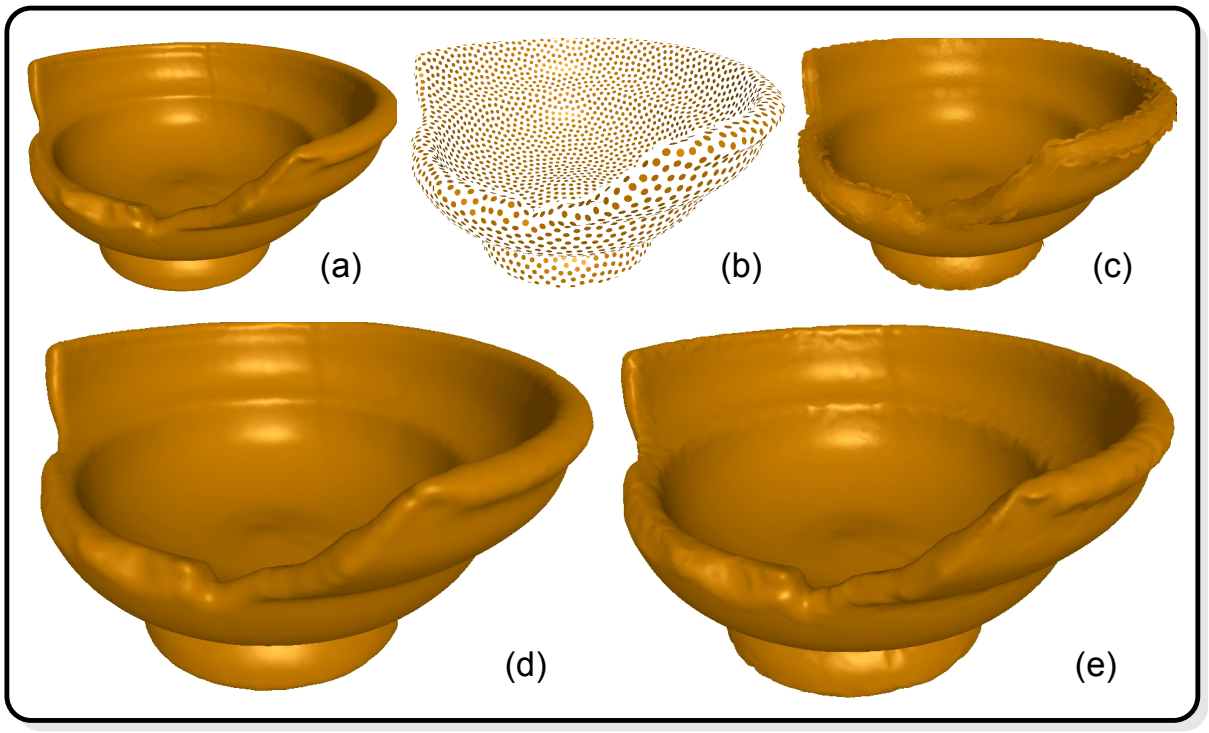


FIG. 5.24 – Un modèle de poterie de 300k points (a) sous échantillonné à 5.2k points (b) et (c). Raffiné à 335k points avec notre méthode (d) et par le schéma de subdivision butterfly (e).

Notre raffinement $\sqrt{3}$ présentant une surface d'une bien meilleure qualité visuelle que notre raffinement diadique, et par conséquent la suite des résultats ne concerne que notre raffinement $\sqrt{3}$.

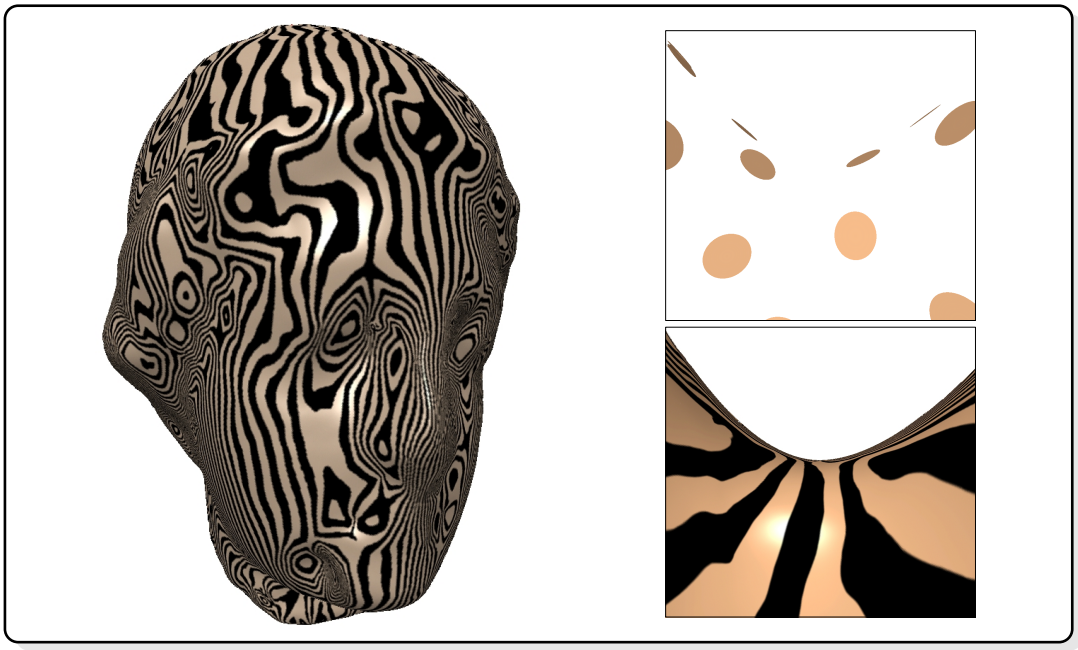


FIG. 5.25 – Mise en évidence des variations de normales et de courbures par des lignes de reflexions. À droite zoom sur le raffinement d'une configuration en "selle".

La qualité observée des surfaces générées par notre raffinement $\sqrt{3}$ est renforcée par les figures 5.25 qui ont été obtenues par application d'une texture d'environnement contenant des rayures. Cette dernière permet de mettre en évidence les éventuelles discontinuités de normales (G^1) et de courbures (G^2) d'une surface. Sur ces images aucune discontinuité dans les lignes de réflexion n'est décelable.

Tout au long de ce chapitre nous nous somme principalement intéressé à la géométrie des objets. La figure 5.26 illustre le raffinement d'un objet texturé : la couleur d'un nouveau point est simplement interpolée linéairement à partir des points du groupe qui lui est associé.

Évaluation de la robustesse

Afin d'évaluer la robustesse de notre méthode, nous l'avons testée sur plusieurs modèles de points dont la densité a été irrégulièrement réduite par une sélection aléatoire. Par exemple, sur la figure 5.1, 3500 points ont été sélectionnés aléatoirement à partir d'un nuage de 150 000 points représentant une statue d'Isis. Malgré un fort sous-échantillonnage et une répartition irrégulière des points, nos algorithmes de sélection de voisinage et de contrôle de l'échantillonnage sont suffisement robustes pour générer une surface sans trous, lisse et uniformément échantillonnée.

L'exemple de la figure 5.27 illustre la reconstruction d'un large trou dans un modèle complexe qu'est la chevelure d'une statue de David. Bien sûr, pour que le trou soit bouché par notre algorithme, les rayons des splats au bord du trou ont été ajustés afin que les splats recouvrent le trou.

La figure 5.28 illustre un cas réel (l'oeil du David) où à la fois notre projection géodésique et notre mesure d'angle courbe sont indispensable à la reconstruction de la surface sans trou.

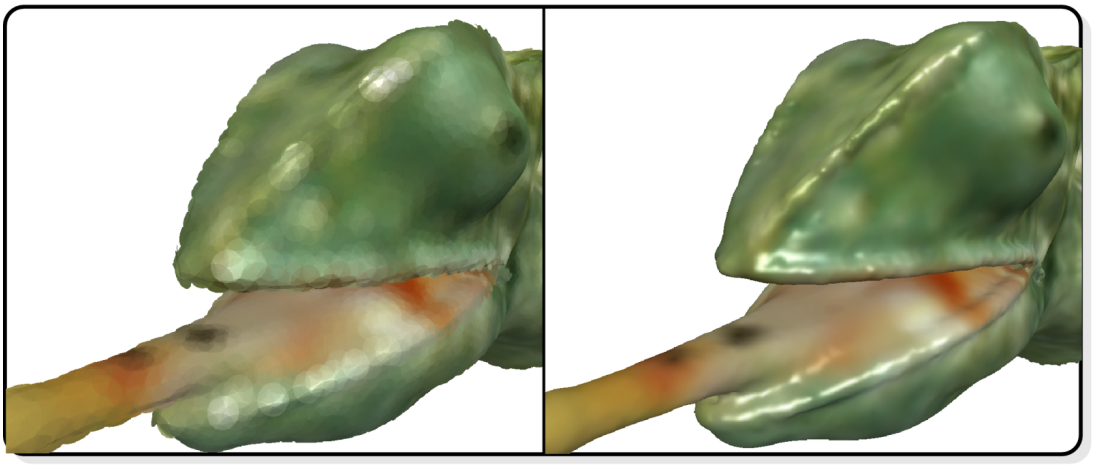


FIG. 5.26 – Raffinement d'un objet texturé.
(a) Splatting sans raffinement (et sans interpolation des normales). (b) Même technique de splatting, après 4 pas de raffinement.

Notons, que nous avons dû désactiver les tests de voisinage sur l'angle entre les normales et le filtrage par co-cône (section 5.3.2) qui sont trop restrictifs pour ce cas. La projection géodésique est indispensable ici pour ordonner correctement les voisins tandis que l'angle courbe permet à l'algorithme de contrôle de l'échantillonnage "d'autoriser" l'insertion de nouveaux points entre des points proches mais ayant des orientations très différentes.

Performances

Nous avons testé notre implémentation sur un Athlon 3500+ disposant d'une carte graphique NVidia GeForce 6800 et de 512Mo de mémoire vive. Nous avons évalué que notre algorithme est capable de générer entre 350k et 450k points par seconde en fonction de la régularité du nuage de points. Les quelques approximations/optimisations que nous avons présentées précédemment permettent d'atteindre un taux de 700k points par seconde.

Lorsque notre algorithme de raffinement est utilisé au dessus de notre pipe-line de rendu et que le temps alloué au rendu est borné de manière à garantir un taux d'au moins 24 images par seconde, alors le temps restant pour le raffinement dynamique est assez faible puisque nous devons soustraire le temps pris par le splatting et le temps pris par le parcours de la structure de données. Aussi, nous avons mesuré que pour notre système complet, un taux de génération de points de 300k points seconde était un minimum. Dans le cas contraire, il est généralement nécessaire de stopper la navigation de temps en temps pour permettre au raffinement de terminer son travail. Le tableau ci-dessous résume les coûts relatifs de chacune des parties de notre algorithme :

Recherche des plus proches voisins (N_p^ϵ)	23%
Filtrage du voisinnage par BSP flou	40%
Contrôle de l'échantillonnage	29%
Interpolation et mise à jour des rayons	8%

TAB. 5.1 – Coûts relatifs des parties de notre algorithme de raffinement.

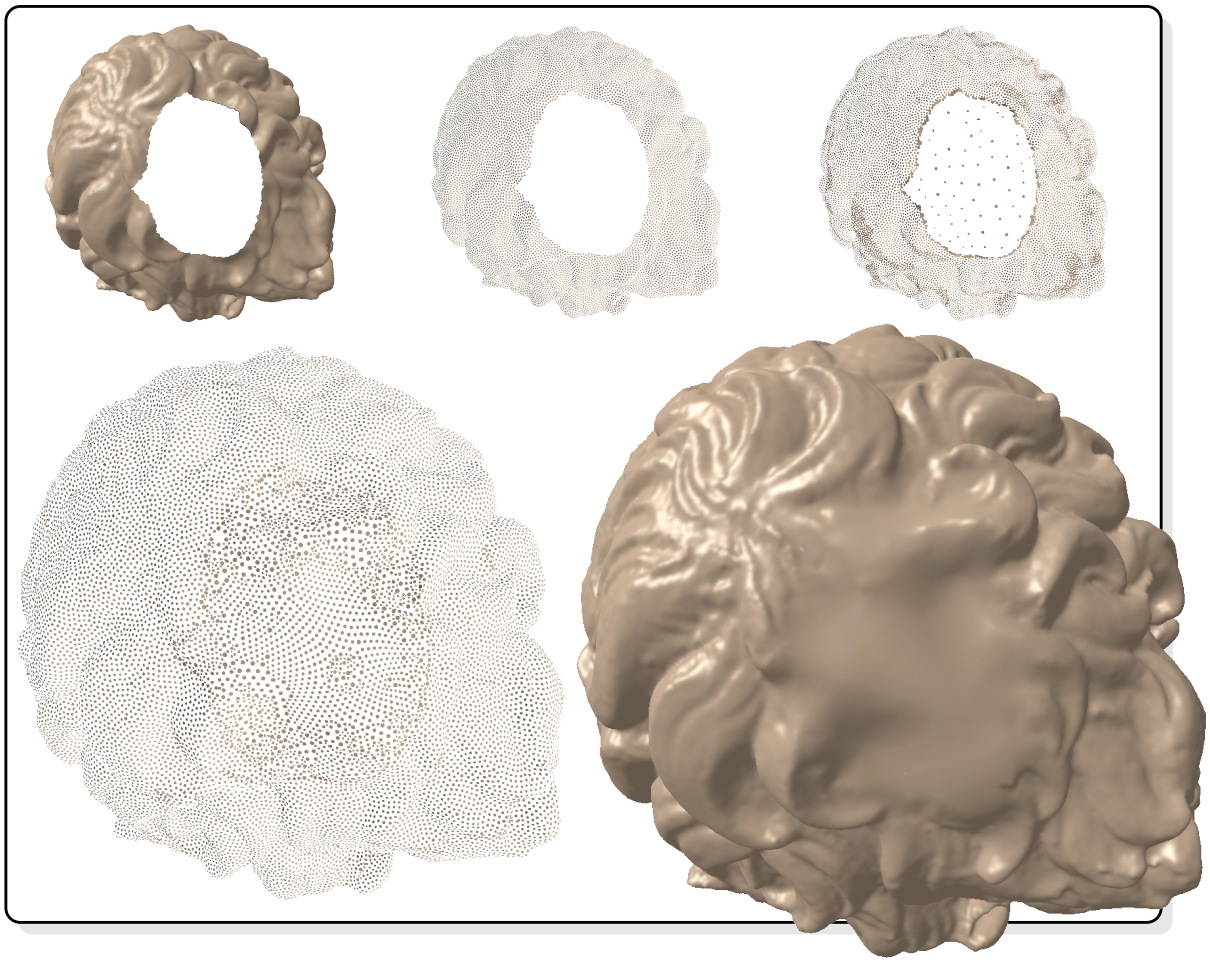


FIG. 5.27 – Reconstruction d'un large trou dans la chevelure du modèle de David.

5.10 Discussions à propos de notre raffinement $\sqrt{3}$

5.10.1 Analyse de notre raffinement $\sqrt{3}$

Nos résultats expérimentaux, démontrent à la fois la robustesse de notre algorithme à prendre en compte des nuages de points éparpillés et une très bonne qualité, lisseur, de la surface générée (dans le cas de notre raffinement $\sqrt{3}$). Puisque notre algorithme de raffinement est une méthode de reconstruction de nuage de points basée sur un processus itératif générant une surface, au delà de ces résultats expérimentaux, trois questions de nature théorique reste en suspens :

- Quelles sont les conditions sur l'échantillonnage du nuage de points initial pour assurer une reconstruction cohérente ?
- Le processus itératif converge t-il, c'est-à-dire existe t-il une surface limite ?
- Si oui, quelle est la continuité de la surface limite ?

L'analyse rigoureuse de notre algorithme de raffinement est cependant très difficile, principalement à cause de son caractère heuristique, et de sa dépendance envers l'ordre de traitement des points. De plus toutes les méthodes d'analyse de surfaces limites développées ces dernières décennies pour les maillages polygonaux ne peuvent s'appliquer aux nuages de points.

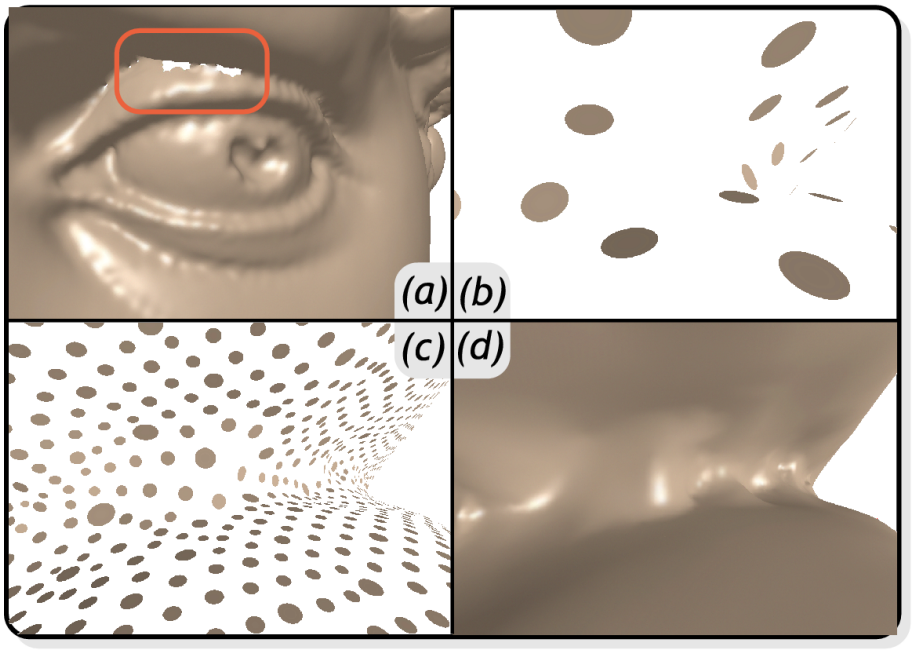


FIG. 5.28 – Intérêt de notre distance géodésique et de notre angle courbe : sans ceux-ci notre algorithme est incapable de reconstruire cette zone de forte courbure (a).

Néanmoins, dans la suite de cette section nous proposons quelques pistes de départ pour une analyse de la surface limite de notre raffinement $\sqrt{3}$.

Fixons un point $\mathbf{p} \in P^{l_0}$. Soit N_p^l, $l \geq l_0$ la suite du voisinage du point $\mathbf{p}$ et r^l la suite des rayons du point $\mathbf{p}$. Rappelons que r^l correspond à la distance maximale entre $\mathbf{p}$ et un de ses voisins représentés par N_p^l.

Convergence locale. Tout d'abord, nous pouvons montrer que la suite r^l a pour limite zéro lorsque l tend vers l'infini, c'est-à-dire que la suite des points insérés autour du point $\mathbf{p}$ a pour limite $\mathbf{p}$. En effet, une borne supérieure du rayon r^{l+1} est donnée par la distance entre $\mathbf{p}$ et le plus éloigné de ses nouveaux voisins $\mathbf{p}_k \in P^{l+1}$, $k \in N_p^{l+1}$ qui est au centre d'un triangle de Bézier formé par $\mathbf{p}$ et deux de ses voisins de N_p^l, qui sont eux même situés à une distance de $\mathbf{p}$ inférieur à r^l. D'après l'équation 5.9, nous avons :

$$r^{l+1} = \|\mathbf{p} - \mathbf{p}_k\| = \|\mathbf{p} - \frac{1}{6}(\mathbf{b}_{210} + \mathbf{b}_{120} + \mathbf{b}_{021} + \mathbf{b}_{012} + \mathbf{b}_{102} + \mathbf{b}_{201})\| \tag{5.47}$$

Puisque :

$$\|\mathbf{p} - \frac{1}{2}(\mathbf{b}_{210} + \mathbf{b}_{120})\| \leq \frac{5}{6}r^l \ , \quad \|\mathbf{p} - \frac{1}{2}(\mathbf{b}_{201} + \mathbf{b}_{102})\| \leq \frac{5}{6}r^l$$
$$\|p - \frac{1}{2}(\mathbf{b}_{021} + \mathbf{b}_{012})\| \leq (\cos(\beta/2) + \frac{2}{3}\sin(\ \beta/2))r^l \tag{5.48}$$

Nous avons :

$$r^{l+1} \leq c_\beta r^l \tag{5.49}$$

$$c_\beta = \frac{1}{3}(\frac{5}{6} + \frac{5}{6} + (\cos(\beta/2) + \frac{2}{3}\sin(\ \beta/2))) \tag{5.50}$$

où β est l'angle entre le plus éloigné des voisins de $\mathbf{p}$ et son successeur (ou prédécesseur). Quel que soit la valeur de β, nous avons $c_\beta < 0.96$ qui est strictement inférieur à 1 et donc la limite de la suite r^l est zéro. Bien sûr, cela permet juste de prouver que la suite des points insérés autour du point $\mathbf{p}$ ont pour limite $\mathbf{p}$, ce qui est fondamentale. Cependant, nous rappelons que ceci n'est pas suffisant pour prouver que l'on converge vers une surface, par exemple, une surface fractale est un contre-exemple.

Continuité locale. Soit θ^l la suite des angles maximaux que forment les voisins N_p^l avec le point $\mathbf{p}$ et le plan tangent de $\mathbf{p}$. En d'autres termes, cet angle défini aussi un co-cône (complémentaire d'un cône, illustré figure 5.9) d'apex $\mathbf{p}$, d'axe $\mathbf{n}$ et d'angle $\frac{\pi}{2} - \theta^l$ contenant l'ensemble des voisins du point $\mathbf{p}$. Cette région est définie par l'ensemble :

$$\left\{ \mathbf{x} \in \mathbb{R}^3 \mid \left| \frac{\mathbf{x} - \mathbf{p}}{\|\mathbf{x} - \mathbf{p}\|} \right| < cos\left(\frac{\pi}{2} - \theta^l\right) \right\} \tag{5.51}$$

Nous pouvons continuer l'analyse de la suite des points formant le voisinage du point $\mathbf{p}$ fixé, en analysant la suite des angles θ^l, θ^{l+1}, ...

Pour cela nous faisons les deux hypothèses suivantes :

1. Il existe un l_0 à partir duquel les triangles de Bézier construits ne contiennent aucun point d'inflexion.
2. Le nouveaux voisinage du point $\mathbf{p}$ n'est composé que de nouveaux points chacun issu de l'insertion d'un point sur un triangle de Bézier.

Alors, nous pouvons trouver, comme précédemment, une constante μ strictement inférieur à 1 telle que :

$$\theta^{l+1} < \mu\theta^l \; \forall \, l \geq l_0 \tag{5.52}$$

La seconde hypothèse est bien sûr en partie fausse puisque le nouveau voisinage d'un point peut comporter des points de l'ancien voisinage, comme nous l'avons vue section 5.5. Aussi, il est plus juste d'écrire :

$$\theta^{l+1} < \mu_l\theta^l \; \forall \, l \geq l_0 \tag{5.53}$$

avec soit $\mu_l = 1$ soit $\mu_l < \mu$. La suite des rayons r^l du point $\mathbf{p}$ étant contractante, nous pouvons affirmer que si $\mu_l = 1$ alors il existe un entier $k > 0$ tel que $\mu_{l+k} < \mu$. Cela signifie, que si l'hypothèse qu'il existe un l_0 à partir duquel les triangles de Bézier construits ne contiennent aucun point d'inflexion peut être vérifiée alors le voisinage du point $\mathbf{p}$ converge vers $\mathbf{p}$ de manière G^1. Finalement, si cette hypothèse est vérifiée et que nous réussissions à prouver la convergence, c'est-à-dire l'existence d'une surface limite, alors cette surface serait bien G^1.

Pour prouver la convergence globale, une approche possible est de prouver que la suite des distances maximales entre les nouveaux points $P^{l+1} - P^l$ et l'approximation C^{-1} par les splats de P^l est une suite contractante. Le résultat précédent semble aller dans ce sens, malheureusement le chevauchement possible des triangles de Bézier rend à la fois l'analyse de cette suite et la vérification de l'existence d'un l_0 (hypothèse 1) extrêmement compliquée.

5.10.2 Application aux maillages triangulaires

Dans les paragraphes précédents, nous avons vu que l'analyse de la convergence vers une surface limite de notre algorithme de raffinement était rendu très difficile par l'absence de l'existence d'un schéma se reproduisant à l'identique et par les chevauchements des triangles de Bézier. De plus nous avons observé que notre méthode donnait visuellement de meilleurs résultats que les surfaces de subdivisions interpolantes existantes. Nous pouvons donc nous demander s'il ne serait possible d'appliquer notre méthode de raffinement interpolant $\sqrt{3}$ aux maillages triangulaires.

Le principe est assez simple puisque nous n'avons qu'une seule et unique règle de subdivision : pour chaque triangle du maillage, construire un triangle de Bézier à partir des trois sommets et insérer un nouveau point en son centre. Les anciens points restent inchangés. Après l'insertion de tous les nouveaux points, les sommets sont remaillés comme pour le $\sqrt{3}$ approximant (figure 5.16).

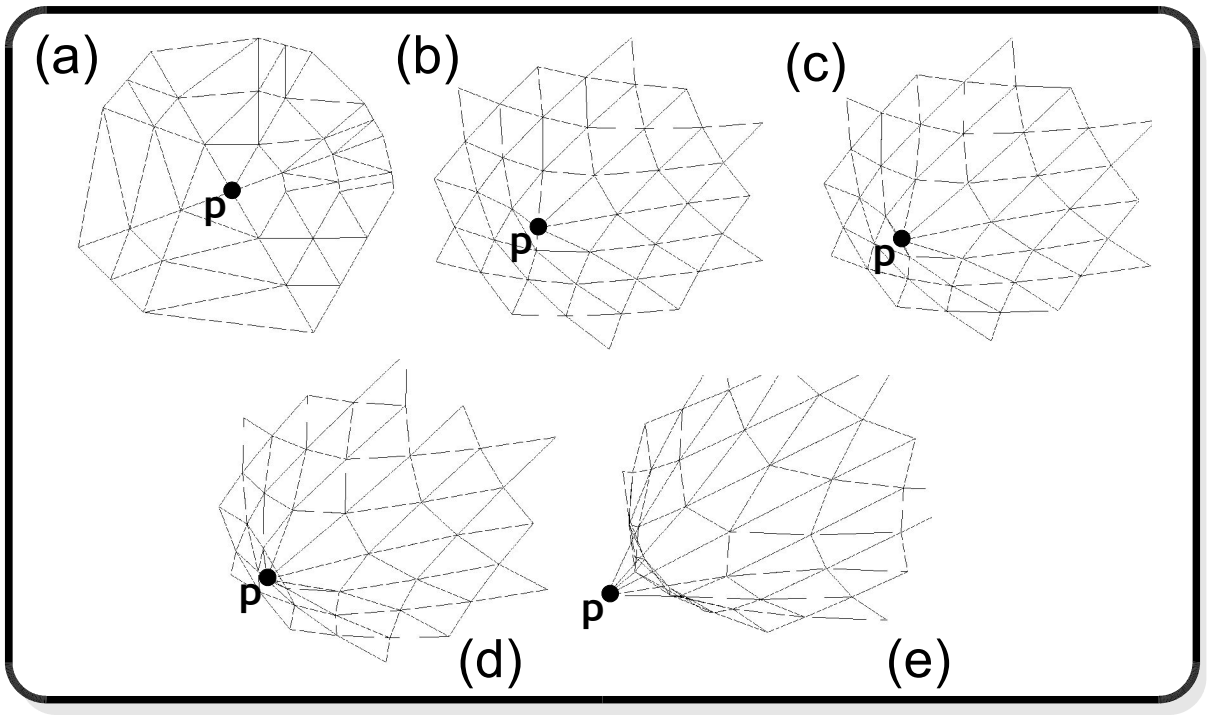


FIG. 5.29 – Application de notre raffinement $\sqrt{3}$ interpolant sur un maillage plan. Chaque image correspond au même maillage initial (a), après différents pas de subdivision. À chaque pas de subdivision, les bords sont supprimés et le maillage mis à l'échelle par $\sqrt{3}$ afin de conserver une image lisible. (b) après 5 pas de subdivision. (c) 7 pas. (d) 9 pas. (e) 12 pas.

Malheureusement, bien que potentiellement convergent, ce schéma à un très mauvais comportement de convergence, comme le montre la figure 5.29. De plus celui-ci ne peut pas être C^1. En effet, en considérant un maillage plan où la normale de chaque sommet est la normale du plan, la règle d'insertion d'un nouveau point est considérablement simplifiée puisque cela revient à insérer les nouveaux points aux centres de gravité des triangles. Dans ces conditions il est très facile d'analyser ce schéma, par exemple en utilisant la méthode décrite dans [DS78, Cla78].

Analyse. Le raffinement autour d'un point $\mathbf{p}$ de valence 6 (cas régulier de la subdivision sur un maillage triangulaire) peut s'écrire sous forme matricielle de la manière suivante :

$$\begin{bmatrix} \mathbf{q} \\ \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \\ \mathbf{q}_4 \\ \mathbf{q}_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 1/3 & 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 1/3 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 1/3 & 1/3 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 1/3 & 1/3 \\ 1/3 & 1/3 & 0 & 0 & 0 & 0 & 1/3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p} \\ \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{p}_4 \\ \mathbf{p}_5 \end{bmatrix} \tag{5.54}$$

$$\mathbf{Q} = \mathbf{S} \cdot \mathbf{P} \tag{5.55}$$

Ici, les points $\mathbf{p}_i$ correspondent au premier anneau de voisinage du point $\mathbf{p}$, le point $\mathbf{q}$ correspond au point $\mathbf{p}$ après un pas de raffinement ($\mathbf{p}$ et $\mathbf{q}$ sont identiques car le schéma est interpolant) et les points $\mathbf{q}_i$ correspondent aux points insérés, c'est-à-dire au nouveau anneau de voisinage du point $\mathbf{p}$.

Une décomposition en valeurs propres et vecteurs de la matrice de subdivision $\mathbf{S}$, nous permet d'écrire le raffinement sous la forme :

$$\mathbf{Q} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1} \cdot \mathbf{P} \tag{5.56}$$

où $\mathbf{V}$ est la matrice des vecteurs propres et $\mathbf{\Lambda}$ la matrice diagonale des valeurs propres associées. En triant les sept valeurs propres λ_i de manière à ce que $|\lambda_i| \leq |\lambda_{i+1}| \; \forall i in [0..5]$, nous obtenons :

$$|\lambda_0| = 1\,,\ |\lambda_1| = \tfrac{2}{3}\,,\ |\lambda_2| = \tfrac{1}{\sqrt{3}}\,,\ |\lambda_3| = \tfrac{1}{\sqrt{3}}$$
$$|\lambda_4| = \frac{1}{3}\,,\ |\lambda_5| = \frac{1}{3}\,,\ |\lambda_6| = 6.7951e-18$$

Convergence. Nous pouvons remarquer que nous avons bien $|\lambda_0| = 1$ et $|\lambda_i| < 1 \; \forall i \in [1..5]$, ce qui est une condition nécessaire de convergence. De plus, l'écriture du raffinement autour de $\mathbf{p}$ sous la forme de l'équation 5.56 permet d'obtenir facilement les positions des points $\mathbf{q}$ et $\mathbf{q}_i$ après une infinité de pas de subdivisions. Après vérification par le calcul, tous ces points convergent bien sur le point initial $\mathbf{p}$.

Continuité. Du point de vue de la continuité, une condition nécessaire pour que la surface limite soit C^1 est que $|\lambda_1| = |\lambda_2|$ 'car le schéma de subdivision est symétrique). Cette condition n'est malheureusement pas satisfaite et cela permet d'expliquer l'étrange comportement de la convergence mis en évidence par la figure 5.29. En fait, une analyse en fréquence permet d'identifier que la valeur propre λ_1 est celle contrôlant la composante elliptique de la courbure. Pour assurer un bon comportement du schéma, son module devrait être inférieur aux modules des deux valeurs propres suivantes (λ_2 et λ_3) qui elles contrôlent la convergence au niveau du plan tangent. Ainsi, intuitivement, nous pouvons observé que les points $\mathbf{q}_i$ du voisinage convergent plus rapidement sur eux même (sur leur centre de gravité) que sur le point central $\mathbf{p}$.

Ce très mauvais comportement ne peut se produire dans notre cas de raffinement des nuages de points. Cela est principalement dû à notre sélection du premier anneau de voisinage qui n'est pas basé sur une topologie et grâce à notre procédure d'insertion qui optimise l'équilibre

du voisinage. Aussi, nous pouvons nous demander s'il ne serait tout de même pas possible d'appliquer notre principe de raffinement interpolant aux maillages triangulaire, en modifiant l'étape de re-maillage afin de maintenir un équilibrage constant comme le fait naturellement notre algorithme.

5.11 Conclusion

Voisinages Dans ce chapitre nous avons présenté un algorithme de raffinement itératif des nuages de points. Pour cela nous avons dans un premier temps proposé deux nouvelles méthodes de calcul de voisinage : la première, prenant en compte les spécificités d'un nuage de points orientés, fournit un premier anneau de voisinage particulièrement précis, tandis que la deuxième a comme avantage majeur d'être naturellement symétrique. Bien sûr, celles-ci ne sont pas limitées à notre algorithme de raffinement et de nombreuses autres méthodes de traitement des nuages de points pourraient tirer profis des avantages de nos deux nouveaux voisinages. Du point de vue des perspectives de recherche, il serrait intéressant de combiner entre elles nos deux méthodes de voisinage afin de profiter de leur deux avantages respectifs : symétrie et robustesse (grâce à la prise en compte des orientations des points). Une analyse rigoureuse des critères d'échantillonnage, que doivent satisfaire le nuage de points pour garantir une reconstruction consistante, serait également souhaitable. Cependant, l'établissement d'un critère d'échantillonnage est rendu très difficile par la prise en compte des normales des points. De plus, il n'existe toujours pas de théorie de l'échantillonnage satisfaisante pour les surfaces manifold discrète. Les quelques critères qui ont été proposés pour d'autre type de reconstruction sont donc généralement très pessimistes et donc peu utilisable en pratique.

Stratégies de raffinement et interpolation Au dessus de ces calculs de voisinage nous avons construit notre algorithme de raffinement itératif interpolant. Nous avons vu que parmis les deux stratégies de raffinement différentes que nous avons proposées, notre raffinement $\sqrt{3}$ offre une bien meilleure qualité de surface que notre raffinement diadique. Ce dernier présente donc peu d'intérêt en pratique. Pourtant, une approche diadique présenterait quelques avantages, comme une prise en compte des bords et arêtes simplifiée et une analyse de la convergence vers une surface limite sans doute plus aisée puisque le raffinement diadique ne présent pas le problème du $\sqrt{3}$ mis en évidence à la section 5.10.2. Notre approche diadique est également moins sensible à toute connectivité arbitraire grâce au regroupement des points voisins. Les artefacts que nous avons remarqués avec notre stratégie diadique pourrai s'expliquer par l'utilisation mixte de règles d'interpolation univariées et bivariées, sans réussir à obtenir de meilleur résultats. Finalement, l'association d'une interpolation par triangle de Bézier à un raffinement $\sqrt{3}$ apparaît donc comme un outil vraiment efficace pour la reconstruction de surface lisse interpolante.

Pour résumer nous allons lister les principaux points forts et points faibles de notre algorithme de raffinement $\sqrt{3}$. Notre algorithme de raffinement étant initialement inspiré des surfaces de subdivisions, nous apporterons dans cette énumération quelques points de comparaisons.

Les points forts :

- Interpolation visuellement très lisse et générant moins d'oscillations que les surfaces de subdivisions interpolantes existantes [DLG90, Kob96, ZSS96].
- Contrôle aisé de la normale de la surface générée. Notons, que certaines surfaces de subdivisions approximantes offrent également ce type de contrôle sur la normale.

- Particulièrement bien adapté au raffinement local puisque, contrairement aux surfaces de subdivisions, nous n'avons pas de problème de gestion de la connectivé.
- Très faible sensibilité aux artefacts polaires en cas de forte valence [SB03]. Cet avantage est principalement dû au fait que nous ne nous soucions pas de la connectivité et que notre algorithme de contrôle de l'échantillonnage interdit l'insertion d'un trop grand nombre de points autour du point raffiné. Rappelons, que ces points de valence non régulière posent aux surfaces de subdivisions de nombreux problèmes de continuité et d'oscillation.
- Performance suffisante pour être inséré dans un *pipe-line* de rendu temps-réel.
- Prise en compte des bords et arêtes franches.

Les points faibles :

- Comme toute méthode de reconstruction de nuage de points désorganisés, le nuage de points initial doit satisfaire un certains nombre de critères. Malheureusement nous n'avons pas encore réussi à établir ces critères de manière précise. Notons que, les surfaces de subdivision n'ont aucun problème de robustesse par rapport à l'échantillonnage initial puisque la connectivité est fournie.
- Bien que nos résultats expérimentaux nous laissent penser que nos surfaces générées sont très certainement au moins G^1 continue dans la majeur partie des cas, nous n'avons pas encore réussi à finaliser une analyse rigoureuse de la convergence vers une surface limite lisse. De ce point de vue, les surfaces de subdivisions, étudiées depuis plus de trente ans, disposent de nombreux outils d'analyse de la convergence et de la continuité. Ainsi, les surfaces limites de certaines surfaces de subdivisions sont connues explicitement. De plus, notons que certains schémas de subdivision générèrent des surfaces (approximante) dont le degré de continuité est bien supérieur à C^1 (normales) ou C^2 (courbures).
- Bien que notre approche soit globalement indépendante d'une quelconque topologie, la surface générée est tout de même dépendente d'une connectivité locale nécessaire à la construction de nos triangles de Bézier. De plus, cette surface est également dépendante de l'ordre de traitement des points.

Autres applications Nous n'avons pour l'instant proposé qu'une seule application de notre algorithme de raffinement, à savoir le rendu temps-réel de haute qualité. Bien que notre algorithme de raffinement a été conçu dans ce but, il est clair qu'une telle approche est intéressante pour bien d'autres types d'application. Par exemple, nous suggérons la transmission de géométrie à travers un réseau ou la modélisation multi-résolution à base de splats. Dans le premier cas, les nuages de points ont déjà montré leur simplicité et supériorité par rapport aux maillages triangulaire puisqu'il n'y pas d'information topologique à transmettre. En fonctions des capacités du serveur, de la ligne de transmition et du client, nous pouvons imaginer deux scénarios. Si le client réalisant l'affichage n'est pas assez puissant pour effectuer un rendu par splatting de qualité (PDA ou téléphones portable), alors une implantation de notre algorithme de raffinement sur le serveur permettra de garantir une densité de points suffisante pour permettre un rendu trivial par le client. Au contraire si le client est suffisement puissant pour accueillir notre algorithme de raffinement (n'importe quel PC récent) alors ce dernier sera capable de rendre une image de qualité en attendant que les détails de la géométrie soient transmis. Cette dernière configuration pose le problème de la prise en compte efficace des détails alors que le nuage de points à déjà été raffiné par le client.

Chapitre 6

Conclusion et perspectives

6.1 Résumé de notre *pipeline*

En introduction nous avions présenté nos travaux sous la forme d'un *pipe-line* de rendu à base de points très simplifié (figure 1.1). Dans cette conclusion générale, nous sommes maintenant en mesure de présenter une nouvelle vue de ce *pipe-line*, plus détaillée et plus proche de l'organisation générale réelle, synthétisant nos différentes contributions, figure 6.1.

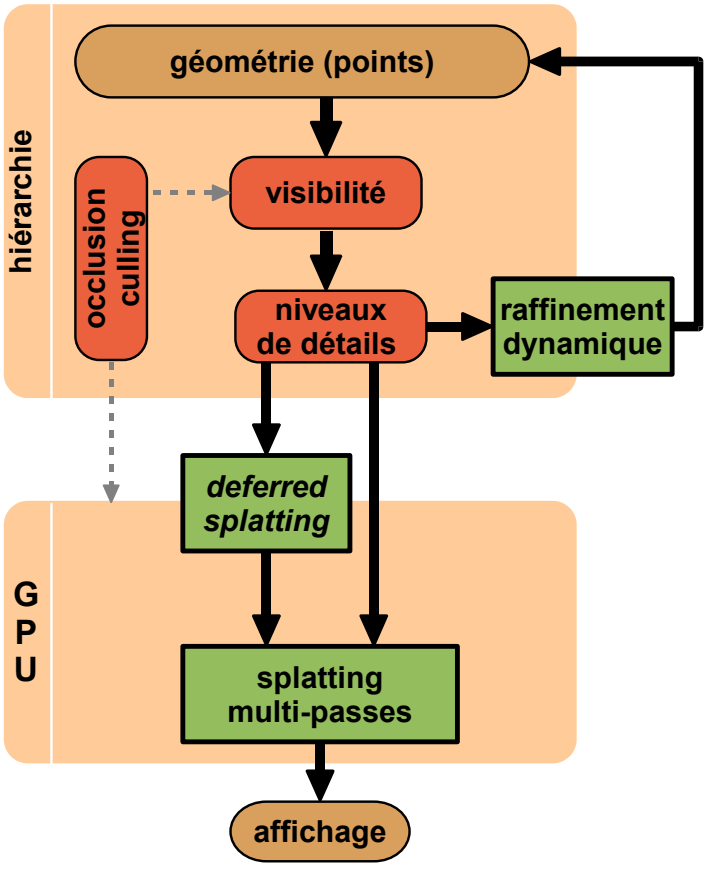


FIG. 6.1 – Notre pipe-line de rendu par points complet.

La vie d'un point au sein de ce *pipe-line* de rendu commence par un regroupement des points les plus proches spatialement au sein d'une structure de données hiérarchique et dynamique. Le rendu d'une image peut alors commencer. Cette structure de données est parcourue noeud par noeud. Pour chaque noeud nous commençons par tester sa visibilité via des tests de fenêtrage et d'élimination des faces arrières classiques. Grâce au *deferred splatting*, nous pouvons aussi faire effectuer par la carte graphique un test d'occlusion (section 4.3.1). Si le résultat de tous ces tests est positif alors nous testons si la densité des points du noeud courant est suffisante ou non :

- Si la réponse est oui, alors les points du noeud sont envoyés à l'étage de *deferred splatting* (chapitre 4) qui se chargera de sélectionner efficacement les points réellement visibles (point par point). Les points passant ce test sont finalement envoyés à l'étage de rendu par splatting de haute qualité (chapitre 3).
- Si la réponse est non, et que nous n'avons plus de temps disponible pour améliorer la qualité du rendu en traçant plus de points (garantie du temps-réel), alors les points sont directement envoyés à l'étage de rendu par splatting puisqu'ils sont trop gros pour être traités par le *deferred splatting*. Dans le cas contraire, alors les fils du noeud courant sont placés dans une file, et seront traités ultérieurement (non visible sur le schéma, section 5.8.2). Cependant, si le noeud courant est une feuille de l'arborescence, alors de nouveaux points sont créés en raffinant ce noeud par notre algorithme de raffinement dynamique (chapitre 5). Les noeuds ainsi créés sont mis dans la file d'attente.

Remarquons que nous avons présenté ici notre *pipeline* dans sa version la plus efficace, c'est-à-dire en s'appuyant sur une structure de données multi-résolution. Cependant, aussi bien notre algorithme de splatting que notre méthode de *deferred splatting* sont parfaitement utilisables sans cela. Notre sélection de bas-niveau par *deferred splatting* est d'ailleurs d'autant plus utile que le processus de sélection de haut-niveau est grossier. Finalement, si la scène ne contient pas trop de points, alors ceux-ci peuvent être rapidement (quelques millisecondes) plongés dans une grille 3D pour permettre le raffinement dynamique, lui-même générant dynamiquement une structure de données multi-résolution.

6.2 Principales contributions

Dans un premier temps, nous nous sommes interessés au développement d'une **méthode de rendu à base de points par splatting** permettant d'exploiter au mieux les performances accrues des dernières cartes graphiques tout en conservant une très bonne qualité de l'interpolation et du filtrage des attributs. Notre méthode de rendu accéléré par la carte graphique supporte également les objets semi-transparents et un rendu hybride points/polygones de qualité puisque nous avons proposé une méthode pour filtrer les transitions.

Les cartes graphiques actuelles ne permettant pas une implantation optimale du rendu des nuages de points et malgré toutes les optimisations que nous avons apportées à notre implantation, le rendu d'un point reste plus coûteux que le rendu d'un triangle. Or, dans le cadre du rendu de scènes complexes, de nombreux travaux ont montré l'intéret d'une représentation à base de points ou d'une représentation hybride points/polygones. Dans ce domaine d'application, notre principale contribution est notre **algorithme de *deferred splatting***. Grâce à une sélection très précise des points à tracer, celui-ci permet de rendre les coûts du splatting de qualité indépendants de la complexité de la scène. Ainsi, la visualisation temps-réel de scènes

complexes et le rendu par points de qualité deviennent donc parfaitement compatibles.

Hors du cadre de la problématique de visualisation, nous avons ensuite proposé une nouvelle **méthode de calcul de voisinage** prenant en compte les spécificités d'une représentation à base de points orientés. Comparativement aux méthodes précédentes, la prise en compte des normales des points dans le processus de sélection d'un voisinage permet en effet de rendre cette sélection bien plus robuste et précise. Bien sûr, nous faisons ici l'hypothèse que les normales sont connues et correctes.

En nous appuyant sur ce calcul de voisinage précis, nous avons développé un **algorithme de raffinement itératif de nuages de points**. Les principaux atouts de notre méthode sont la génération d'une surface visuellement lisse et interpolante, et sa rapidité. La qualité d'un rendu par splatting étant dépendante de la densité du nuage de points, notre algorithme de raffinement est donc ici particulièrement intéressant, puisqu'il permet de maintenir en temps-réel une densité de points suffisante à un rendu d'une qualité constante.

6.3 Discussions et perspectives de recherche

En plus des différentes discussions et perspectives de recherche que nous avons developpés dans chacune des conclusions des chapitres précédents, nous proposont ici quelques perspectives de recherche d'ordre plus général.

Ombres douces. Comme nous l'avons vu, une approche par *deferred shading* permet de nombreuses possibilités au niveau des effets visuels qu'il est possible de réaliser. Cependant, à l'heure actuelle, les rares méthodes de rendu en temps-réel des ombres douces qui sont applicables aux représentations par points, c'est-à-dire les méthodes basées images, sont encore très limitées : soit seule la pénombre externe est prise en compte, soit les ombres générées souffrent de nombreux artefacts. Le rendu temps-réel des ombres douces pour les géométries basées points reste donc un problème ouvert.

Splats à orientations multiples. Dans le cadre de la visualisation de scènes complexes, les problèmes que peuvent poser une extrême simplification de la géométrie ont été peu abordés. En effet, une heuristique communément utilisée lors de la construction des faibles niveaux de détails, est de ne pas mélanger entre eux des points ayant des orientations trop diférentes. Malgré cela, la perte d'information est trop importante et la qualité de l'éclairement peut être grandement diminuée. Une piste serait d'associer aux points une distribution de normales représentant l'ensemble des orientations de la surface représentées par le point. Bien qu'initialement proposés pour améliorer la qualité du rendu en cas d'agrandissement, les points-différentiels, approchant la surface par des morceaux d'ellipsoïdes, définissent implicitement une distribution de normales qui pourrait être intégrée dans les calculs de l'éclairement. Une alternative, offrant plus de flexibilité, serait d'utiliser le modèle de réflectance de LaFortune [LFTG97]. Le challenge est alors d'intégrer cette nouvelle représentation dans la procédure de ré-échantillonnage et d'anti-aliasing du splatting.

Les géométries simples. Malgré tous leurs avantages, les géométries à base de points restent conceptuellement assez peu adaptées à la représentation d'objets simples (e.g. une boite) ayant une texture très détaillée. Paramétriser l'ensemble du nuage de points pour y appliquer une texture 2D est bien sûr possible, mais cela serait au sacrifice d'un avantage majeur des points qui, comme l'avons fait remarquer à mainte reprises, est justement l'absence d'une paramétrisation et la non décorrélation de la géométrie et de la texture. Une solution intermédiaire serait peut-être d'utiliser une application de texture locale, via une paramétrisation locale. Cela n'enlèverait en aucun cas les avantages que peut avoir une représentation par splats, mais reste à étudier la faisabilité d'un point de vue pratique ainsi que les limites d'une telle approche. Remarquons que les limitations sont sans doute assez proches de celles des méthodes de placage de texture par *texture sprite* [LHN05].

Bibliographie

[AA03a] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 245–254, 2003.

[AA03b] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *SMI '03 : Proceedings of the Shape Modeling International 2003*, page 272, Washington, DC, USA, 2003. IEEE Computer Society.

[AA04a] Anders Adamson and Marc Alexa. Approximating bounded, non-orientable surfaces from points. In *Proceedings of Shape Modeling International 2004*, 2004.

[AA04b] Marc Alexa and Anders Adamson. On normals and projection operators for surfaces defined by point sets. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 149–155, June 2004.

[ABCO+01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *VIS '01 : Proceedings of the conference on Visualization '01*, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society.

[ABCO+03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surface. *IEEE Transaction on Visualization and Computer Graphics*, 9(1) :3–15, 2003.

[ABK98] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoï-based surface reconstruction algorithm. In *Proceedings of ACM SIGGRAPH 98*, 1998.

[AD03] Bart Adams and Philip Dutré. Interactive boolean operations on surfel-bounded solids. *ACM Trans. Graph.*, 22(3) :651–656, 2003.

[AGPS04] Mattias Andersson, Joachim Giesen, Mark Pauly, and Bettina Speckmann. Bounds on the k-neighborhood for locally uniformly sampled surfaces. In *Proceedings of Eurographics Symposium on Point-Based Graphics*, 2004.

[AK04] Nina Amenta and Yong J. Kil. Defining point set surfaces. In *Proceedings of ACM SIGGRAPH 2004, Computer Graphics Proceedings*, 2004.

[AKP+05] Bart Adams, Richard Keiser, Mark Pauly, Leonidas J. Guibas, Markus Gross, and Philip Dutré. Efficient raytracing of deforming point-sampled surfaces. In *Proceedings of Eurographics 2005*, 2005.

[AWD+04] Bart Adams, Martin Wicke, Phil Dutré, Markus Gross, Mark Pauly, and Matthias Teschner. Interactive 3d painting on point-sampled objects. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 57–66, 2004.

[Ba02] Pat Brown and al. Gl_arb_vertex_program extention, 2002. http ://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex-program.txt.

[BCD+99] Christoph Bregler, Michael F. Cohen, Paul Debevec, Leonard McMillan, FranÃ§ois X. Sillion, and Richard Szeliski. Image-based modeling, rendering, and lighting. In *SIGGRAPH 99 Course Notes*, 1999.

[BHZK05] Mario Botsch, Alexander Hornung, Matthias Zwicker, and Leif Kobbelt. High-quality surface splatting on today's gpus. In *Proceedings of Point-Based Graphics 2005*, pages 17–24, 2005.

[BK03] Mario Botsch and Leif Kobbelt. High-Quality Point-Based Rendering on Modern GPUs. In *11th Pacific Conference on Computer Graphics and Applications*, pages 335–343, 2003.

[BK04] Mario Botsch and Leif Kobbelt. Phong splatting. In *Proceedings of Point-Based Graphics 2004*, pages 25–32, 2004.

[BK05] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. In *Proceedings of Eurographics 2005*, 2005.

[Bli82] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82 : Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 21–29, New York, NY, USA, 1982. ACM Press.

[BWK02] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 53–64, 2002.

[Car84] Loren Carpenter. The a -buffer, an antialiased hidden surface method. In *SIGGRAPH '84 : Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 103–108, New York, NY, USA, 1984. ACM Press.

[Cat74] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, University of Utah, Salt Lake City, december 1974.

[CAZ01] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiang Zhang. Hybrid simplification : combining multi-resolution polygon and point rendering. In *VIS '01 : Proceedings of the conference on Visualization '01*, pages 37–44, Washington, DC, USA, 2001. IEEE Computer Society.

[CBC+01] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001*, pages 67–76, 2001.

[CH02] Liviu Coconu and Hans-Christian Hege. Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 43–52, 2002.

[CHJ03] C. S. Co, B. Hamann, and K. I. Joy. Iso-splatting : A Point-based Alternative to Isosurface Visualization. In J. Rokne, W. Wang, and R. Klein, editors, *Proceedings of Pacific Graphics 2003*, pages 325–334, October 8–10 2003.

[CHP+79] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, and M. Howard. Towards an interactive high visual complexity animation system. In *SIGGRAPH '79 : Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 289–299, New York, NY, USA, 1979. ACM Press.

[CK03] Jatin Chhugani and Subodh Kumar. Budget sampling of parametric surface patches. In *SI3D '03 : Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 131–138, New York, NY, USA, 2003. ACM Press.

[Cla78] E. Catmull Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6) :350–355, 1978.

[CN01] Baoquan Chen and Minh Xuan Nguyen. Pop : a hybrid point and polygon rendering system for large data. In *VIS '01 : Proceedings of the conference on Visualization '01*, pages 45–52, Washington, DC, USA, 2001. IEEE Computer Society.

[COCSD03] Daniel Cohen-Or, Yiorgos Chrysanthou, Claudio T. Silva, and Fréo Durand. A survey of visibility for walkthrough applications. *IEEE TVCG*, pages 412–431, 2003.

[CRT04] Ulrich Clarenz, Martin Rumpf, and Alexandru Telea. Surface processing methods for point sets using finite elements. *Computers & Graphics*, 28 :851–868, 2004.

[DCSD01] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of IEEE Visualization*, pages 37–44, 2001.

[DCSD02] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of the IEEE Visualization Conference*. IEEE, October 2002.

[DD04] Florent Duguet and George Drettakis. Flexible point-based rendering on mobile devices. *IEEE Computer Graphics and Applications*, 24(4), July-August 2004.

[DLG90] N. Dyn, D. Levin, and J. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transaction on Graphics, 9 (2)*, pages 160–169, 1990.

[DS78] D. Doo and M.A. Sabin. Analysis of the behaviour of recursive subdivision surfaces near extraordinary points. *Computer Aided Design*, 10(6) :356–360, 1978.

[DVS03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In *Proceedings of ACM SIGGRAPH 2003*, pages 657–662, 2003.

[Far02] Gerald Farin. *CAGD a pratical guide*. Academic Press, 5 edition, 2002.

[FCOAS03] Shachar Fleishman, Daniel Cohen-Or, Marc Alexa, and Claudio T. Silva. Progressive point set surfaces. *ACM Trans. Graph.*, 22(4) :997–1011, 2003.

[FCOS05] Shachar Fleishman, Daniel Cohen-Or, and Claudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3) :544–552, 2005.

[FR01] M. S. Floater and M. Reimers. Meshless parameterization and surface reconstruction. *Comp. Aided Geom. Design 18*, pages 77–92, 2001.

[GBP04a] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Deferred splatting. In *Proceedings of Eurographics 2004*, 2004.

[GBP04b] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Dynamic surfel set refinement for high-quality rendering. *Computers & Graphics*, 28 :827–838, 2004.

[GBP04c] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time point cloud refinement. In *Proceedings of Eurographics Symposium on Point-Based Graphics*, pages 41–48, 2004.

[GBP05] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Interpolatory refinement for real-time processing of point-based geometry. In *Proceedings of Eurographics 2005*, 2005.

[GD98] J. P. Grossman and William J. Dally. Point sample rendering. *Rendering Techniques 98*, pages 181–192, 1998.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96 : Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM Press.

[GH86] Ned Greene and Paul S. Heckbert. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Comput. Graph. Appl.*, 6(6) :21–27, 1986.

[GKM93] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *SIGGRAPH '93 : Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238, New York, NY, USA, 1993. ACM Press.

[GM04] Enrico Gobbetti and Fabio Marton. Layered point clouds : a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28 :815–826, 2004.

[GP03] Gaël Guennebaud and Mathias Paulin. Efficient screen space approach for Hardware Accelerated Surfel Rendering. In *Proceedings of Vision, Modeling and Visualization*, pages 41–49. IEEE Signal Processing Society, 2003.

[Gro98] J. P. Grossman. Point sample rendering. Master's thesis, Massachusetts Intitute of Technology, 1998.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuezle. Surface reconstruction from unorganized points. In *Proceedings of ACM SIGGRAPH 92*, 1992.

[Hec86] Paul S. Heckbert. Survey of texture mapping. *IEEE Comput. Graph. Appl.*, 6(11) :56–67, 1986.

[Hec89] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California at Berkley, 1989.

[HSLM02] K. Hillesland, B. Salomon, A. Lastra, and D. Manocha. Fast and simple occlusion culling using hardware-based depth queries. Technical report, Department of Computer Science, University of North Carolina, 2002.

[HXC04] Xiaoru Yuan Hui Xu, Minh X. Nguyen and Baoquan Chen. Interactive silhouette rendering for point-based models. In *Proceedings of Eurographics Symposium on Point-Based Graphics*, 2004.

[JC99] Norman P. Jouppi and Chun-Fa Chang. Z3 : an economical hardware technique for high-quality antialiasing and transparency. In *HWWS '99 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 85–93, New York, NY, USA, 1999. ACM Press.

[KB04] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28 :801–814, 2004.

[KBR04] John Kessenich, Dave Baldwin, and Randi Rost. The opengl shading langage, april 2004. http ://opengl.org/documentation/oglsl.html.

[Kob96] Leif Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Proceedings of Eurographics 1996*, 1996.

[Kob00] Leif Kobbelt. $\sqrt{3}$ subdivision. In *Proceedings of ACM SIGGRAPH 2000*, 2000.

[Kri03] Jaroslav Krivanek. Representing and rendering surfaces with points. Technical Report DC-PSR-2003-03, Department of Computer Science and Engineering, Czech Technical University in Prague, 2003.

[KV01] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 139–150, London, UK, 2001. Springer-Verlag.

[KV03] Aravind Kalaiah and Amitabh Varshney. Modeling and rendering of points with local geometry. *IEEE Transactions on Visualization and Computer Graphics*, 9(1) :30–42, 2003.

[La02] Benj Lipchak and al. Gl_arb_fragment_program extention, 2002. http ://oss.sgi.com/projects/ogl-sample/registry/ARB/fragment-program.txt.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes : a high resolution 3d surface reconstruction algorithm. In *Proceedings of ACM SIGGRAPH 1987*, pages 163–169, 1987.

[Lev01] D. Levin. Mesh-independent surface interpolation. In *Advances in Computational Mathematics*, 2001.

[Lev03] D. Levin. Mesh-independent surface interpolation. In *Geometric Modeling for Data Visualization*, 2003.

[LFTG97] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. In *SIGGRAPH '97 : Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 117–126, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[LHN05] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Texture sprites : texture elements splatted on surfaces. In *SI3D '05 : Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 163–170, New York, NY, USA, 2005. ACM Press.

[Loo87] C. T. Loop. *Smooth subdivision surfaces based on triangles.* PhD thesis, Department of Mathematics, University of Utah, 1987.

[LP02a] L. Linsen and H. Prautzsch. Fan clouds - an alternative to meshes. In *Proceedings of Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology, and Computational Imaging*, 2002.

[LP02b] Lars Linsen and Hartmut Prautzsch. Fan clouds - an alternative to meshes. In *Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology and Computational Imaging*, 2002.

[LR98] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques*, pages 301–314, 1998.

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

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Math. Comp.*, 37 :141–159, 1981.

[LW85] M. Levoy and T. Whitted. The use of points as display primitives. Technical Report Technical Report TR 85-022, The University of North Carolina at Chapel Hill,Department of Computer Science, 1985.

[MKN+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04 : Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, New York, NY, USA, 2004. ACM Press.

[MMS+04] Carsten Moenning, Facundo Mémoli, Guillermo Sapiro, Nira Dyn., and Neil A. Dodgson. Meshless geometric subdivision. Technical report, IMA Preprint Series number #1977, 2004.

[MO95] Nelson L. Max and Keiichi Ohsaki. Rendering trees from precomputed z-buffer views. In *Rendering Techniques*, pages 74–81, 1995.

[MP89] Gavin S. P. Miller and Andrew Pearce. Globular dynamics : A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3) :305–309, 1989.

[OBA+03] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22(3) :463–470, July 2003.

[Pau03] Mark Pauly. Point primitives for interactive modeling and processing of 3d geometry. Master's thesis, ETH Zürich, 2003.

[PGK02] Mark Pauly, Markus Gross, and Leif Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the 13th IEEE Visualization Conference*, pages 163–170, 2002.

[PKG03] Mark Pauly, Richard Keiser, and Markus Gross. Multi-scale feature extraction on point-sampled models. In *Proceedings of Eurographics 2003*, pages 121–130, 2003.

[PKKG03] Mark Pauly, Richard Keiser, Leif Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. In *Proceedings of ACM SIGRAPH 2003*, pages 641–650, 2003.

[PZvG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels : surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000*, pages 335–342, 2000.

[Ree83] William T. Reeves. Particle systems : a technique for modeling a class of fuzzy objects. In *SIGGRAPH '83 : Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 359–375, New York, NY, USA, 1983. ACM Press.

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat : A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH 2000, Computer Graphics Proceedings*, pages 343–352, 2000.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Streaming qsplat : a viewer for networked visualization of large, dense models. In *SI3D '01 : Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 63–68, New York, NY, USA, 2001. ACM Press.

[RLE+82] Smith A. R., Carpenter L., Catmull E., Cole P., Cook R., Poor T., Porter T., and Reeves W. Genesis demo documentary (film), June 1982.

[RPZ02] L. Ren, H. Pfister, and M. Zwicker. Object space ewa surface splatting : A hardware accelerated approach to high quality point rendering. In *Proceedings of Eurographics 2002*, 2002.

[SA04] Mark Segal and Kurt Akeley. The opengl graphics system : A specification, October 2004. http ://www.opengl.org/documentation/specs/version2.0/glspec20.pdf.

[SAE93] Leon A. Shirmun and Salim S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *Proceedings of Eurographics 93*, 1993.

[Sam89] H. Samet. *The Design and Analysis of Spatial Data Structures*. Reading, Mass. : Addison Wesley, 1989.

[SB03] M.A. Sabin and L. Barthe. *Curve and Surface Fitting : St Malo 2002*, chapter Artifacts in recursive subdivision surfaces, pages 353–362. Nashboro Press, Brentwood, 2003.

[SD01] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics workshop on Rendering*, pages 151–162, 2001.

[SGwHS98] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, New York, NY, USA, 1998. ACM Press.

[Sim90] Karl Sims. Particle animation and rendering using data parallel computation. In *SIGGRAPH '90 : Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 405–413, New York, NY, USA, 1990. ACM Press.

[SJ00] Gernot Schaufler and Henrik Wann Jensen. Ray tracing point sampled geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 319–328, London, UK, 2000. Springer-Verlag.

[ST92] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *SIGGRAPH '92 : Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 185–194, New York, NY, USA, 1992. ACM Press.

[TRS04] I. Tobor, P. Reuter, and C. Schlick. Multiresolution reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Proceedings of Shape Modeling International 2004*, 2004.

[Tur92] G. Turk. Re-tiling polygonal surface. In *Proceedings of ACM SIGGRAPH 92*, 1992.

[VPBM01] Alex Vlachos, Jörg Peters, Chas Boyd, and Jason L. Mitchell. Curved pn triangles. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001.

[vW97] C. W. A. M. van Overveld and B. Wyvill. Phong normal interpolation revisited. *ACM Transaction On Graphics*, 16(4) :397–419, 1997.

[WFP+01] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Strasser. The randomized z-buffer algorithm : interactive rendering of highly complex scenes. In *SIGGRAPH '01 : Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 361–370, New York, NY, USA, 2001. ACM Press.

[WH94] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *SIGGRAPH '94 : Proceedings of the 21st annual conference*

on Computer graphics and interactive techniques, pages 269–277, New York, NY, USA, 1994. ACM Press.

[WK04] J. Wu and Leif Kobbelt. Optimized sub-sampling of point sets for surface splatting. In *Proceedings of Eurographics 2004*, pages 643–652, 2004.

[WPK+04] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella, and M. Gross. Post-processing of scanned 3d surface data. In *Proceedings of Eurographics Symposium on Point-Based Graphics*, pages 85–94, 2004.

[WS05] Ingo Wald and Hans-Peter Seidel. Interactive Ray Tracing of Point Based Models. In *Proceedings of 2005 Symposium on Point Based Graphics*, 2005.

[WW02] J. Warren and H. Weimer. *Subdivision methods for geometric design : A constructive approach*. 2002.

[ZKEH97] Hansong Zhang and III Kenneth E. Hoff. Fast backface culling using normal masks. In *SI3D '97 : Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 103–ff., New York, NY, USA, 1997. ACM Press.

[ZMHH97] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. *Computer Graphics*, 31(Annual Conference Series) :77–88, 1997.

[ZPKG02] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3d : an interactive system for point-based surface editing. In *Proceedings of ACM SIGGRAPH 2002*, pages 322–329, 2002.

[ZPvBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of ACM SIGGRAPH 2001*, pages 371–378, 2001.

[ZPvBG02] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3) :223–238, 2002.

[ZRB+04] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly. Perspective accurate splatting. In *Graphics Interface 2004*, 2004.

[ZS00] Denis Zorin and Peter Schröder. Subdivision for modeling and animation. In *SIGGRAPH 2000 Course Notes*, 2000.

[ZSS96] Denis Zorin, Peter Schröder, and Wim Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of ACM SIGGRAPH 1996*, pages 189–192, 1996.

[Zwi03] Matthias Zwicker. *Continuous Reconstruction, Rendering, and Editing of Point-Sampled Surfaces*. PhD thesis, ETH Zurich, 2003.

[ZZY04] Mingli Zhang, Sanyuan Zhang, and Xiuzi Ye. Progressive transmission of point set surfaces based on geometry image representation. In *VRCAI '04 : Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 406–411, New York, NY, USA, 2004. ACM Press.