



THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *20 Mars 2014* par :

Frédéric Claux

Rendu interactif de modèles B-Rep sur GPU

JURY

MATHIAS PAULIN
BRUNO LÉVY
JEAN-CLAUDE LÉON
RAPHAËLLE CHAINE
MARC DANIEL
LOÏC BARTHE
JEAN-PIERRE JESSEL

Professeur des Universités
Directeur de recherche
Professeur des Universités
Professeur des Universités
Professeur des Universités
Maître de Conférences (HDR)
Professeur des Universités

Président du Jury
Membre du Jury
Membre du Jury
Membre du Jury
Membre du Jury
Membre du Jury
Membre du Jury

École doctorale et spécialité :

EDMITT : Mathématiques, Informatique et Télécommunications

Unité de Recherche :

Institut de Recherche en Informatique de Toulouse (IRIT-CNRS UMR 5505)

Directeur de Thèse :

Jean-Pierre Jessel

Rapporteurs :

Bruno Lévy et Jean-Claude Léon

Rendu interactif de modèles B-Rep sur GPU

THÈSE

présentée et soutenue publiquement le 20 mars 2014

pour l'obtention du

Doctorat de l'Université de Toulouse
(mention informatique)

par

Frédéric Claux

Composition du jury

<i>Rapporteurs :</i>	Bruno Lévy	Directeur de recherche, INRIA Nancy Grand Est
	Jean-Claude Léon	Professeur, Institut Polytechnique de Grenoble
<i>Examineurs :</i>	Raphaëlle Chaine	Professeur, Université de Lyon
	Marc Daniel	Professeur, Université d'Aix-Marseille
	Loïc Barthe	Maître de conférences HDR, Université de Toulouse
	Mathias Paulin	Professeur, Université de Toulouse
	Jean-Pierre Jessel	Professeur, Université de Toulouse, directeur de thèse

Mis en page avec la classe thesul.

Remerciements

Après plus d'une dizaine d'années dans l'ingénierie et l'industrie, faire une thèse n'est pas une mince affaire ; revenir de l'étranger avec femme et enfant et se lancer dans la recherche n'est pas sans risque ; proposer un sujet de recherche à une université et à des responsables industriels l'est encore moins. Je voudrais ici remercier les personnes qui m'ont aidées dans cette aventure qui a pour moi été passionnante, en dépit des veillées et quelques nuits blanches loin de ma famille.

Loïc Barthe m'a aidé là où j'en avais le plus besoin, pour me remettre à niveau en mathématiques. Mes connaissances étaient quelque peu rouillées suite à des années passées à utiliser des connaissances géométriques qui m'apparaissent primaires aujourd'hui. J'ai apprécié sa rigueur scientifique, son souci du détail, et ses encouragements, en particulier au départ où ce sentiment de « prise de risque » vous met parfois la pression, en grande partie, dans mon cas, pour les raisons citées plus haut.

Je voudrais remercier chaleureusement Mathias Paulin, avec qui j'ai eu des échanges constructifs qui ont grandement amélioré la qualité de mon travail. Je remercie Mathias pour m'avoir fait découvrir le monde de la recherche au sens large, de m'avoir fait profiter de son expérience dans de nombreux domaines, encouragé à de nombreuses reprises, et beaucoup aidé dans mes financements.

Merci à Jean-Pierre Jessel pour m'avoir fait confiance dans mon travail et plus particulièrement dans ma capacité à traiter un sujet que je défendais avec foi, et pour m'avoir fait surmonter plusieurs difficultés qui auraient pu être des obstacles fatals dans mon parcours. Jean-Pierre m'a accompagné dans mes démarches depuis la mise en place de ma thèse, complexe, jusqu'à la remise du manuscrit final.

Je remercie David Vanderhaeghe qui a travaillé avec moi fin 2011 et la première moitié de 2012. Je pense notamment au moment de la première soumission où David a été présent à mes côtés, pendant toutes les phases de préparation.

Je remercie Mathias, Loïc et David pour les tâches d'enseignement qu'ils m'ont confiées.

Merci à Géraldine Morin qui m'a aidé à venir à bout de quelques déboires en géométrie analytique et à démêler certains de mes splines. Véronique Gaildrat pour m'avoir fort bien reçu à mon arrivée à l'IRIT. Sylvain Lefebvre pour des suggestions d'amélioration de mon travail.

Je remercie Datakit pour m'avoir prêté une license d'utilisation de leur bibliothèque de lecture de fichier CATIA, indispensable pour mes recherches ; Airbus pour avoir autorisé l'utilisation de leurs énormissimes modèles de données pour mes tests ; le Centre National d'Etudes Spatiales pour le modèle de donnée utilisé dans la première partie de ma recherche. Merci à Andre Schollmeyer pour m'avoir aidé à bien comprendre et exploiter ses travaux, à plusieurs reprises. Philippe Oster pour m'avoir expliqué les principes de base du processus de fabrication à partir de modèles.

Je remercie ma compagne Adele et ma fille Sophie pour avoir supporté mes journées et nuits difficiles passées à programmer, déboguer, étudier ou rédiger. Enfin, mon père, qui a été d'un grand support tout au long de ma thèse.

Je voudrais enfin remercier les rapporteurs et examinateurs pour le temps consacré à la lecture de ce manuscrit.

*Je dédie cette thèse
à Sophie et à Antoine.*

Sommaire

Partie I	Introduction	1
1	La problématique du rendu interactif de modèles CAO	3
1.1	Précision de l’affichage	4
1.2	Un temps d’affichage compatible avec l’interaction	5
1.3	La prise en charge de grands volumes de données	6
1.4	Visualisation pendant la modélisation ou après assemblage final	6
1.5	Le rôle du GPU	6
1.5.1	La vocation du GPU	6
1.5.2	Spécificités de fonctionnement du GPU	7
2	Les modèles B-Rep	9
2.1	Définition	9
2.1.1	Concept général	9
2.1.2	Surfaces support	10
2.1.3	Découpe	10
2.2	Limites de la représentation B-Rep	10
2.2.1	Jours et recouvrements géométriques	10
2.2.2	Définition surfacique vs. volumique	11
3	Rendu graphique sur GPU	13
3.1	Les pipelines de traitement sur GPU	13
3.1.1	La rasterisation de primitives	15
3.1.1.1	Les primitives simples	15
3.1.1.2	Les primitives de haut niveau	15
3.1.2	Le traitement générique parallèle	16
3.2	Le fonctionnement interne du GPU	18
3.2.1	Un GPU difficile à configurer pour un travail précis	19
3.2.2	Convergence et divergence d’exécution	20

3.2.3	Des algorithmes peu adaptés à une exécution sur GPU... ou l'inverse .	21
3.2.4	Des ressources limitées	21
Partie II	Rendu de modèles B-Rep - un état de l'art	23
1	Evaluation et échantillonnage de points et de dérivées	25
1.1	Evaluation	25
1.2	Echantillonnage de points sur les surfaces support	27
1.2.1	Critère qualitatif de la taille des arêtes	28
1.2.2	Critère qualitatif de la déviation entre la surface et son approximation	28
2	Rendu par tessellation statique globale	35
2.1	Principe général	35
2.2	Tessellation des faces	35
2.3	Elimination des cracks	40
2.4	Niveaux de détails	42
3	Rendu basé découpe	45
3.1	Principe général	45
3.2	Gestion de la découpe	46
3.2.1	Représentation discrète	46
3.2.2	Représentation vectorielle	47
3.3	Rendu des surfaces support	56
3.3.1	Tessellation uniforme	56
3.3.2	Lancer de rayon	57
3.3.2.1	Méthode analytique	58
3.3.2.2	Subdivision uniforme	59
3.3.2.3	Bézier clipping	60
3.3.2.4	Méthode de Newton	63
3.3.3	Tessellation adaptative	65
3.3.4	Micro-polygonisation	67
3.4	Elimination des cracks entre les faces	68
4	Analyse générale de l'état de l'art	73
4.1	Des méthodes d'évaluation matures	73
4.2	Limitations de la tessellation statique de modèle	73
4.3	Opportunités offertes par le rendu basé découpe	74
4.4	De l'état de l'art aux contributions	74

Partie III	Rendu basé découpe optimisé pour les performances	75
1	Introduction	77
1.1	Objectif recherché	77
1.2	Analyse de l'état de l'art	78
1.3	Vue d'ensemble de la méthode proposée	79
2	Approximation du modèle d'entrée	83
2.1	Découpe	83
2.1.1	Implicitisation des courbes de découpe	83
2.1.1.1	Echantillonnage et interpolation par des B-Splines	83
2.1.1.2	Approximation sous contrôle d'erreur	84
2.1.1.3	Décomposition pièce à pièce en courbes de Bézier	84
2.1.1.4	Reconstruction implicite locale	85
2.1.2	Structure multirésolution d'accès à l'espace de découpe	85
2.2	Surfaces support	86
2.2.1	Primitives simples	86
2.2.2	Autres primitives	87
3	Rendu	91
3.1	Rendu des surfaces support	91
3.2	Classification	93
3.2.1	Classification de fragments	93
3.2.1.1	Cellules avec une seule courbe quadratique	93
3.2.1.2	Cellules avec deux courbes quadratiques	93
3.2.1.3	Mécanisme d'accès multirésolution	94
3.2.2	Classification de triangles	96
4	Résultats et limitations	101
4.1	Modèles utilisés	101
4.2	Qualité visuelle	101
4.3	Performances	102
4.4	Occupation mémoire	102
4.5	Limitations	102
5	Conclusion	107

Partie IV	Rendu sans crack de modèles tessellés dynamiquement	109
1	Introduction	111
1.1	Limitations des méthodes de l'état de l'art pour le rendu haute qualité	111
1.2	Vue d'ensemble de la méthode proposée	112
2	Algorithme	113
2.1	Etape 1 : rendu initial	113
2.2	Etape 2 : détection des cracks	114
2.3	Remplissage en espace écran	116
2.4	Remplissage en espace objet	117
2.4.1	Etape 3 : construction du masque de profondeur	117
2.4.2	Etape 4 : remplissage en espace objet	118
2.4.2.1	Lancer de rayon sur les surfaces à l'emplacement des cracks .	119
2.4.2.2	Mécanisme de mailboxing	121
2.4.3	Etape 5 : sortie écran	121
3	Résultats	123
3.1	Qualité de rendu	123
3.1.1	Comparaison des deux méthodes de remplissage de cracks	123
3.1.2	Comparaison avec un rendu effectué intégralement en lancer de rayon	124
3.2	Performance	125
4	Conclusion	129
Partie V	Conclusion et perspectives	131
1	Bilan de notre recherche	133
2	Discussion et perspectives	135
2.1	Accélération de la découpe	135
2.2	Suppression des jours géométriques	136
2.2.1	Au moment du rendu	136
2.2.2	Via transformation du modèle	137
	Glossaire	139
	Index	141

Première partie

Introduction

Chapitre 1

La problématique du rendu interactif de modèles CAO

Les modèles de données pour la Conception Assistée par Ordinateur (ci-après dénommée CAO) sur lesquels cette thèse s'appuie ont une spécification qui répond à un certain nombre de critères qui sont brièvement expliqués dans ce chapitre. Ce chapitre rappelle comment les modèles CAO sont créés et exportés, et dans quel contexte et pour quels objectifs leur visualisation doit être assurée pendant leur cycle de vie, avant d'être finalement usinés et mis sur le marché.

Les objets manufacturés tels que les voitures, les avions mais également les produits de la vie de tous les jours que l'on trouve dans les habitations tels que les lampes ou les écrans d'ordinateur, quel que soit le matériau avec lequel ils sont fabriqués, sont d'abord conçus avec un logiciel de CAO.

La phase de conception, interactive par essence, se fait avec des primitives géométriques simples et aboutit à la création de pièces généralement individuellement usinées. La visualisation de ces modèles, à l'intérieur du logiciel de conception, doit être fidèle à leur représentation analytique définie par son ou ses concepteurs. Le rendu des objets à l'écran propose un assemblage visuel de formes tel qu'il a été décrit dans l'arbre de construction du modèle, ce dernier définissant l'ensemble des opérations de modélisation pour arriver à la forme voulue (Figure 1.1).

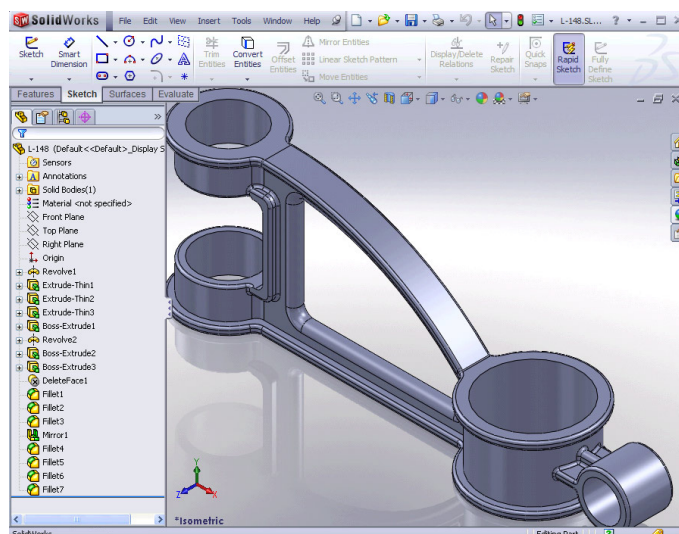


FIGURE 1.1 – Modélisation d'une pièce mécanique au sein du logiciel SolidWorks.

Passée la phase de conception, et avant d’aller dans la phase de fabrication, les modèles CAO conçus sont parfois intégrés dans des logiciels de test ou de simulation pour divers usages. Leur structure est alors verrouillée. La visualisation du modèle CAO peut alors éventuellement être complétée par l’affichage de meta-données sur la surface du modèle, comme illustré sur la Figure 1.2. Par ailleurs, les logiciels de tests permettent d’annoter les modèles en offrant la possibilité de placer des capteurs, éléments disséminés sur toute la maquette virtuelle servant à étudier par exemple la fatigue mécanique des pièces.

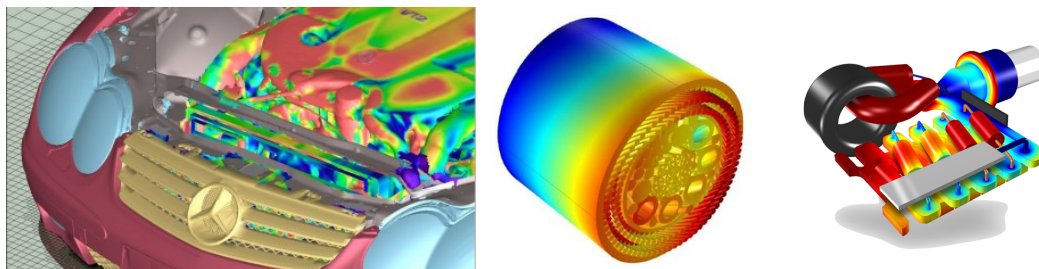


FIGURE 1.2 – Visualisation de modèles B-Rep et représentation de données issues de la simulation. Exemples de représentation visuelle pour l’analyse thermique, mécanique et électromagnétique.

La phase de test ou de simulation fonctionne avec des assemblages entiers de pièces, comme par exemple des sections entières d’avion. Les pièces qui étaient individuellement conçues dans le logiciel de conception sont exportées à intervalles réguliers, puis assemblées et enfin transformées dans un format compréhensible par le logiciel de simulation ou de test utilisé. La fréquence de génération de ces modèles exportés est généralement faible par rapport à la fréquence de modification des modèles pendant la phase de conception. Cela autorise le processus de conversion à préparer le modèle, souvent volumineux, dans un format optimal pour une utilisation plus efficace par la suite. Cette préparation, qui peut être lourde, ne peut être réalisée au sein d’un logiciel de modélisation au rythme des opérations de modélisation, puisque chacune de ces opérations doit être immédiatement visualisable dans la fenêtre de vue.

Les besoins en visualisation interactive se regroupent en **deux grands domaines** distincts. D’une part la **visualisation instantanée de petites pièces** au sein même du logiciel de conception, au rythme de la navigation et des opérations de modélisation. De l’autre, la **visualisation de grands assemblages** de modèles, sans possibilité d’édition. Le premier besoin implique une grande réactivité du moteur de rendu vis-à-vis de son modèle de données sous-jacent, en constante modification, et doit fournir une très grande qualité visuelle. Le second requiert la prise en charge de modèles parfois très grands (sections d’avion entières, modèle d’usine etc.), autorise un prétraitement avant visualisation ainsi qu’une plus grande tolérance relativement à la précision de l’affichage.

Le travail réalisé pendant la thèse adresse ces deux besoins dans les Parties III et IV. Le prochain chapitre pose plus en avant les problématiques de rendu interactif dans leur ensemble.

1.1 Précision de l’affichage

Les modèles CAO sont définis dans un format décrivant d’une manière analytique les objets (Chapitre 2), autorisant un affichage d’une résolution théoriquement illimitée. La précision requise pour l’affichage des modèles dépend de l’applicatif.

Pendant la conception, la précision doit être maximale car l'affichage doit très fidèlement rendre compte de la définition exacte du modèle. Cet objectif ne reste pourtant qu'un idéal dans les logiciels de conception qui existent actuellement sur le marché. Bien que reposant sur des définitions analytiques décrivant des courbes et des surfaces lisses, des défauts de discrétisation engendrant des approximations de mise en couleurs et de reflets apparaissent, et des problèmes de jonctions entre faces peuvent se manifester à l'affichage, quel que soit le logiciel de CAO utilisé. Ces défauts sont identifiables sur la Figure 1.3.

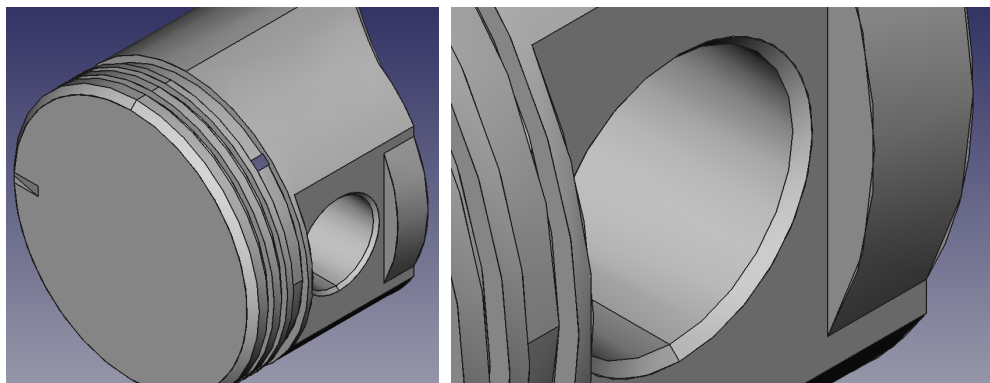


FIGURE 1.3 – Visualisation d'une pièce mécanique pendant le processus de conception dans un logiciel de CAO. L'image de droite est un zoom sur l'image de gauche. À droite, les défauts visuels engendrés par la discrétisation apparaissent assez nettement : les formes perdent leur caractère lisse et des cracks apparaissent entre les faces.

Pendant les phases de test, les opérateurs peuvent être amenés à poser sur de très grandes maquettes virtuelles des capteurs de taille très réduite (Figure 1.4), nécessitant une visualisation fidèle et précise. Ces capteurs sont parfois posés sur des chanfreins ou sur des parties fines des pièces mécaniques, et dont la résistance mécanique doit être mise à l'épreuve. Sans être aussi primordial que pendant la phase de conception, proposer un rendu haute fidélité reste important afin que les techniciens puissent placer avec le plus de précision possible les capteurs qui serviront à étudier entre autres choses la fatigue mécanique, à des endroits très localisés. Proposer un rendu virtuel fidèle est également souhaitable dans la mesure où les techniciens doivent localiser l'endroit physique où le capteur doit être placé à partir de l'emplacement correspondant sur la maquette virtuelle. Cette apparence doit rester la plus proche possible de l'assemblage « réel » physique pour éviter toute confusion.

1.2 Un temps d'affichage compatible avec l'interaction

Les modèles doivent être affichés suffisamment rapidement pour permettre l'interaction. Idéalement, toute la précision à l'affichage doit être conservée pendant l'interaction. Si cela n'est pas réalisable, il doit être possible de dégrader la qualité d'affichage pendant la navigation et de restaurer toute la précision pour les images fixes. Ce changement de qualité doit idéalement être simple à mettre en œuvre, avec un moteur de rendu suffisamment homogène et flexible pour proposer cette fonctionnalité.



FIGURE 1.4 – Exemples de capteurs piézoélectriques utilisés pour étudier des phénomènes physiques tels que les vibrations.

1.3 La prise en charge de grands volumes de données

Pendant leur conception, les pièces mécaniques sont éditées et visualisées d'une manière individuelle et le volume de données est par conséquent réduit. Lorsque les modèles sont visualisés d'une manière virtuelle lors des simulations ou des tests, ils le sont après assemblage, dans leur ensemble. C'est par exemple le cas du modèle d'avion qui nous a été fourni, où des sections de plusieurs centaines de milliers de faces B-Rep doivent être affichées d'un bloc, et dont la navigation doit rester interactive. Outre la problématique de performance évoquée plus haut, celle de l'espace mémoire requis pour contenir le modèle est posée, puisque cet espace est le plus souvent réduit sur les cartes graphiques. Il convient de trouver une représentation adaptée des données permettant de représenter fidèlement les modèles, sans sacrifier la précision lorsque c'est possible, lorsque la représentation B-Rep brute, telle qu'elle est obtenue fraîchement après export, n'est pas suffisante pour effectuer le rendu — ce qui est hélas le cas pour toutes les méthodes d'affichage connues à ce jour (Partie II).

1.4 Visualisation pendant la modélisation ou après assemblage final

Les modèles CAO doivent être traités préalablement à leur prise en charge par le moteur de rendu, quelle que soit la méthode de rendu utilisée. Le temps de conversion doit être réduit le plus possible pour permettre des exports fréquents. Un temps de conversion très bas ouvre la possibilité de visualiser en temps réel le modèle, pendant la conception, en faisant un export après chaque opération de modélisation. Un temps de conversion plus élevé ferme la porte à un affichage pendant la conception, mais ouvre la porte à des transformations plus poussées destinées à augmenter les performances au moment du rendu.

1.5 Le rôle du GPU

1.5.1 La vocation du GPU

Les cartes graphiques étaient autrefois simplement composés d'un RAMDAC (abréviation en langue anglaise de *Random Access Memory Digital to Analog Converter*), un chipset destiné à lire une partie de la mémoire centrale d'un micro-ordinateur interprétée comme étant une série de

pixels, et à envoyer à un moniteur un signal analogique correspondant pour produire une image. Par la suite, des primitives géométriques telles que des lignes ou des triangles ont pu directement leur être passées d'une manière vectorielle pour qu'elles transforment elles-mêmes ces informations en pixels préalablement à l'envoi vers un moniteur. Les processeurs graphiques — Graphics Processing Units, ou GPU — avaient vu le jour. La définition des données géométriques ne peut se limiter à de simples lignes ou triangles, les modèles B-Rep en sont un parfait exemple. La programmabilité des GPU (Chapitre 3), très avancée aujourd'hui, permet de traiter des données arbitraires, vectorielles ou non, et de les afficher d'une manière autonome, par l'intermédiaire de programmes directement embarqués sur les cartes graphiques. Le processeur central n'a même plus à être sollicité et est libre d'effectuer d'autres tâches, en particulier celles non liées à l'affichage. Le processeur graphique assume donc non seulement l'affichage en lui-même des données, mais leur **interprétation** : on peut maintenant lui donner des données à afficher sous formes de données métier, plus ou moins brutes, et il se charge de manière autonome d'en effectuer le rendu à la demande. C'est dans cet état d'esprit et en respectant cette vocation que cette thèse a été réalisée. La problématique du passage de données B-Rep aussi brutes que possible au GPU est abordée.

1.5.2 Spécificités de fonctionnement du GPU

Les GPU intègrent des unités dédiées au rendu graphique. Ils disposent également d'une architecture reposant sur le traitement parallèle et possèdent des ressources mémoire spécifiques dont l'utilisation doit être judicieusement orchestrée. L'exploitation du GPU exige de penser les algorithmes de sorte que les ressources soient toujours utilisées au mieux, que le traitement parallèle puisse effectivement se faire, et que les unités graphiques dédiées soient au mieux exploitées. Cette thèse présente ces problématiques dans le Chapitre 3. Elle précise ultérieurement comment les algorithmes des Parties III et IV sont implémentés pour répondre aux exigences de fonctionnement du GPU.

Chapitre 2

Les modèles B-Rep

2.1 Définition

2.1.1 Concept général

B-Rep est une abbréviation de l'anglais *Boundary Representation*, ou Représentation par Frontières en français. Le format B-Rep est utilisé pour représenter précisément l'apparence extérieure d'objets volumiques des modèles CAO en vue d'être visualisés ou fabriqués. Ils s'est développé dans les années 1980 au sein des logiciels de modélisation et au détriment de la représentation CSG (Constructive Solid Geometry) qui permet plus difficilement de représenter les opérations de modélisation très riches des logiciels de CAO.

Les modèles sont représentés par une collection de *faces*, définies par des *surfaces support* et des *courbes de découpe* exprimées dans l'espace paramétrique de ces surfaces support (Figure 2.1). La représentation des surface support et des courbes de découpe est analytique, et non discrète comme c'est le cas pour les maillages, ce qui permet de définir des formes non linéaires, courantes dans le monde de la CAO, et autorise un affichage ou une fabrication ultérieurs à une résolution arbitraire, théoriquement illimitée.

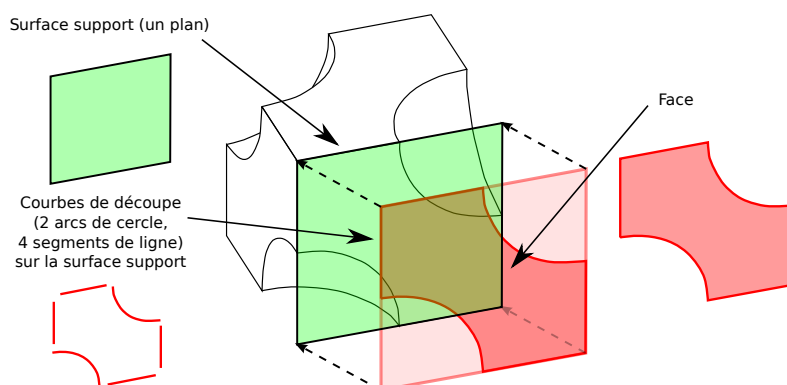


FIGURE 2.1 – Anatomie d'une face B-Rep.

Se focalisant exclusivement sur l'apparence surfacique des objets, le format B-Rep n'expose aucune information de conception et ne dispose d'aucun arbre de construction. Son utilité est double. Tout d'abord, il est utilisé comme format d'export dans les logiciels de conception. Dénués de toute information de conception qui n'a de toute façon pas à être divulguée par les ingénieurs mécaniciens, les modèles CAO exportés au format B-Rep sont ensuite lus par un logiciel de

fabrication utilisé pour l'usinage des pièces. On parle de Fabrication Assistée par Ordinateur, ou FAO (Computer Aided Manufacturing, ou CAM, en anglais). Le format B-Rep est également utilisé d'une manière interne par les logiciels de conception pour afficher les objets, pendant les sessions de modélisation. Les logiciels exportent ainsi en temps-réel la représentation B-Rep à partir de leur format propriétaire de données et de l'arbre de construction du modèle en cours d'édition. Le modèle objet B-Rep est disponible sous forme d'API de programmation dans ces logiciels et le moteur de rendu intégré l'utilise pour les besoins de l'affichage. Le format B-Rep est donc incontournable pour le rendu, et les algorithmes présentés dans la Partie II reposent tous dessus.

2.1.2 Surfaces support

Les surfaces sont définies d'une manière bi-paramétrique et peuvent être de différents types. Il peut s'agir de primitives simples, comme des surfaces planes, coniques, sphériques ou toroïdales. D'autres surfaces sont définies à partir d'une esquisse filaire qui a été extrudée, ou ayant subi une révolution. D'autres types de surfaces sont définies avec des NURBS (Non Uniform Rational B-Splines). Les NURBS peuvent facilement être décomposées en ensembles de surfaces de Bézier rationnelles [CLR80, Far90](Partie II). Les primitives simples, les esquisses extrudées ou révolues et la quasi-totalité des primitives présentes dans les modèles B-Rep peuvent être converties en NURBS sans perte d'information. En pratique, avec les modèles que nous avons utilisés comportant parfois plusieurs millions de faces B-Rep, aucune surface n'a posé de problème pour une conversion sans perte dans une représentation en NURBS.

2.1.3 Découpe

Les courbes de découpe résident à la jonction de deux faces adjacentes. Chaque courbe de découpe est définie deux fois, une fois dans chaque espace paramétrique de la surface support adjacente correspondante. L'ensemble des courbes de découpe d'une surface support délimite la partie de la surface support dite *sur la face* de celle située *hors de la face*. On parle également de partie *dans la zone de découpe* ou *hors de la zone de découpe*. Seule la partie *dans la zone de découpe* est affichée et prise en considération pour l'usinage de la pièce (ou pour un affichage à l'écran dans notre cas). La Figure 2.2 montre en couleur rouge la jonction entre deux faces et les deux courbes de découpe qui en résultent. Comme pour les surfaces, les courbes de découpe sont définies avec des représentations simples telles que des arcs de cercle ou des lignes, ou sous forme de NURBS. Comme pour les surfaces, la totalité des courbes de découpe peut être transformée sous forme de NURBS, qui peuvent être converties à leur tour en ensemble de courbes de Bézier rationnelles.

Le lecteur est invité à consulter l'ouvrage intitulé *Computer Integrated Manufacturing and Engineering* [RNS93] pour plus d'informations sur la représentation des données au format B-Rep.

2.2 Limites de la représentation B-Rep

2.2.1 Jours et recouvrements géométriques

La représentation B-Rep décrit l'apparence extérieure face par face, et indépendamment les unes des autres. Le fait que les courbes de découpe soient définies pour chaque face peut créer dans certains cas des écarts ou *jours* non désirés entre les faces adjacentes. Plus rarement,

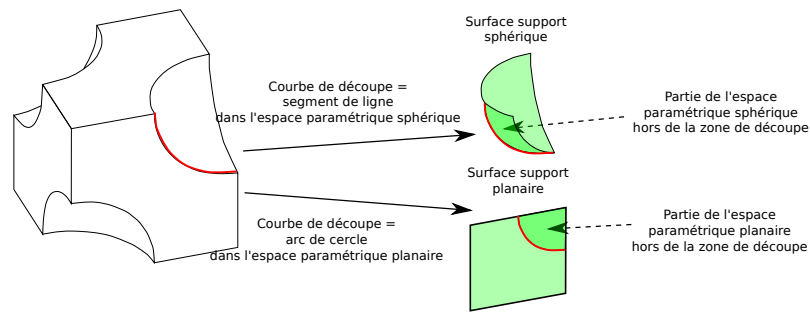


FIGURE 2.2 – Les courbes de découpe sont exprimées dans l'espace paramétrique de leur surface support respective.

des recouvrements peuvent se produire. Ces imperfections sont illustrées sur la Figure 2.3. Les jours et recouvrements résultent de la non-concordance dans l'espace 3D de l'empreinte laissée par les deux courbes de découpe paramétriques, pour chaque face adjacente. Leur existence s'explique par la difficulté à représenter analytiquement l'intersection de deux surfaces d'une manière parfaitement précise [SAG84]. Leur taille est souvent très réduite, de l'ordre du dixième ou du centième de millimètre.

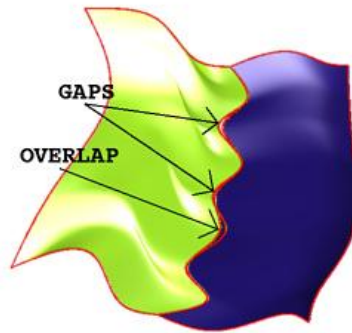


FIGURE 2.3 – Intersection de deux surfaces complexes de type NURBS, représentée sous forme de deux courbes de découpe, l'une définie dans l'espace paramétrique de la surface jaune, l'autre dans celui de la surface bleue. Deux jours (gaps en anglais) et un recouvrement (overlap) sont visibles.

Pendant le rendu, aussi bien pendant une session de modélisation que pendant la visualisation d'un grand modèle lors d'une session de test, ces jours et recouvrements ne sont visibles qu'à très fort niveau de zoom. Les opérateurs CAO sont habitués à les voir, d'autant plus que les moteurs de rendu internes aux logiciels de modélisation souffrent de défauts de discrétisation, qui sont plus gênants (Figure 1.3 ; ces défauts seront illustrés et discutés dans la Partie II). Pour les phases de simulation, les jours et recouvrement entre les faces compromettent la réalisation de nombreuses opérations. Il en va de même pour la fabrication.

2.2.2 Définition surfacique vs. volumique

Pour contourner la limitation de la représentation B-Rep relative à son incapacité à définir un modèle parfaitement fermé, en raison des jours, c'est-à-dire un volume parfaitement étanche représentant un solide, il est possible d'exporter le modèle avec des informations topologiques

définissant explicitement la continuité entre les faces, en plus de la géométrie. La représentation B-Rep peut à ce titre être *surfactive* ou *volumique*. La forme surfactive ne comporte qu'une collection de faces qui sont indépendantes les unes des autres comme il a été mentionné précédemment. La forme volumique, ou solide, définit, outre les faces, des informations d'adjacence entre faces. On parle alors de format *manifold B-Rep*. Dans l'industrie, le format surfactive se trouve généralement sous la forme de fichier IGES basique, contenant uniquement des entités géométriques. Le format volumique (solide) est supporté dans une version plus récente d'IGES (entités de type 186), ainsi que par le format STEP (AP203 et AP214).

Les logiciels d'usinage tirent parti des informations d'adjacence entre les faces, et les appareils de fabrication peuvent, pour les faces tangentes, usiner le matériau d'une manière continue, en passant d'une face à l'autre, produisant un résultat optimal. Le jour ou recouvrement entre les faces n'étant pas un problème puisque l'information d'adjacence et la continuité géométrique est explicite. En l'absence de ces informations, les appareils de fabrication ne peuvent assurer la continuité de l'usinage et il en résulte des problèmes de finition parfois importants.

Pour le rendu, les jours restent cependant visibles. Il peut être opportun ou non d'afficher les modèles avec ou sans ces jours topologiques, selon les besoins. Le produit usiné n'ayant pas de jours, ceux-ci peuvent être qualifiés d'indésirables. Cependant, l'affichage avec les jours représente en tout état de cause un état brut du modèle tel qu'il est strictement défini sur le plan géométrique.

Il est possible d'éliminer les jours pendant une étape de discrétisation en vue d'une intégration dans un moteur de rendu, comme nous le verrons dans la Partie II. Les logiciels de simulation nécessitant des modèles parfaitement étanches proposent des approches similaires. Des travaux ont été réalisés pour permettre d'éliminer les jours et recouvrements dans un prétraitement, en conservant une représentation analytique, non discrétisée, mais ces méthodes sont relativement lourdes [SSZ*04][SFL*08].

Chapitre 3

Rendu graphique sur GPU

Cette partie détaille les moyens mis à disposition des programmeurs en imagerie 3D par les configurations matérielles disposant d'un GPU. Nous aborderons la programmation graphique d'un point de vue d'une spécification **logicielle** dans un premier temps, puis en nous référant à l'implémentation **matérielle** actuelle dans un second temps. La dualité de cette présentation nous permettra de maîtriser au mieux les moyens mis à notre disposition, tant logiciels que matériels, avant de rentrer dans l'état de l'art relatif aux méthodes de rendu dans la Partie II.

Les travaux présentés dans les Parties III et IV s'intègrent bien dans les couches logicielles existantes dédiées au rendu en trois dimensions. Nous aborderons donc cette partie en nous focalisant d'abord sur le pipeline de rendu standard offert par ces couches logicielles. L'importance des couches logicielles graphiques existantes ne doit pas être sous-estimée. Les fabricants de cartes graphiques sont en effet influencés par les fonctionnalités spécifiées par ces dernières. Elles les guident dans la façon de fabriquer leur matériel pour exécuter des tâches précises, d'une manière dédiée et avec une efficacité optimale. Il est donc opportun de comprendre le modèle de programmation graphique d'un point de vue logiciel, celui-ci étant calqué sur des besoins métier qui sont dans le cadre de cette thèse le rendu 3D.

Dans un deuxième temps, le fonctionnement interne des GPU est exposé. Nous expliquerons en quoi l'architecture interne actuelle des GPU a été adoptée par la plupart des fabricants pour implémenter efficacement les couches logicielles graphiques mentionnées dans le premier temps. Nous examinerons le modèle de programmation massivement parallèle offert par les GPU, et décrirons les ressources mises à notre disposition.

L'analyse combinée des couches logicielles et de leur implémentation matérielle actuelle nous permettra de mieux comprendre en quoi les algorithmes décrits dans les Parties III et IV ont été imaginés pour répondre aux exigences de performances décrites dans la Partie I, Chapitre 1.

3.1 Les pipelines de traitement sur GPU

Le GPU se comporte comme une unité passive sous contrôle du processeur central. Les traitements proposés sont au nombre de trois.

La programmation graphique repose depuis longtemps sur une première couche logicielle permettant d'effectuer les opérations de base du rendu graphique : l'affichage des primitives géométriques simples telles que les points, les lignes et les triangles. Cette couche logicielle est mise à disposition du développeur sous forme d'un *pipeline* de **rastérisation**. Ce pipeline est une suite de traitements appliqués à une ou plusieurs primitives géométriques dans le but d'en obtenir le rendu sous forme d'image 2D, constituée de pixels (le framebuffer). Certains traitements ont un

comportement figé, non paramétrable par le développeur, tandis que d’autres traitements doivent être explicitement définis par ce dernier par l’intermédiaire de programmes embarqués appelés *shaders*. Le pipeline de rendu des bibliothèques graphiques OpenGL et DirectX est présenté sous une forme simplifiée dans la Figure 3.1.

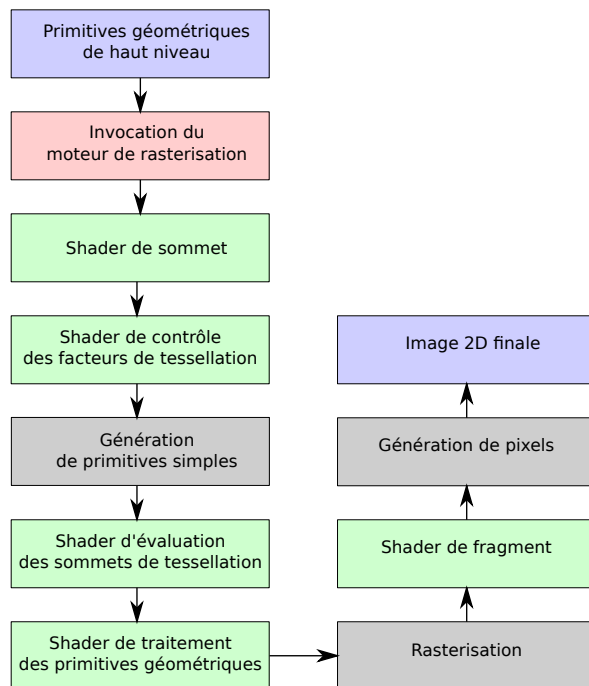


FIGURE 3.1 – Pipeline de rendu proposé par les couches logicielles graphiques modernes. Bleu : données applicatives d’entrée et de sortie. Rouge : appel de rendu (le terme *draw call* est utilisé dans la littérature anglophone). Vert : briques logicielles programmables du pipeline. Gris : briques logicielles non programmables.

Les couches logicielles graphiques mettent également à disposition des programmeurs un autre pipeline pour effectuer des traitements divers qui n’ont pas vocation à rentrer dans le moule du pipeline mentionné précédemment. Ce pipeline de **traitement générique** est très semblable à celui proposé par d’autres interfaces logicielles telles que CUDA [NVI07] ou OpenCL [SGS10], et est illustré dans la Figure 3.2. Il peut être utilisé pour effectuer du calcul parallèle, ou plus généralement lorsqu’il est opportun de configurer le GPU dans un mode de fonctionnement dédié, suffisamment éloigné de celui utilisé pour la mise en œuvre du pipeline de rendu graphique, et où les unités de rasterisation et de tessellation (Section 3.1.1), n’ont pas à être sollicitées.

Le GPU offre enfin un troisième pipeline dédié au **transfert de données** entre la mémoire centrale de la machine et la mémoire locale du GPU. La prise en charge de ce pipeline est assurée par l’unité DMA (*Direct Memory Access*, accès direct à la mémoire). Nous ne nous attarderons pas sur ce pipeline, dont la problématique d’utilisation n’est réellement posée que pour les méthodes de rendu s’intéressant à alimenter en données le processeur graphique en temps-réel et au fur et à mesure de la navigation dans les modèles 3D — une méthodologie souvent dénommée rendu *out of core* [Vit01], non abordée dans cette thèse. Nous écarterons dans le reste de cette partie le transfert de données et ne nous référerons qu’à l’existence des deux traitements et pipelines associés cités précédemment.

Intéressons-nous maintenant aux traitements dans leurs détails.

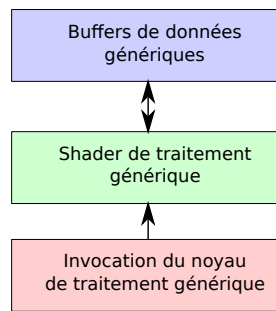


FIGURE 3.2 – Pipeline de traitement générique proposé par les couches logicielles graphiques. Ce pipeline permet de réaliser des traitements spécifiques qui n’ont pas vocation à être exécutés par le pipeline graphique conventionnel.

3.1.1 La rasterisation de primitives

La rasterisation de primitives convertit une image vectorielle définie avec des entités géométriques simples ou de plus haut niveau, en image matricielle de pixels.

3.1.1.1 Les primitives simples

Les primitives simples comportent les points, les lignes et les triangles. Les primitives simples peuvent également être composites, et représenter un ensemble prédéfini de primitives simples unitaires. Ces primitives composites comportent les bandes de lignes ouvertes et fermées ainsi que les bandes et éventails de triangles. Ces entités sont automatiquement décomposées en primitives simples avant d’être passées au shader de traitement des primitives géométriques.

3.1.1.2 Les primitives de haut niveau

Les applicatifs devant faire le rendu d’entités géométriques définies d’une manière vectorielle telles que des cylindres, des cônes ou des entités plus complexes comme des NURBS (Non Uniform Rational B-Splines, primitives que l’on trouve dans le monde de la CAO) se heurtent à l’incapacité évidente des GPU à pouvoir traiter nativement ces données, de manière directe et brute. Ces entités, qui ne sont pas directement prises en charge par le GPU, ont historiquement dû solliciter le processeur central pour une conversion préalable en ensembles de triangles. Nous avons parlé dans la Partie I, Section 1.5.1 de la vocation du processeur graphique, qui est de prendre en charge l’affichage de données métier d’une manière toujours plus autonome. Cette incapacité historique à traiter des formes complexes peut être perçue comme allant à l’encontre de cette vocation.

Depuis 2006, les couches logicielles graphiques proposent des interfaces de programmation permettant de répondre dans une certaine mesure à ces besoins. Les *shaders de traitement des primitives géométriques* (Figure 3.1) permettent de supprimer, modifier ou créer des primitives géométriques à la volée. Versatiles, leur utilisation est toutefois fortement limitée car leur exécution est séquentielle. Plus intéressants, les traitements génériques parallèles (Section 3.1.2) permettent aux développeurs de faire convertir des entités géométriques de haut niveau en primitives simples par le GPU. Performante, cette conversion a toutefois deux désavantages. Le premier désavantage vient du fait qu’elle doit être faite dans une étape préalable à la rasterisation, et ne s’intègre pas à cette dernière. Le second désavantage est qu’elle requiert une configuration du GPU étroitement dépendante des données pour être efficacement exécutée. Sa

mise en œuvre peut être assez lourde (Section 3.1.2).

Depuis 2009, pour permettre la prise en charge d'entités complexes directement au sein d'un traitement de rasterisation de primitives, et ce, sans avoir recours à un traitement générique parallèle préalable, de nouvelles primitives de haut niveau, dénommées *patches*, sont supportées. Ces primitives représentent un maillage de primitives triangulaires ou quadrangulaires ou un assemblage de lignes parallèles. La finesse de génération de ces maillages ou assemblages est contrôlée par un sous-programme spécialisé appelé *shader de contrôle de la tessellation* ; l'emplacement géométrique des sommets discrétisés relève quant à lui du *shader d'évaluation des sommets de tessellation*. L'utilisation de ces shaders est illustré sur la Figure 3.3. Ces shaders apparaissent sous forme de deux boîtes fonctionnelles sur la Figure 3.1. Le mode opératoire basé sur un nouveau type de primitive permet de prendre en charge le rendu d'objets géométriques complexes définis par la couche applicative au moment précis où c'est nécessaire, c'est-à-dire au moment de l'appel aux fonctions de tracé de l'API graphique. Enfin, l'implémentation de l'API graphique assure une exploitation optimale du GPU pour exécuter les shaders associés, sans avoir à manuellement configurer le GPU comme c'est le cas pour les traitements génériques (Section 3.1.2).

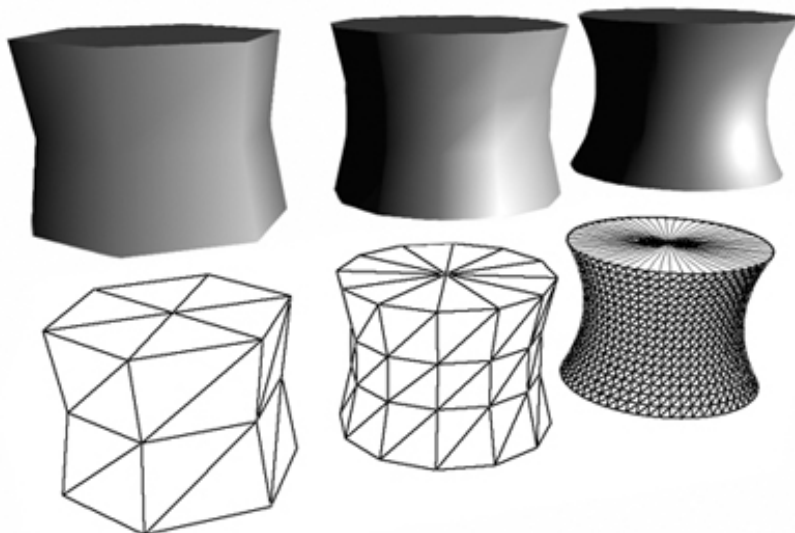


FIGURE 3.3 – Tessellation dynamique d'un cylindre incurvé à partir de sa représentation analytique. La tessellation est ici montrée à trois niveaux de résolution différents.

Le GPU effectue ensuite la rasterisation de chaque primitive simple. Cette rasterisation obéit à une spécification documentée dans le manuel spécifique à la couche logicielle utilisée, que nous ne développerons pas ici. La rasterisation de triangles est illustrée sur la Figure 3.4.

3.1.2 Le traitement générique parallèle

Le traitement générique est un type de traitement proposé par les GPU modernes destiné à effectuer des opérations similaires sur un grand nombre de données, et pouvant typiquement être exécutées en parallèle. Ce type de traitement est calqué sur l'architecture même des GPU, qu'il est nécessaire d'assimiler pour d'une part comprendre, et d'autre part tirer au mieux parti de la couche de traitement générique parallèle, qui est destinée à faire exécuter par le GPU des tâches liées à des données graphiques ou non. On parle du concept de GPGPU (General Purpose

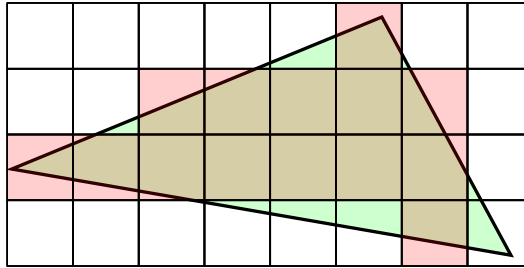


FIGURE 3.4 – Rastérisation d’un triangle plein, affiché en vert. Le GPU produit un fragment à chaque fois que le centre de chaque pixel intersecté par le triangle se trouve effectivement dans l’enceinte de ce dernier. Ainsi, seuls les fragments rouges seront affichés.

GPU).

Les processeurs centraux (CPU) proposent depuis longtemps d’exécuter des programmes différents en parallèle. On parle de processeurs multi-cœurs. En 2014, le nombre de cœurs peut typiquement aller jusqu’à 32. L’exploitation de ces cœurs en simultané ne requiert pas d’attention vraiment particulière de la part du programmeur, ces cœurs étant le plus souvent complètement indépendants les uns des autres. Les GPU peuvent quant à eux exécuter efficacement un très grand nombre d’opérations en parallèle (jusqu’à 2800 à l’heure où nous écrivons ces lignes), à la condition explicite que ces opérations soient *identiques*. On parle d’architecture de type SIMT (*Single Instruction Multiple Threads*) et les cœurs s’appellent des *processeurs de flux*. L’existence de milliers de ces cœurs est rendue précisément possible, et à moindre coût, par le fait qu’ils ne peuvent exécuter qu’une même instruction au même moment. Une utilisation coordonnée sur des jeux de données différents permet d’en tirer pleinement parti.

Leur existence est justifiée par le fait que beaucoup d’algorithmes requièrent l’exécution indépendante d’un programme identique sur de petits jeux de données différents, et où l’exécution d’une même instruction va être réalisée pour toutes les instances du même programme en parallèle et au même moment. Un grand nombre de ces algorithmes touche au traitement d’image. Parmi les exemples les plus simples, et pour illustrer cette mécanique de traitement parallèle, on peut citer la conversion d’une image couleur en noir et blanc, qui nécessite d’examiner la couleur de chaque pixel et d’en extraire la luminance, puis de traduire cette luminance en nuance de gris. Le pseudo-code de ce programme est affiché dans l’Algorithme 1. Cet algorithme peut efficacement être exécuté sur le GPU, où chaque pixel peut être indépendamment traité par un processeur de flux différent.

```

1 foreach pixel p de l’image do
2    $c \leftarrow \text{GetRGB}(p)$ ;
3    $L \leftarrow 0.2126 * c.R + 0.7152 * c.G + 0.0722 * c.B$ ;
4    $\text{SetRGB}(p, L)$ ;
5 end

```

Algorithme 1 : Conversion de la couleur d’un pixel en nuance de gris.

D’autres algorithmes nécessitant de réaliser des traitements simples sur un très grand volume de données incluent les algorithmes numériques, comme par exemple la décomposition d’un nombre entier en produit de facteurs premiers. Une implémentation naïve de cet algorithme est

montrée dans l’Algorithme 2.

```

1  $i \leftarrow n - 1$ ;
2 while  $i > 0$  do
3    $r \leftarrow n/i$ ;
4   if  $r == \text{floor}(r)$  then
5     return  $r, n/r$ ;
6   end
7    $i \leftarrow i - 1$ ;
8 end

```

Algorithme 2 : Décomposition naïve d’un nombre entier en facteurs premiers.

La division $r = n/i$ d’un nombre de plusieurs centaines de chiffres par un autre est une opération coûteuse et il est donc opportun de réduire ce coût. L’Algorithme 2 apparaît comme séquentiel au premier abord, puisque tous les entiers allant de $n - 1$ à 1 sont successivement traités. Il peut être reformulé comme une suite de traitements exécutables en parallèle. De la même manière que les pixels de l’Algorithme 1 étaient indépendamment traités, il est possible d’affecter une division à chaque processeur de flux disponible sur le GPU.

Si le GPU dispose de 500 processeurs de flux par exemple, nous allons lui demander de diviser n par les 500 entiers immédiatement inférieurs à n . Le résultat de ces divisions sous forme de booléen indiquant si le nombre obtenu est réel ou entier sera mis à disposition du processeur central dans un tableau-résultat de 500 booléens. Le processeur central examinera ensuite ce tableau pour déterminer si le résultat de la division est un nombre entier pour l’un des 500 essais, un processus séquentiel simple et rapide. Si aucun résultat entier n’a été obtenu, la division avec les 500 nombres entiers inférieurs suivant sera effectuée par le GPU. Ce processus se répétera jusqu’à ce qu’un résultat entier soit identifié comme étant disponible dans le tableau de sortie. Le parallélisme et la puissance du GPU seront ainsi pleinement mises à contribution.

3.2 Le fonctionnement interne du GPU

Nous avons introduit dans le chapitre précédent les fonctionnalités proposées par les **interfaces logicielles** modernes spécifiées par les principaux consortia de fabricants. Nous avons vu que les API logicielles proposent deux types de traitement. D’une part, la rasterisation de primitives de haut niveau, et d’autre part, le traitement générique et parallèle de données. Ce dernier type de traitement nous a permis de comprendre en partie comment le GPU effectue des traitements parallèles.

Dans cette section, nous allons nous arrêter sur la manière dont sont **implémentés** ces traitements sur les GPU actuels afin de mieux comprendre comment optimiser le **rendement** de ces derniers. Les travaux des Parties III et IV détaillés dans cette thèse reposant sur la rasterisation de primitives, nous nous intéressons plus particulièrement à ce type de traitement. Pour bien le cerner, il est toutefois nécessaire de comprendre le principe de fonctionnement du traitement générique parallèle, puisque la rasterisation de primitive n’en est qu’une aggrégation, sur le plan théorique du moins — en pratique, elle en est également une spécialisation et une optimisation, puisqu’elle met à contribution des unités matérielles dédiées dont le fonctionnement exact n’est pas publié par les fabricants (le travail de Laine et Karras [LK11] montre en quoi le GPU ne peut se réduire à un processeur de traitement parallèle ; citons également le travail de Loop et Eisenacher [LE09]).

Seuls les détails importants concernant le traitement générique parallèle sont abordés dans

ce chapitre. Nous redirigeons le lecteur intéressé par davantage de détails sur des ouvrages ou articles spécialisés [NVI07, SGS10].



FIGURE 3.5 – Architecture physique du GPU GF100 fabriqué par Nvidia.

3.2.1 Un GPU difficile à configurer pour un travail précis

Le GPU repose sur une architecture physique dont un exemple est représenté sur la Figure 3.5. Elle est conçue pour exécuter efficacement un grand nombre d’opérations identiques en concurrence.

A chaque appel d’une fonction exécutant un traitement, que ce traitement soit une rasterisation ou bien un traitement générique parallèle, le GPU va dispatcher des ordres de traitement à ses différentes unités. Un traitement dans son ensemble est dénommé *grille*, segmenté en *blocs* de tailles égales. Pour un bloc de taille n , le GPU a vocation à exécuter ses n threads en parallèle. Les GPU sont limités à un parallélisme de quelques milliers de threads à l’heure actuelle et sérialisent donc les appels en essayant d’occuper au mieux l’ensemble de leurs ressources pour maximiser leur rendement. Pour exécuter un traitement générique parallèle, le programmeur spécifie lui-même la taille de la grille en nombre de blocs, et la taille d’un bloc en nombre de threads. C’est donc lui qui maîtrise l’utilisation des ressources du GPU. La raison même d’être de la grille et des blocs, et le paramétrage au sens large du GPU, s’explique à cause d’un dilemme résidant dans un double objectif que ce dernier se propose de réaliser. D’une part, les traitements initiés sur le GPU se veulent granulaires et indépendants — ils doivent pouvoir se matérialiser sous forme de micro-traitements indépendants les uns des autres et donc facilement parallélisables. Cela incite les fabricants à décomposer leur architecture en plusieurs sous-unités de traitement indépendantes, disposant de leur propre mémoire locale. D’autre part, ces micro-unités de traitements doivent pouvoir communiquer les unes avec les autres, tout du moins dans une proportion bien déterminée : celle d’un bloc. Communiquer peut être utile pour par exemple mettre en mémoire locale des informations fréquemment utilisées, ou mutualiser certains calculs lourds à réaliser et dont l’exécution répétée, aussi granulaire soit-elle, serait préjudiciable pour les performances. Le dilemme de configuration du GPU est ainsi posé, puisque une forte mutualisation des efforts

de traitement consiste à définir des blocs de grande envergure, tandis qu'un meilleur rendement est atteint en évitant de créer des dépendances entre les traitements, c'est-à-dire en évitant de mutualiser des efforts.

Pour les ordres de rastérisation, aucune taille de grille ou de blocs n'est spécifiée. Une unité spécialisée du GPU est chargée de diviser automatiquement la rastérisation en un ensemble de travaux génériques et de maximiser le rendement d'utilisation de toutes les ressources. De même, étant donné que l'ordonnancement des différentes étapes impliquées par le pipeline de rastérisation est maîtrisé puisqu'il répond à une interface de programmation parfaitement connue (pipeline de la Figure 3.1 du Chapitre précédent), le GPU dispose d'unités dédiées pour par exemple mettre en cache des informations critiques ou pour faciliter la répartition optimale et par ailleurs dynamique du travail sur différentes sous-unités internes. C'est le cas par exemple des unités de rastérisation mettant en cache le résultat de rastérisation de chaque fragment, et répartissant le travail de rastérisation entre les différents groupes de processeurs de flux.

Comme nous l'avons évoqué auparavant, l'exécution d'un traitement générique parallèle oblige le développeur à spécifier un nombre de blocs et de threads ad-hoc pour le travail à réaliser. Si l'appel à des ordres de rastérisation prémuni le développeur d'un paramétrage explicite et fastidieux du GPU, sans nuire au rendement, le problème du rendement se pose néanmoins au niveau de granularité le plus petit de l'interface de programmation graphique, c'est-à-dire au niveau des shaders. Notamment, il se pose au niveau de la convergence ou de la divergence d'exécution, que nous abordons dans la section suivante.

3.2.2 Convergence et divergence d'exécution

Les GPU ont un fonctionnement fondamentalement différent des CPU. Les cœurs présents dans les CPU exécutent les instructions de threads différents d'une manière réellement indépendante. Avec les GPU, les processeurs de flux d'une même sous-unité ne peuvent exécuter qu'une même instruction au même moment. Chaque divergence d'exécution oblige le GPU à exécuter les blocs d'instruction associés à tour de rôle, l'un après l'autre, pénalisant considérablement les performances.

Cette limitation peut avoir d'importantes conséquences au niveau du shader de fragment (Figure 3.6), fortement sollicité. Toute divergence d'exécution dans le traitement de fragments voisins peut ainsi avoir pour effet de forcer le GPU à séquentialiser un à un les traitements associés à chaque fragment. Ce mode de fonctionnement impose donc une certaine cohérence dans le traitement affecté à chaque fragment.

Pour les autres types de shader comme par exemple le shader de sommet ou le shader d'évaluation des sommets de tessellation, il est préférable d'éviter d'évaluer des primitives de types différents qui peuvent être à l'origine d'importantes divergences d'exécution. Spécialiser le code d'évaluation des sommets pour chaque primitive reste possible, voire parfois même préférable, dans la mesure où les traitements de toutes les primitives d'un même type peuvent être regroupés, et exécutés à tour de rôle.

Pour les traitements génériques parallèles, les blocs et threads associés ont tout autant intérêt à éviter des divergences d'exécution, puisque le programmeur, partitionnant explicitement le travail sur le GPU, se confronte ainsi frontalement aux problèmes de divergences, là où les couches logicielles de rastérisation et de tessellation peuvent potentiellement essayer de s'affranchir des effets néfastes de la divergence, en théorie tout du moins, étant des couches logicielles de plus haut niveau.

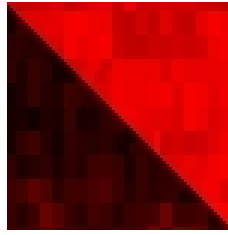


FIGURE 3.6 – Sur cette figure, une couleur différente est affectée à chaque fragment au moyen d’un compteur atomique. La coloration permet de dissocier les warps les uns des autres, plus petite unité d’exécution des GPU de chez Nvidia — les fragments d’une primitive sont en effet traités par paquets de $4 \times 8 = 32$ pixels sur le GPU ici utilisé, un GF100.

3.2.3 Des algorithmes peu adaptés à une exécution sur GPU... ou l’inverse

Certains algorithmes, par leur nature, se prêtent peu à une exécution parallèle (Section 3.1.2). Il est parfois possible de repenser un algorithme pour qu’il puisse être parallélisable. Ce n’est en revanche pas toujours possible, et augmente parfois considérablement la complexité de sa mise œuvre. De nombreux algorithmes générant un nombre variable de données, typiquement représentables sous forme d’un flux de données de sortie, peuvent poser des difficultés d’implémentation sur GPU. En effet, même si les données d’entrée peuvent être traitées parallèlement, par blocs indépendants, il est difficile d’aligner le stockage des données de sortie car leur emplacement n’est pas connu avant exécution complète de l’algorithme. Des méthodes d’exécution en deux temps et des algorithmes spécialement prévus pour gérer ces situations existent [HSO08] et, bien qu’ils soient efficaces à mettre en place pour de grands jeux de données sur GPU, ils sont lourds à mettre en œuvre et restent peu efficaces pour de petits jeux de données.

D’une manière générale, il est difficile de porter sur GPU des algorithmes récursifs. Les GPU nécessitent un partitionnement particulier de leurs ressources pour exécuter un algorithme donné avec un bon rendement, et où les threads doivent effectuer des traitements sur des jeux réduits de données, de taille constante. Les algorithmes récursifs divisent les jeux de données en 2 et bouleversent ce mode de fonctionnement, qui devient inadapté. L’incapacité du GPU à générer des sous-traitements en repartitionnant ses ressources peut rendre inefficace l’implémentation de ces algorithmes.

3.2.4 Des ressources limitées

L’utilisation judicieuse des différentes banques mémoire, que ce soit celles spécifiques aux blocs, performantes, ou bien les banques de mémoires globales et leurs caches associés, est nécessaire à une bonne exploitation du GPU, comme c’est le cas avec un CPU vis-à-vis de ses différents caches. Notamment, les accès à la mémoire globale sont relativement coûteux et, quand bien même le GPU dispose de mécanismes permettant de pallier des états de latence pour en minimiser voire en contourner les effets, ses capacités ont des limites et les accès à la mémoire doivent être judicieusement coordonnés. De même, le nombre de registres est limité. Un programme GPU doit utiliser le moins de registres possible pour affiner la granularité d’exécution et donc maximiser les effets bénéfiques du parallélisme. Cette contrainte est due au fait que les différents blocs doivent utiliser leurs propres registres mis à disposition par le GPU, en quantité restreinte. Moins de registres signifie moins de variables utilisées au même instant par un programme. Les tableaux de valeurs doivent ainsi être évités si leur taille est trop grande, car leur utilisation peut entraîner la génération d’un code allouant un nombre de registres égal à la taille

du tableau, augmentant le nombre de registres utilisés, réduisant le nombre de blocs pouvant tourner en simultané, et par conséquent diminuant les effets bénéfiques du parallélisme.

Deuxième partie

Rendu de modèles B-Rep - un état de l'art

Chapitre 1

Evaluation et échantillonnage de points et de dérivées

Nous avons donné dans la Partie I, Chapitre 2 une brève description des données géométriques définies dans les modèles B-Rep. Nous avons vu dans les Sections 2.1.2 et 2.1.3 que les surfaces support, comme les courbes de découpe, peuvent être converties sous forme de NURBS sans perte d'information. Les NURBS peuvent à leur tour être décomposées en courbes ou patches de Bézier rationnels, rendant l'évaluation et le calcul de dérivées plus aisé (des méthodes optimisées d'évaluation de courbes et surfaces NURBS existent [AGM06, KKM07] mais restent notoirement plus lentes que les méthodes utilisant des courbes et surfaces de Bézier).

Cette versatilité rend incontournable les courbes et surfaces de Bézier rationnelles et le présent chapitre a pour but de présenter des méthodes d'évaluation de données sur ces entités.

1.1 Evaluation

Nous devons à la fois évaluer des points sur des surfaces de Bézier rationnelles, ainsi que sur les courbes de Bézier rationnelles, qui représentent la découpe dans l'espace paramétrique des surfaces. Nous avons également besoin des dérivées partielles sur les surfaces, nécessaires pour calculer les normales pour prendre en charge l'éclairage, mais également pour utiliser des méthodes de rendu basées sur le lancer de rayon. Ces méthodes utilisent des algorithmes de recherche de racine comme l'itération de Newton, nécessitant les dérivées partielles.

Rappelons tout d'abord la définition d'une courbe de Bézier rationnelle, donnée dans l'équation (1.1).

$$C(s) = \frac{\sum_{i=0}^m w_i P_i B_i^m(s)}{\sum_{i=0}^m w_i B_i^m(s)} \quad (1.1)$$

où P_i sont les points de contrôle, w_i sont les poids et $B_i^m(s)$ sont des polynômes de Bernstein.

La définition d'une surface de Bézier est donnée dans l'équation (1.2).

$$P(s, t) = \frac{\sum_{i=0}^m \sum_{j=0}^n w_{ij} P_{ij} B_i^m(s) B_j^n(t)}{\sum_{i=0}^m \sum_{j=0}^n w_{ij} B_i^m(s) B_j^n(t)} \quad (1.2)$$

où P_{ij} sont les points de contrôle, w_{ij} sont les poids et $B_i^m(s)$ et $B_j^n(t)$ sont des polynômes de Bernstein de degrés respectifs m et n . Les polynômes de Bernstein sont définis pour rappel dans l'équation (1.3).

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad (1.3)$$

Plusieurs méthodes existent pour évaluer des points sur des courbes de Bézier, dont l'incontournable algorithme de De Casteljau. Cet algorithme oblige à définir un nombre de variables temporaires égal au degré de la courbe à évaluer (au degré maximum, dans le cas des surfaces), ce qui peut nuire au parallélisme du GPU (Partie I, Section 3.2). Sederberg [Sed95] propose une modification de l'algorithme de Horner, normalement utilisé pour évaluer les polynômes en base des puissances, pour qu'il fonctionne en base de Bernstein. Grâce à cela, la méthode de Horner peut être utilisée pour évaluer des points sur une courbe de Bézier. Sederberg s'appuie sur la relation de récurrence suivante des coefficients binomiaux

$$\binom{n}{i} = \frac{n-i+1}{i} \binom{n}{i-1} \quad (1.4)$$

Comme il le note, il est possible de définir une courbe de Bézier rationnelle avec la notation

$$p(t) = \Pi(P(t)) \quad (1.5)$$

avec

$$P(t) = (P_x(t), P_y(t), P_w(t)) = \sum_{i=0}^n P_i B_i^n(t) \quad (1.6)$$

et où $P_i = w_i(x_i, y_i, 1)$ et l'opérateur de projection Π est défini comme suit : $\Pi(x, y, w) = (x/w, y/w)$.

La méthode pour évaluer un point sur la courbe avec la méthode de Horner apparaît dans l'Algorithme 3, où p est de dimension 4 et doit ensuite être projeté dans R^3 avec Π .

entrée : points de controle p , degré n , paramètre t
sortie : point sur la courbe

```

1  $u \leftarrow 1.0 - t;$ 
2  $bc \leftarrow 1.0;$ 
3  $tn \leftarrow 1.0;$ 
4  $tmp \leftarrow p[0] * u;$ 
5 for  $i \leftarrow 1$  to  $n - 1$  do
6    $tn \leftarrow tn * t;$ 
7    $bc \leftarrow bc * (n - i + 1) / i;$ 
8    $tmp \leftarrow (tmp + tn * bc * p[i]) * u;$ 
9 end
10 return  $tmp + tn * t * p[n]$ 

```

Algorithme 3 : Algorithme de Horner pour l'évaluation d'un point sur une courbe de Bézier.

De la même manière qu'une courbe rationnelle peut être définie avec les équations (1.5) et (1.6), une surface peut être définie avec

$$p(s, t) = \Pi(P(s, t)) \quad (1.7)$$

et

$$P(s, t) = \sum_{i=0}^m \sum_{j=0}^n P_{ij} B_i^m(s) B_j^n(t) \quad (1.8)$$

Toujours selon Sederberg, $P(s, t)$ peut être défini avec la formulation suivante :

$$P(s, t) = (1-s)(1-t)P^{00}(s, t) + s(1-t)P^{10}(s, t) + (1-s)tP^{01}(s, t) + stP^{11}(s, t) \quad (1.9)$$

L'algorithme de Horner pour les surfaces rationnelles est mis en œuvre dans l'équation suivante :

$$\hat{P}^{kl}(u, v) = \sum_{i=k}^{m+k-1} \sum_{j=l}^{n+l-1} \hat{P}_{ij}^{kl} u^{i-k} v^{j-l}, \quad k, l = 0, 1 \quad (1.10)$$

où $u = s/(1-s)$, $v = t/(1-t)$ et $\hat{P}_{ij}^{kl} = \binom{m-1}{i-k} \binom{n-1}{j-l} P_{ij}$. Cette formulation ne marche que si $s \neq 1$ et $t \neq 1$, et peut souffrir de problèmes de précision à l'implémentation lorsque s et t s'approchent de 1. Pour éviter ces problèmes, il est possible de passer par une représentation *duale* de l'équation (1.10), tel que Pavlidis l'explique dans son évaluation pour les courbes rationnelles [Pav82, p. 226].

Pour calculer les dérivées partielles, Floater [Flo92] propose la formulation suivante de la dérivée pour les courbes de Bézier :

$$P'_{i,k}(t) = n \frac{w_{i,k-1} w_{i+1,k-1}}{w_{i,k}^2} (P_{i+1,k-1}(t) - P_{i,k-1}(t)) \quad (1.11)$$

où $w_{i,k} = (1-t)w_{i+1,k-1} + tw_{i,k-1}$. $P_{i,k}$ et $w_{i,k}$ représentent le i -ème point de contrôle sur la k -ième itération de De Casteljau et où $P(t) = P_{0,n}(t)$. Par extension aux surfaces, nous pouvons dès lors nous appuyer sur cette formulation et sur l'équation (1.10) pour déduire les deux dérivées partielles en (s, t) [Sed95] :

$$\frac{dp(s, t)}{ds} = n \frac{R_w^s(s, t) Q_w^s(s, t)}{((1-t)Q_w^s(s, t) + tR_w^s(s, t))^2} (r^s(s, t) - q^s(s, t)) \quad (1.12)$$

$$\frac{dp(s, t)}{dt} = n \frac{R_w^t(s, t) Q_w^t(s, t)}{((1-t)Q_w^t(s, t) + tR_w^t(s, t))^2} (r^t(s, t) - q^t(s, t)) \quad (1.13)$$

où

$$Q_w^s(s, t) = (1-t)P^{00}(s, t) + tP^{01}(s, t) \quad (1.14)$$

$$R_w^s(s, t) = (1-t)P^{10}(s, t) + tP^{11}(s, t) \quad (1.15)$$

$$Q_w^t(s, t) = (1-s)P^{00}(s, t) + sP^{10}(s, t) \quad (1.16)$$

$$R_w^t(s, t) = (1-s)P^{01}(s, t) + sP^{11}(s, t) \quad (1.17)$$

1.2 Echantillonnage de points sur les surfaces support

Tout échantillonnage doit respecter un ou plusieurs critères qualitatifs. Pour la discrétisation de point sur les faces, on en dénombre deux principaux dans l'état de l'art. D'une part, le critère

de distance entre deux points échantillonnés sur la surface, que nous dénommerons critère de la *taille des arêtes* (Section 1.2.1). De l'autre, la *déviaton maximale* de la surface par rapport à son approximation linéaire (Section 1.2.2). Distance et déviation sont idéalement définies par l'utilisateur sous forme de valeurs en espace écran. Une fois fixées, elles doivent être converties en espace paramétrique sur la surface sous forme de *pas de discrétisation*. Deux pas distincts peuvent être utilisés, un sur u et un sur v .

1.2.1 Critère qualitatif de la taille des arêtes

Dans le travail de Rockwood et al., le critère de la taille des arêtes est retenu. Une valeur appelée TOL , exprimée en espace écran, est spécifiée par l'utilisateur pour définir la taille maximale d'une arête de discrétisation reliant deux points échantillonnés sur la surface de Bézier rationnelle support, parallèlement à un des deux axes paramétriques (u ou v). Un nombre de segments de discrétisation uniformes ci-après dénommés n_u et n_v , reliant $n_u + 1$ points sur l'axe u et $n_v + 1$ points sur l'axe v , sont calculés à partir de l'équation donnée en (1.18). Dans cette équation, les points de contrôle r_{ij} et poids associés w_{ij} subissent la transformation définie par la matrice de vue et la projection associée pour donner R_{ij} et W_{ij} , π_3 représentant la transformation de l'espace de vue après projection vers l'espace écran.

$$\begin{aligned} n_u &= n\sqrt{2} \frac{\max(\|W_{ij}\pi_3 R_{ij} - W_{i+1,j}\pi_3 R_{i+1,j}\|)}{TOL \min(W_{ij})} & 1 \leq i \leq n-1, 1 \leq j \leq m \\ n_v &= m\sqrt{2} \frac{\max(\|W_{ij}\pi_3 R_{ij} - W_{i,j+1}\pi_3 R_{i,j+1}\|)}{TOL \min(W_{ij})} & 1 \leq i \leq n, 1 \leq j \leq m-1 \end{aligned} \quad (1.18)$$

Les pas paramétriques de discrétisation peuvent ensuite être calculés : $STEP_u = 1/n_u$ et $STEP_v = 1/n_v$.

Rockwood et al. définissent également un pas $STEP_t = \min(1/n_{tu}, 1/n_{tv})$ (1.19). Pour chaque courbe de découpe, il assure que la distance écran entre la projection de deux points consécutivement discrétisés sur la courbe de degré s , de points de contrôle c_i et de poids associés w_i n'excède jamais TOL .

$$\begin{aligned} n_{tu} &= s \frac{\max(\|w_i c_i - w_{i+1} c_{i+1}\|)}{STEP_u \min(w_i)} & 1 \leq s \leq s-1 \\ n_{tv} &= s \frac{\max(\|w_i c_i - w_{i+1} c_{i+1}\|)}{STEP_v \min(w_i)} & 1 \leq s \leq s-1 \end{aligned} \quad (1.19)$$

La méthode de Rockwood et al. utilise essentiellement une représentation de la surface support en espace écran, à partir de laquelle est calculée des limites sur la valeur des dérivées. Les pas paramétriques de discrétisation sont ensuite déduits.

1.2.2 Critère qualitatif de la déviation entre la surface et son approximation

Le critère de taille des arêtes n'est pas toujours suffisant pour obtenir un bonne qualité d'affichage. En particulier, il est mal adapté aux surfaces ayant une forte courbure, localement ou globalement, comme illustré sur la Figure 1.1.

Le critère de déviation entre la surface support et sa discrétisation est utile pour conserver une bonne qualité d'affichage. Cette valeur est exprimée en nombre de pixels à l'écran. Nous voulons calculer les pas paramétriques de discrétisation sur la surface de sorte que cette dernière et son

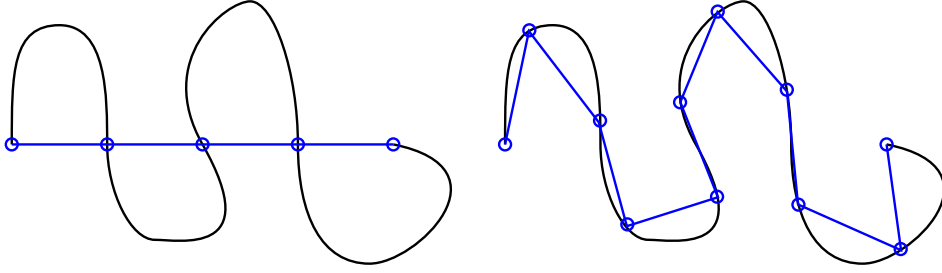


FIGURE 1.1 – Le critère de la taille des arêtes tel que proposé par Rockwood et al. ne prend pas en compte la courbure des surfaces support. Pour la courbe de forte courbure présentée sur cette figure, la discrétisation produite est celle de gauche (4 segments), alors que nous nous attendons à obtenir celle de droite (10 segments). Notons que, dans un cas comme dans l'autre, la déviation entre la surface et son approximation linéaire n'est pas maîtrisée.

approximation linéaire ne dévient pas de plus de k pixels écran. Il existe plusieurs méthodes pour calculer des pas de discrétisation à partir d'une déviation exprimée en espace objet. Nous verrons ensuite comment calculer une déviation en espace objet à partir d'une déviation exprimée en espace écran.

Filip et al. [FMM87] proposent une méthode pour borner la déviation maximale en espace objet ϵ d'une surface S par rapport à son approximation linéaire L . Il est tout d'abord établi la relation décrite dans la formule (1.20), où l_1 et l_2 désignent le pas paramétrique sur u et v .

$$\epsilon = \sup_{(u,v) \in T} \|S(u,v) - L(u,v)\| \leq \frac{1}{8}(l_1^2 M_1 + 2l_1 l_2 M_2 + l_2^2 M_3) \quad (1.20)$$

avec

$$M_1 = D_{uu} = \sup_{u,v \in [0,1]} \left\| \frac{d^2 S(u,v)}{du^2} \right\|, \quad M_2 = D_{uv} = \sup_{u,v \in [0,1]} \left\| \frac{d^2 S(u,v)}{dudv} \right\|, \quad M_3 = D_{vv} = \sup_{u,v \in [0,1]} \left\| \frac{d^2 S(u,v)}{dv^2} \right\| \quad (1.21)$$

Filip et al. fournissent un moyen de calculer M_1 , M_2 et M_3 à partir des points de contrôle de Bézier et des poids rationnels. Des travaux récents proposent de calculer des bornes plus étroites pour ces valeurs [Sel05, DL13]. Ensuite, un moyen de calculer les pas paramétriques de discrétisation $n = STEP_u$ et $m = STEP_v$ de sorte que la déviation ϵ soit toujours respectée, est proposé.

$$\frac{1}{8} \left(\frac{1}{n^2} M_1 + \frac{2}{nm} M_2 + \frac{1}{m^2} M_3 \right) = \epsilon \quad (1.22)$$

Une méthode est ensuite donnée pour déterminer n et m en tenant compte des différentes valeurs possibles pour M_1 , M_2 et M_3 . Dans le cas le plus général où $M_1 > 0$ et $M_3 > 0$, on obtient m via l'équation (1.23), et avec $k = M_1/M_3$, on en déduit $n = km$.

$$m = \sqrt{\frac{1}{8k^2 \epsilon} (M_1 + 2kM_2 + k^2 M_3)} \quad (1.23)$$

Guthe et al. [GBK05] simplifient le calcul de $n = n_u$ et $m = n_v$ en se basant sur le fait que $n_u n_v \leq \frac{1}{2}(n_u^2 + n_v^2)$, et en reformulant (1.20).

$$\epsilon \leq \frac{1}{8} (n_u^2 (M_1 + M_3) + n_v^2 (M_2 + M_3)) \quad (1.24)$$

Abi-Ezzi [AES91] proposent une amélioration du travail de Filip et al (1.25).

$$n_d^u = \frac{\sqrt{D_{uu}D_{vv} + D_{uv}\sqrt{D_{uu}D_{vv}}}}{2\sqrt{TOLD_{vv}}}, \quad n_d^v = \frac{\sqrt{D_{uu}D_{vv} + D_{uv}\sqrt{D_{uu}D_{vv}}}}{2\sqrt{TOLD_{uu}}} \quad (1.25)$$

Kumar et Manocha [KM94] proposent également une amélioration

$$n_u = \frac{\left\| \frac{X(u,v)}{W(u,v)_u}, \frac{Y(u,v)}{W(u,v)_u}, \frac{Z(u,v)}{W(u,v)_u} \right\|}{TOL}$$

$$n_v = \frac{\left\| \frac{X(u,v)}{W(u,v)_v}, \frac{Y(u,v)}{W(u,v)_v}, \frac{Z(u,v)}{W(u,v)_v} \right\|}{TOL} \quad (1.26)$$

où $\frac{X(u,v)}{W(u,v)_u}$ définit la magnitude maximale de la dérivée partielle $\frac{X(u,v)}{W(u,v)}$ par rapport à u sur le domaine $(0, 1) \times (0, 1)$.

D'autres méthodes existent pour déterminer la longueur maximale autorisée d'une arête sur une surface en regard de la déviation maximale autorisée entre la surface et son approximation linéaire. Citons notamment la méthode proposée par Zheng et Sederberg [ZS00](1.27).

$$\delta_u = \sqrt{\frac{4D_{vv}\epsilon \min w_{ij}}{D_{uu}D_{vv} + D_{uv}\sqrt{D_{uu}D_{vv}}}}$$

$$\delta_v = \sqrt{\frac{4D_{uu}\epsilon \min w_{ij}}{D_{uu}D_{vv} + D_{uv}\sqrt{D_{uu}D_{vv}}}}$$

$$\delta_u = \delta_v = \sqrt{\frac{4\epsilon \min w_{ij}}{D_{uv}}} \quad \text{si } D_{uu} = D_{vv} = 0 \text{ et } D_{uv} \neq 0 \quad (1.27)$$

D_{uu} , D_{uv} et D_{vv} peuvent être calculés avec les formules données par (avec $r = \max_{\substack{0 \leq i \leq n \\ 0 \leq j \leq m}} \|P_{ij}\|$) :

$$D_{uu} = n(n-1) \max_{\substack{0 \leq j \leq n-2 \\ 0 \leq i \leq m}} (\|w_{i+2,j}P_{i+2,j} - 2w_{i+1,j}P_{i+1,j} + w_{i,j}P_{i,j}\| + (r-\epsilon)|w_{i+2,j} - 2w_{i+1,j} + w_{i,j}|)$$

$$D_{uv} = nm \max_{\substack{0 \leq i \leq n-1 \\ 0 \leq j \leq m-1}} (\|w_{i+1,j+1}P_{i+1,j+1} - w_{i+1,j}P_{i+1,j} + w_{i,j}P_{i,j}\| + (r-\epsilon)|w_{i+1,j} - w_{i+1,j} + w_{i,j}|)$$

$$D_{vv} = n(m-1) \max_{\substack{0 \leq i \leq m-2 \\ 0 \leq j \leq n}} (\|w_{i,j+2}P_{i,j+2} - 2w_{i,j+1}P_{i,j+1} + w_{i,j}P_{i,j}\| + (r-\epsilon)|w_{i,j+2} - 2w_{i,j+1} + w_{i,j}|)$$

Jusqu'à présent la déviation maximale est exprimée en espace objet. Nous voulons l'exprimer en espace écran. Typiquement, nous souhaitons que la surface et son approximation linéaire ne devienne pas de plus de k pixels écran. La littérature est peu bavarde dans ce domaine. Bien que passée sous silence, la solution généralement adoptée est de partir du principe que le rapport entre la déviation écran et celle en espace objet est le même que celui existant entre les deux mesures d'une même longueur entre deux points — l'une en espace objet et l'autre en espace écran. Les points de référence pour la mesure doivent être choisis dans la vue dans un esprit conservatif. Les boîtes englobantes ou polygones de contrôle sont généralement utilisés.

Abbi-Ezzi et Shirman [AES91] s'intéressent spécifiquement à ce problème de mise à l'échelle. Ils proposent de passer par un système de coordonnées spécifique, baptisé *système de coordonnées*

d'éclairage, qui dépend de la transformation de la vue. La définition des surfaces (eg. points de contrôle) est d'abord convertie dans ce système pour chaque rendu à effectuer. Ils proposent ensuite de calculer la limite supérieure d'un facteur de mise à l'échelle à employer pour convertir la déviation du système de coordonnées écran (en pixels) vers le système de coordonnées d'éclairage (1.28).

$$\xi_{max,Q}^2(A) = \frac{P_z^2}{2d^4} (X_4 + X_1 + \sqrt{(X_4 - X_1)^2 + 4X_1X_2}), \quad \text{où}$$

$$X_1 = d^2 s^2, \quad X_2 = h^2 s^2, \quad X_3 = P_z^2 s_z^2, \quad X_4 = X_2 + X_3. \quad (1.28)$$

Les valeurs pour d et h^2 dépendent de la définition de la surface et sont données dans l'équation (1.29). W_k représentent les quatre coins de la fenêtre de vue. B_{ij} sont les points de contrôle du patch de Bézier utilisé. A représente un point dont la distance avec P , origine de vue, doit être étudiée, avec $A_z \neq P_z$, et d_n représentant la distance entre P et le plan de projection (*near plane*).

$$\tau(A) = \frac{A_x^2 + A_y^2}{(P_z - A_z)^2}$$

$$t = \min(\max_{i,j}(\tau(B_{ij})), \max_k(\tau(W_k)))$$

$$d = \max(\min_{i,j}(P_z - B_{ij,z}), d_n)$$

$$h^2 = d^2 t \quad (1.29)$$

La discrétisation des points sur la surface est réalisée dans le système de coordonnées d'éclairage, puisque la déviation est exprimée dans ce système. Une fois effectuée, Abbi-Ezzi et Shirman fournissent une transformation matricielle permettant de convertir les primitives générées en espace écran.

Les travaux de Yeo et al. [YBP12], se basant sur ceux de Lutterkort et Peters [LP99], s'intéressent également à la déviation entre une surface et son approximation linéaire. Ils proposent le concept de *slefes*. Les *slefes*, abbréviation de *Subdividable Linear Efficient Function Enclosures* sont des constructions géométriques linéaires construites à partir de surfaces de Bézier rationnelles arbitraires. Des *slefe-boxes* sont créées uniformément sur la surface, à des espaces paramétriques réguliers, suivant un pas déterminé. Ces entités, une fois reliées entre elles par construction d'une enveloppe locale dénommée *tuile* ou *slefe-tile*, forment une enveloppe globale contenant la surface de Bézier rationnelle à partir de laquelle elles ont été construites. Les *slefes* sont illustrées en 2D sur la Figure 1.2. La construction de chaque tuile se fait avec deux *slefe-boxes* consécutives en regard de l'orientation paramétrique de la courbe.

La Figure 1.3 illustre les tuiles, construites à partir des *slefe-boxes*, avec une surface de Bézier. Les tuiles sont alors créées à partir de quatre *slefe-boxes* adjacentes. Chaque tuile englobe localement la surface.

Lorsque m *slefes* sont créées sur un domaine U d'une surface de Bézier p , il est facile de calculer $w(m, U, p)$, identifiant la largeur maximale (ou épaisseur) de chaque tuile créée. En effet, cette largeur est bornée par les dimensions des *slefe-boxes* ayant servies à sa construction.

Yeo et al. précalculent $n \times m$ *slefe-boxes* pour chaque surface de Bézier reçue en entrée. Au moment du rendu, ils en effectuent la projection à l'écran, dont ils conservent le rectangle englobant. Ils parcourent chaque *slefe-box* projetée, et gardent l'arête la plus grande de tous les rectangles englobants, dont la longueur est dénotée $\bar{w}(m, U, p)$.

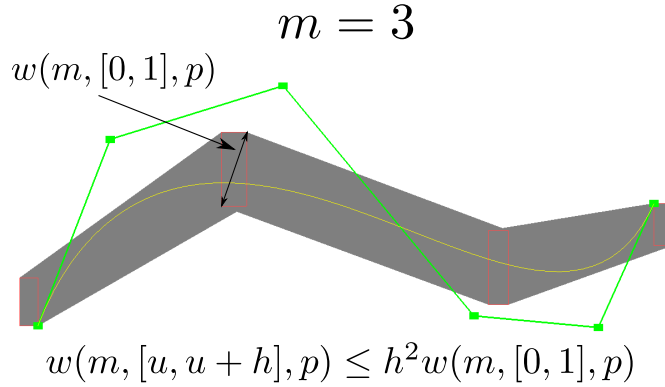


FIGURE 1.2 – Création de $m = 3$ tuiles slefe englobant une courbe de Bézier. Jaune : courbe de Bézier. Vert : polygone de contrôle. Rectangles rouges : slefe-boxes. Zone grise : espace contenu dans l'union des tuiles créées en reliant deux slefe-boxes consécutives.

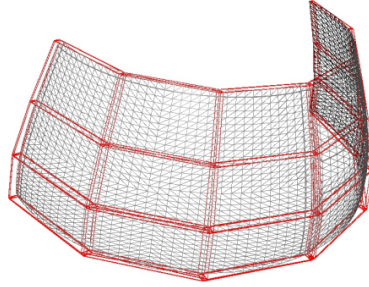


FIGURE 1.3 – Tuiles (slefe-tiles), affichées en rouge. L'union de toutes ces tuiles englobe la surface de Bézier.

Selon le travail de Lutterkort et Peters [LP99], la fonction w obéit à la relation exposée dans la formule (1.30).

$$w(m, [u, u+h], p) \leq h^2 w(m, [0, 1], p) \quad (1.30)$$

Il est dès lors possible de calculer h , identifiant le pas de discrétisation dans une *slefe-tile* donnée pour que la déviation écran $\epsilon_s = \bar{w}(m[u, u+h], p)$ soit respectée (1.32).

$$\sqrt{\frac{\epsilon_s}{\bar{w}(m, [0, 1], p)}} \leq h \quad (1.31)$$

La *slefe-tile* ayant la plus grande largeur écran est sélectionnée. Il est possible de tesseller chaque tuile de p avec un pas paramétrique de valeur h pour respecter \bar{w} . Au final, la surface est discrétisée en $n_u \times n_v$ segments uniformes, ou $n_u = n_v = \bar{\tau}_{xy}(m, p) = 1/h$.

Signalons pour finir que la précision de couverture sur la profondeur de vue après projection n'est pas prise en compte dans les formules précédemment citées. Pour s'assurer qu'une tolérance sur la profondeur tol_z est respectée, un pas paramétrique h_z est défini comme suit

$$\sqrt{\frac{tol_z}{w_z(m, [0, 1], p)}} \leq h_z \quad (1.32)$$

Le pas paramétrique à utiliser pour la tessellation afin de respecter à la fois la déviation écran ϵ_s et celle sur la profondeur tol_z devient $h_{xyz} = \min(h, h_z)$.

Chapitre 2

Rendu par tessellation statique globale

2.1 Principe général

Le rendu par tessellation statique globale s'intéresse à effectuer la tessellation individuelle de chaque face (Figure 2.1).

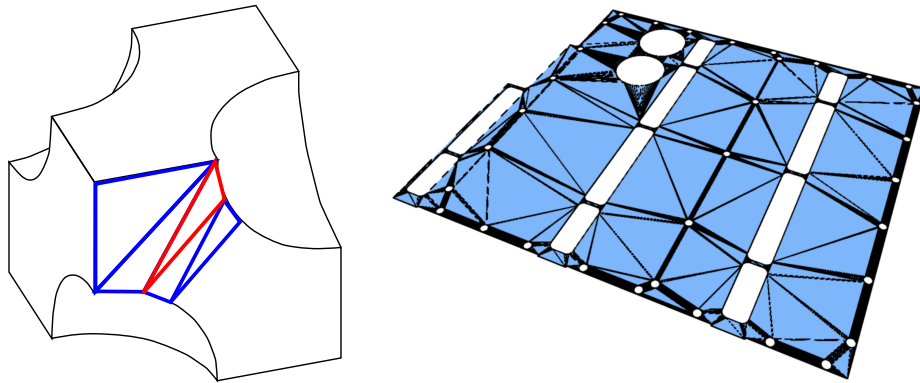


FIGURE 2.1 – A gauche : tessellation individuelle d'une face B-Rep, qui sera répétée pour chaque face. Les sommets et triangles générés le long de la surface support suivent les courbes de découpe. A droite : la tessellation d'une face peut générer un très grand nombre de triangles.

2.2 Tessellation des faces

L'un des principaux travaux cités comme référence dans le domaine de la tessellation de modèles B-Rep est celui réalisé par Rockwood et al. [RHD89].

Les différentes faces B-Rep sont tout d'abord examinées et les surfaces support sont converties en NURBS. Les NURBS sont ensuite décomposées en patches de Bézier rationnels [CLR80, Far90]. Les régions de découpe résultantes sont ensuite fermées dans leurs patches respectifs, par création de courbes de découpe linéaires additionnelles (Figure 2.2).

Chaque espace de découpe obtenu est ensuite décomposé en régions uv -monotones. Selon Rockwood et al., une région est dite monotone sur un axe paramétrique (u ou v) si toute ligne perpendiculaire à cet axe a une intersection convexe avec cette région, en d'autres termes, que cette intersection est une ligne unique, continue, ou bien un point. Une région monotone sur les

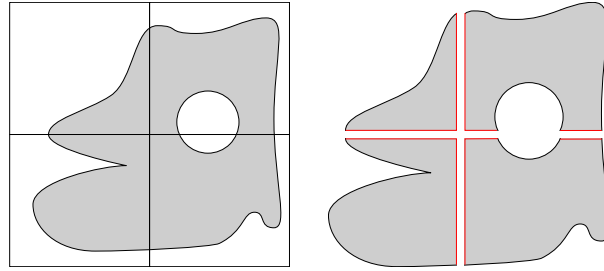


FIGURE 2.2 – Les surfaces support, exprimées sous forme de NURBS, sont décomposées en patches de Bézier rationnels. L'espace de découpe des patches correspondants est fermé par des courbes de découpe additionnelles, affichées en rouge.

axes paramétriques u et v est dite uv -monotone. La décomposition de l'espace paramétrique en zones uv -monotones est illustré sur la Figure 2.3.

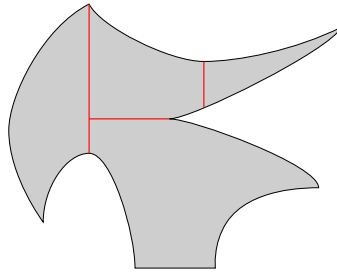


FIGURE 2.3 – L'espace paramétrique des patches de Bézier est découpé en régions uv -monotones (4 régions ici).

Une grille représentant la discrétisation de l'espace paramétrique telle qu'elle est définie par les pas respectifs $STEP_u$ et $STEP_v$ (Chapitre 1) est ensuite placée sur chaque région uv -monotone, en respectant un positionnement relatif à l'ensemble de la surface, $STEP_u$ et $STEP_v$ étant définis pour toute la surface support, dans son ensemble. Chaque intersection de région uv -monotone avec une case de la grille produit une région elle-même uv -monotone, et va donner lieu à la formation d'un maillage. Les courbes de découpe sont discrétisées suivant un pas $STEP_t$ tel qu'il est calculé avec l'équation (1.18) du Chapitre 1.

La méthode de triangulation finale utilisée doit fonctionner avec des polygones uv -monotones. Rockwood et al. citent celle de Garey et al. [GJPT78], de complexité $O(n \log(n))$. D'autres méthodes plus performantes existent [TvW88]. La tessellation des faces avec leur algorithme est résumée dans la Figure 2.4.

Le résultat de la tessellation avec la méthode de Rockwood et al. introduit des triangles superflus. Kumar et Manocha [KM94, Kum96] proposent une méthode qui n'a pas ce défaut (Figure 2.6). Leur algorithme n'a pas besoin de décomposer l'espace de découpe en région uv -monotones. L'espace de découpe est d'abord partitionné en $STEP_u \times STEP_v$ cellules. Pour calculer $STEP_u$ et $STEP_v$, ils utilisent le critère de la taille des triangles proposé par Rockwood et al., expliqué dans le Chapitre 1. Afin d'éviter un sous-échantillonnage pour les surfaces à forte courbure, et sans utiliser le critère de déviation de Filip et al. (Chapitre 1) qu'ils jugent peu efficace, ils prennent en compte la courbure aux coins du patch. Deux valeurs k_u et k_v (2.1) sont calculées et sont ensuite ajoutées aux pas calculés avec la méthode reposant sur la taille des arêtes. K est un facteur librement défini par l'utilisateur.

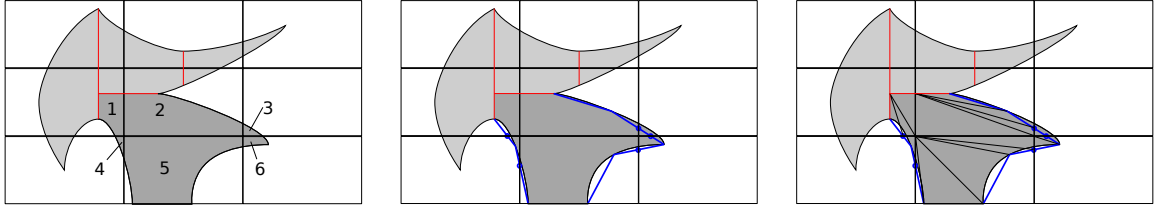


FIGURE 2.4 – Chaque région uv -monotone est tessellée pour chaque intersection avec une case de la grille de discrétisation. Gauche : la région mise en surbrillance doit être tessellée 6 fois, intersectant 6 cases de la grille. Centre : les courbes de découpe sont discrétisées en respectant leur facteur $STEP_i$ qui leur est propre. Au besoin, des points sont interpolés pour les intersections avec la grille (cercles bleus). Droite : une méthode de tessellation adaptée aux polygones monotones achève le travail.

$$\begin{aligned}
 k_u &= K \times \max \left\{ \begin{array}{l} \|F_u(\epsilon_u, 0) - F_u(0, 0)\| \\ \|F_u(\epsilon_u, 1) - F_u(0, 1)\| \\ \|F_u(1, 0) - F_u(1 - \epsilon_u, 0)\| \\ \|F_u(1, 1) - F_u(1 - \epsilon_u, 0)\| \end{array} \right\} \\
 k_v &= K \times \max \left\{ \begin{array}{l} \|F_v(0, \epsilon_v) - F_v(0, 0)\| \\ \|F_v(1, \epsilon_v) - F_v(1, 0)\| \\ \|F_v(0, 1) - F_v(0, 1 - \epsilon_u)\| \\ \|F_v(1, 1) - F_v(1, 1 - \epsilon_u)\| \end{array} \right\} \quad (2.1)
 \end{aligned}$$

Les cellules totalement hors de la zone de découpe sont ignorées, celles totalement dans la zone de découpe sont simplement tessellées avec deux triangles. Pour les cellules partiellement dans la zone de découpe, leur méthode évite de créer des sommets à la jonction entre les cellules, au prix (hélas) d'un considérable effort algorithmique, comme illustré sur la Figure 2.5. Cette stratégie leur permet de soigner la jonction entre les faces tessellées, expliquée dans la prochaine section (élimination des cracks).

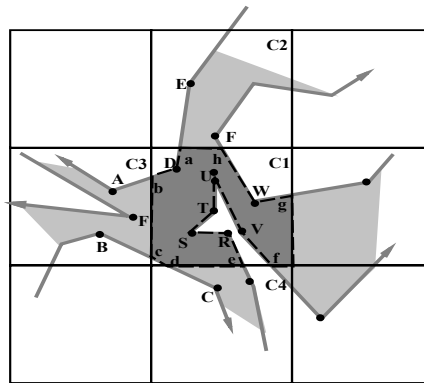


FIGURE 2.5 – Bien que basée cellule, la méthode de Kumar et Manocha produit une tessellation sans avoir à calculer et introduire les points d'intersection (a, b, c, d, e, f, g, h ici) entre les cellules et les courbes de découpe tessellées. Ceci est fait au prix d'une très significative complexité algorithmique, fonctionnant en plusieurs étapes.

Un PSLG (*Planar Straight Line Graph*) est généré pour chaque cellule. Le PSLG est ensuite

décomposé en trapézoïdes, puis les trapézoïdes sont décomposés en polygones v -monotones [Sei91].

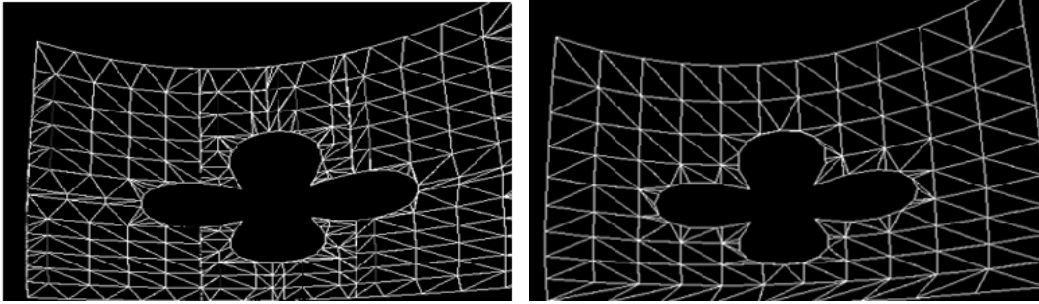


FIGURE 2.6 – Tessellation d’une face. A gauche, produite par la méthode de Rockwood et al. [RHD89]. A droite, produite par la méthode de Kumar et Manocha [KM94]. La décomposition en région uv -monotone à gauche perturbe la tessellation par rapport aux axes de paramétrisation, ce qui n’est pas le cas à droite.

Sheng et Hirsch [SH92] utilisent une autre approche. Ils effectuent une tessellation des faces avec une version modifiée de la méthode de triangulation de Delaunay, puis subdivisent les triangles obtenus pour forcer ces derniers à avoir des arêtes d’une taille maximale. Les pas paramétriques de discrétisation $STEP_u$ et $STEP_v$ ne sont pas calculés. Ils déterminent la longueur maximale Ω d’une arête d’un triangle en espace objet. Pour ce faire, ils s’appuient sur la méthode proposée par Filip et al. (qui ne marche que pour les triangles droits, puisqu’elle ne prend en compte que les discrétisations alignées sur les axes paramétriques u et v), et la généralise à n’importe quel triangle. Ils obtiennent

$$\Omega = 3\sqrt{\frac{\epsilon}{2(M_1 + 2M_2 + M_3)}} \quad (2.2)$$

Rappelons que M_1 , M_2 , M_3 et ϵ sont définis dans le chapitre précédent. Les courbes de découpe sont tout d’abord discrétisées en respectant Ω . Les surfaces NURBS prises en entrée sont décomposées en patches de Bézier, la discrétisation des courbes de découpe se voyant rajouter des sommets. Les discrétisations des découpes sont fermées patch par patch avec des lignes suivant les axes paramétriques. Ces lignes entre les patches sont discrétisées à leur tour en respectant la valeur Ω minimale entre les deux patches adjacents, ceci afin d’éviter les cracks entre les patches d’une même surface. Un PSLG est ensuite généré pour chaque patch, puis triangulé avec l’algorithme de triangulation de Delaunay, modifié pour pouvoir prendre en charge des polygones complexes arbitraires, constitués de contours externes et internes (trous). Les triangles obtenus sont ensuite subdivisés jusqu’à ce que toutes les arêtes aient une longueur inférieure ou égale à Ω (Figure 2.7).

Piegl et Richard [PR95] utilisent une méthode encore différente pour réaliser la tessellation d’une face. Ils reprennent le calcul de Sheng et Hirsch et la méthode de calcul pour Ω . Les courbes de découpe ne sont ici pas discrétisées par pas paramétrique constant. Piegl et Richard avancent le long des courbes de découpe en recherchant, itération après itération, la valeur du paramètre t à utiliser pour discrétiser les points de sorte qu’une longueur d’arête Ω soit toujours respectée. Toujours à partir de Ω , un pas de discrétisation v_{incr} et un nombre d’isoligne NSL associé, utilisés comme support pour la discrétisation par pas constant sur v , sont estimés. Des points sont ensuite discrétisés pour chacune des NSL isolignes, en intersectant ces lignes avec l’approximation linéaire des courbes de découpe. Chacune de ces intersections est matérialisée par un ensemble de segments situés à l’intérieur de la zone de découpe, le long desquels sont échantillonnés des points, de sorte qu’ Ω , le long de ces segments, soit respecté. Une fois la

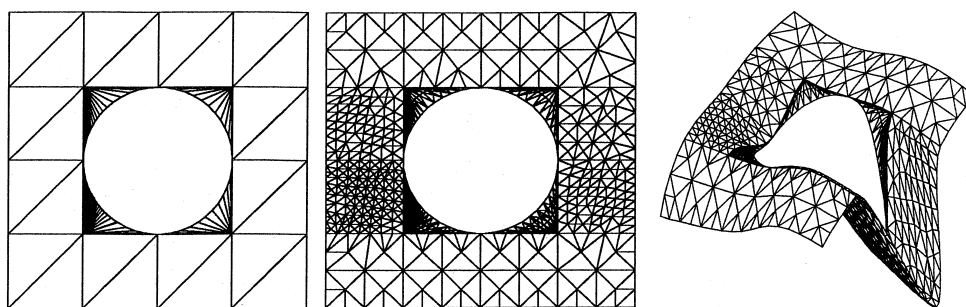


FIGURE 2.7 – Tessellation d’une surface NURBS produite par la méthode de Sheng et Hirsch. A gauche : les patches de Bézier sont tessellés individuellement avec une triangulation de Delaunay modifiée pour prendre en charge les polygones complexes. Au centre : les triangles sont ensuite subdivisés de sorte que la longueur de l’arête maximale n’excède jamais Ω , valeur propre à chaque patch (ce qui explique la densité de discrétisation, variable sur la figure du centre). A droite : tessellation obtenue.

discrétisation achevée, les différentes zones discrétisées délimitées par les points et les arêtes des courbes de découpe extérieures sont extraites. Elles sont triangulées avec une triangulation de Delaunay dont la règle du cercle circonscrit est modifiée [CR90]. L’algorithme en globalité est résumé sur la Figure 2.8.

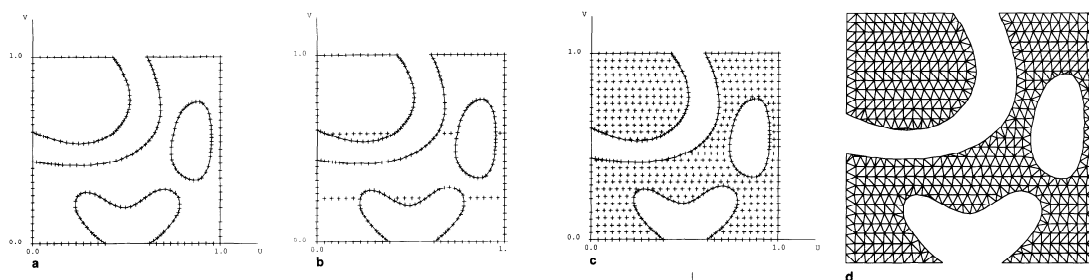


FIGURE 2.8 – Méthode de Piegl et Richard. (a) les courbes de découpe sont discrétisées avec un pas paramétrique non constant, pour respecter strictement Ω de manière la plus exacte, et non conservative. (b) pour chacune des NSL isolignes à v constant de l’espace paramétrique, des points sont créés à l’intérieur de la zone de découpe. (c) ensemble des points créé. (d) tessellation finale.

Shantz et Chang [SC88] balayent la surface dans son sens paramétrique u pour en effectuer le rendu. Initialement imaginée pour effectuer le rendu en réalisant une extraction successive de sections de courbes très rapprochées le long d’une surface, leur méthodologie peut potentiellement être utilisée pour générer une tessellation de la surface.

Les courbes de découpe sont décomposées en segments monotones sur u , puis triées dans un tableau par ordre croissant des coordonnées u minimales de départ pour chaque courbe. La surface est ensuite balayée de gauche à droite, par coordonnée u croissante, suivant un pas correspondant à $STEP_u$, sauf si le u de départ ou d’arrivée d’une courbe monotone dans le tableau précédemment mentionné se trouve sur le chemin de u , auquel u prend la valeur associée. Chaque fois qu’une telle situation se produit, est ainsi ajoutée ou enlevée une courbe de découpe u -monotone, et est donc maintenu un ensemble de courbes *actives* de découpe, qui définissent

la découpe pour une coordonnée u donnée sur toute la longueur paramétrique v , dans le sens vertical. A chaque coordonnée de $u = u_s$ où la discrétisation et la tessellation qui en découle doit être faite, sont calculées toutes les intersections entre l'isoligne paramétrique $u = u_s$ et les courbes de découpe actives. Les intersections calculées sont triées par coordonnées v croissantes, et sont marquées d'un drapeau d'information indiquant le sens de la courbe monotone intersectée (un sens u croissant signifiant que la zone dans la découpe est sous la courbe, un sens décroissant indiquant que la zone dans la découpe se trouve au dessus). Les intersections progressivement sauvegardées dans une structure au fur et à fur que la discrétisation progresse sur u servent à générer, toujours au fur et à mesure, des triangles (Figure 2.9). Leur méthode est donc complètement séquentielle.

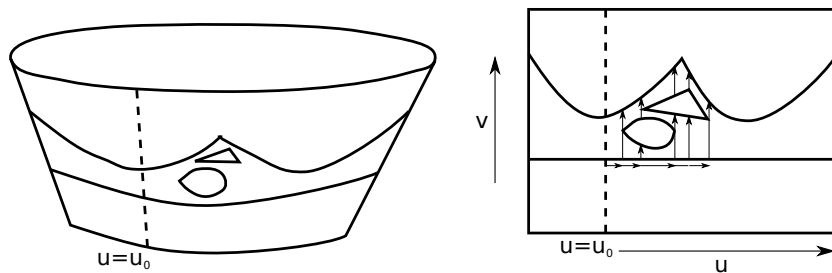


FIGURE 2.9 – La méthode de tessellation de Shantz et Chang balaie la surface dans le sens paramétrique u . u est incrémenté en fonction de $STEP_u$ ou lorsque qu'une courbe de découpe vient s'ajouter au fur et à mesure que l'on avance sur u . Pour une valeur de u donnée, les courbes de découpe actives sont maintenues dans une liste, ainsi que leur orientation. L'orientation de ces dernières permet de déterminer les parties pleines de la découpe pour un u constant. En maintenant la liste des courbes actives d'une valeur de u donnée à une autre, et en balayant la surface de gauche à droite, il est possible de tesser les espaces entre les courbes.

2.3 Elimination des cracks

Comme nous l'avons évoqué dans la Partie I, il existe un jour géométrique entre deux faces adjacentes lorsque la jointure au niveau des deux courbes de découpe sur les surfaces respectives n'est pas parfaitement continue. Nous avons vu également que, même en présence d'une jointure continue, le processus de discrétisation des sommets sur les faces et les courbes de découpe et la tessellation laisse des cracks apparaître entre les faces (Figures 2.10 et 2.11). La dimension de ces cracks est par ailleurs largement supérieure à celle des jours géométriques. L'épaisseur mesurable sur le modèle des jours géométriques n'excède généralement pas celle d'un cheveu, et rend les jours géométriques difficilement représentables graphiquement (pour peu que l'on veuille expressément les représenter).

Pour s'assurer que le maillage final du modèle résultant de la tessellation de toutes les faces B-Rep soit étanche, il faut s'assurer que chaque jonction entre deux faces adjacentes l'est effectivement. Cette étanchéité peut être assurée en tessellant les faces indépendamment les unes des autres et en générant de la géométrie additionnelle pour combler les cracks. Cette approche est inefficace et esthétiquement très discutable.

Si un jour géométrique existe au niveau de la jointure, l'empreinte en espace objet des deux découpes ne correspond pas. Ensuite, en l'absence de jour géométrique et même à niveau de discrétisation égal des courbes de découpe d'une face à l'autre ($STEP_t$), la différente paramétrisation de ces dernières, due à leur différente définition dans leur espace paramétrique respectif ne

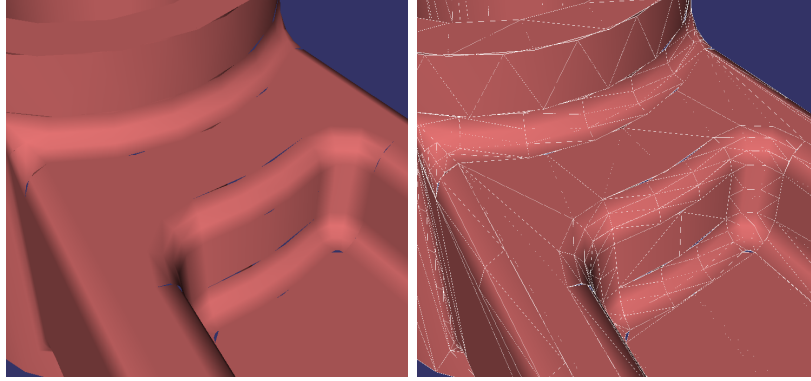


FIGURE 2.10 – La tessellation individuelle de faces B-Rep peut mener à l'apparition de cracks entre les faces.

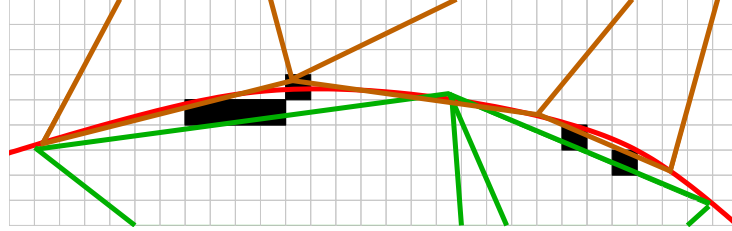


FIGURE 2.11 – La rasterisation des faces tessellées mène à l'apparition de cracks (ici les pixels noirs) entre les faces.

peut garantir que les points discrétisés se trouvent au même endroit en espace objet. Passer par une représentation commune permet de s'affranchir de ce problème. Une meilleure approche pour assurer l'étanchéité consiste donc à disposer d'une représentation commune des deux courbes de découpe définies dans l'espace paramétrique des deux faces adjacentes, de sorte que cette courbe commune, définie en espace objet (c'est-à-dire en trois dimensions, et non en espace paramétrique relativement à une surface spécifique), serve de support pour la discrétisation de sommets sur les *deux* courbes de découpe associées, chaque fois que cela est nécessaire.

Dans cette optique, Kumar [KKMN95] calcule donc une représentation de la jointure f_s en espace objet en bornant la déviation maximale ϵ entre cette représentation et les deux empreintes laissées, toujours en espace objet, par les courbes f_{d1} et f_{d2} sur leur face respective. La définition de la courbe résultante répond à la propriété suivante :

$$\|f_s(t_i) - S(f_{d_n}^u(t_i), f_{d_n}^v(t_i))\| < \epsilon \quad (2.3)$$

avec $n \in [1, 2]$, $t_i \in [0, 1]$, ϵ proche de 0 et S étant la surface support. Ensuite, les sommets peuvent être discrétisés le long de cette courbe en utilisant un pas paramétrique $STEP_t$ calculé par exemple avec la méthode de Filip et al. [FMM87] :

$$STEP_t \leq \sqrt{\frac{8\epsilon}{\sup \|f''(t)\|}} \quad (2.4)$$

Rappelons un point important dans la méthode de tessellation de faces de Kumar : il évite de discrétiser des sommets à la jonction entre les courbes de découpe et la grille de dimension $n_u \times n_v$, propre à chaque face, servant de support pour la discrétisation et la tessellation (Section 2.2). Les

seuls points discrétisés sont ceux obtenus en parcourant la courbe f_s avec le pas paramétrique $STEP_t$. Pour éviter des cracks liés à la tessellation avec d'autres algorithmes de tessellation comme par exemple l'algorithme de Rockwood et al. [RHD89], leur méthode peut être utilisée, mais il faut veiller à discrétiser également les intersections de la courbe f_s avec les deux grilles de support $STEP_u, STEP_v$ associées (c'est-à-dire une grille correspondant à chaque surface support), une opération complexe à mettre en œuvre. Pour finir, signalons l'existence de méthodes qui « cousent » les faces les unes entre elles dans un traitement fait *a posteriori*. Ces méthodes sont lourdes, étant confrontées à des problèmes, outre de représentation commune des courbes partagées par deux faces, d'ajout et de suppression de points sur ces dernières, puisqu'elles ont été discrétisées séparément et n'ont pas le même nombre de points.

2.4 Niveaux de détails

Etant donné que la tessellation statique génère un nombre très important de sommets et de triangles, la problématique de l'occupation mémoire est posée. En effet, les données discrétisées produites consomment souvent beaucoup plus d'espace mémoire que la définition des faces B-Rep à partir desquelles elles ont été créées. Cela est d'autant plus vrai que l'erreur autorisée pendant la discrétisation est réduite. D'autre part, la grande précision des pièces mécaniques discrétisées ainsi produites n'est pas nécessaire lorsque ces dernières sont visualisées de manière distante, à très faible niveau de zoom, et cette précision superflue limite la réactivité du rendu, nuisant à l'interaction.

Pour pallier ces problèmes, les modèles peuvent être discrétisés à différents niveaux de détails (Figure 2.12), mais cela impose de changer de niveau de détail objet par objet pendant le rendu, créant un effet de *popping* désagréable, où l'on voit les objets changer brusquement de représentation. Une autre approche plus judicieuse consiste à adopter une représentation avec niveau de détail intégré, où un modèle tessellé peut avoir sa représentation visuelle dégradée pendant le rendu, à la demande et automatiquement, en fonction de sa profondeur et de son emplacement dans le champ de vue. Ce type de représentation, qui fonctionne pour n'importe quel maillage et pas simplement pour les modèles B-Rep, a été notamment introduit par Hoppe dans son travail sur les maillages dit *progressifs* [Hop96]. Des améliorations ont suivi pour que ces représentations soient efficacement mises en œuvre sur GPU [HSH09]. Concrètement, cette dernière représentation progressive jusqu'à un niveau de résolution maximum donné a un coût de stockage environ 50 à 60% supérieur à celui de la représentation discrète simple, de même résolution. Les performances dépendent du temps alloué à la migration d'un niveau de résolution à un autre pendant le rendu de chaque image, la migration se faisant par étapes. Un temps élevé permet de rapidement s'adapter à une hausse de résolution induite par un zoom rapproché par exemple, au prix d'une baisse temporaire du taux de rafraîchissement de l'affichage. Un temps réduit permet de bénéficier d'un taux de rafraîchissement constant mais de s'adapter plus lentement à une hausse de résolution.

D'autres représentations similaires ont été imaginées par la suite et sont résumées dans le rapport de Floriani et al. [DFKP05].

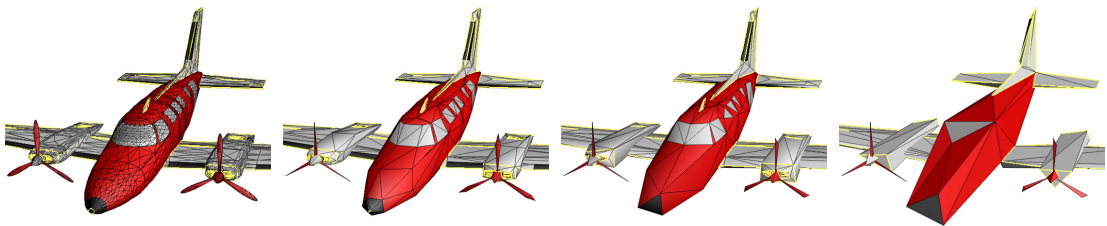


FIGURE 2.12 – Niveaux de détails d'un même maillage. Le modèle B-Rep sous-jacent est discrétisé à différents niveaux de résolution.

Chapitre 3

Rendu basé découpe

3.1 Principe général

Nous avons vu dans le chapitre précédent une méthode de rendu où chaque face B-Rep est tessellée en prenant en compte l'ensemble des données qui la définissent, c'est-à-dire la surface support et les courbes de découpe qui lui sont associées. La tessellation produite génère un grand nombre de sommets et de triangles qui suivent les courbes de découpe (Figure 2.1 du Chapitre précédent).

Le rendu basé découpe fonctionne autrement. Il effectue le rendu d'une surface support et découpe *dynamiquement* cette dernière, pendant la rasterisation. La découpe s'effectue typiquement fragment par fragment. Les fragments dont les coordonnées u, v se trouvent dans la zone de découpe sont affichés tandis que ceux hors de la zone de découpe sont ignorés. On appelle ce procédé la *classification* d'une coordonnée u, v dans l'espace de découpe. On parle de coordonnée u, v située *sur la face* (ou *dans la zone de découpe*), ou *hors de la face* (encore appelé *hors de la zone de découpe*) (Figure 3.1).

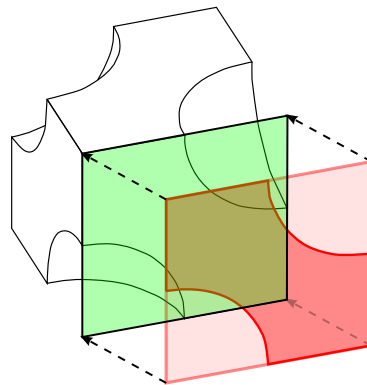


FIGURE 3.1 – Vert : surface support et espace paramétrique de définition, représentant l'espace de découpe. Rouge : partie du domaine de définition située dans la zone (rouge foncé) ou hors de la zone (rouge clair) de découpe.

Cette approche a plusieurs avantages. Nous avons vu dans le chapitre précédent que la tessellation dynamique de faces B-Rep est très complexe, et se prête mal au mode de fonctionnement du GPU, dédié aux tâches simples, parallélisables et de dimensions connues. L'approche basée découpe permet d'effectuer le rendu des surfaces support qui, contrairement à celui des faces, est beaucoup plus aisé à faire dynamiquement, notamment sur le GPU. La découpe est ensuite

prise en compte de manière individuelle pour chaque fragment, et devient donc un travail bien adapté au shader de fragment.

Plusieurs méthodes existent pour d'une part effectuer le rendu des surfaces support, et d'autre part, pour réaliser la découpe dynamiquement sur le GPU, ces deux problématiques étant séparées. Nous allons les détailler successivement dans ce chapitre.

3.2 Gestion de la découpe

La découpe dynamique, effectuée fragment par fragment, repose sur la définition d'une structure avec laquelle une classification est effectuée.

3.2.1 Représentation discrète

Guthe et al. [GBK05] sont les premiers à s'intéresser à la découpe directe sur GPU en proposant l'utilisation de texture de découpe (*trim texture*). Ils réalisent donc la découpe avec une discrétisation de l'espace de découpe. Une texture monochrome est utilisée, où chaque texel est représenté par un bit, et où les texels dans la zone de découpe ont une valeur de 1, les autres ayant une valeur de 0. En se basant sur l'affichage d'une surface support S dont la boîte englobante est connue au moment du rendu, Guthe et al. proposent un moyen simple et efficace de générer une texture de découpe en temps réel en s'appuyant sur le rasteriseur. Ils proposent tout d'abord de calculer la résolution de la texture de découpe à utiliser à partir d'une déviation maximale ϵ entre la surface support et son approximation linéaire. ϵ est déduite d'une valeur ϵ_{img} exprimée en espace écran, égale ou inférieure à un pixel. La résolution est donnée par la formule

$$res_u = \left\lceil \frac{u_1 - u_0}{\epsilon} \sup_{p \in (0,0),(1,1)} \left\| \frac{\delta S(p)}{\delta u} \right\| \right\rceil \quad (3.1)$$

res_v étant calculé de manière similaire avec v_0, v_1 ($(u_0, u_1) \times (v_0, v_1)$ étant un sous-domaine considéré pour une découpe). Les courbes de découpe doivent également être discrétisées en prenant en compte la déviation ϵ . Exprimées dans l'espace paramétrique des surfaces support, elles doivent respecter une déviation maximale ϵ_{uv} , calculée à partir de ϵ . La formule de Filip et al. [FMM87] est utilisée pour déduire de la déviation le nombre de segments à discrétiser

$$n = \left\lceil \sqrt{\frac{\sup_{t \in T} \|f''(t)\|}{8\epsilon_{uv}}} \right\rceil \quad (3.2)$$

où $f''(t)$ représente la dérivée seconde d'une courbe de découpe sur la surface. La texture peut ensuite être générée dans un framebuffer dédié de résolution $res_u \times res_v$, à la demande. Pour chaque boucle de découpe, des triangles sont générés et rasterisés, en éventail. Le premier sommet de chaque triangle est toujours le point de départ de la boucle, les deux autres sommets correspondant à deux sommets de discrétisation consécutifs le long de la boucle, en parcourant les courbes de découpe de cette boucle, dans l'ordre. La rasterisation sur le GPU est configurée de sorte que soient comptabilisés le nombre de fragments rasterisés à un emplacement donné. Une fois la rasterisation de tous les triangles effectués, un nombre impair de rasterisation effectué identifie un texel dans la zone de découpe, tandis qu'une impaire identifie un pixel hors de la zone de découpe. Le procédé de génération de la texture de découpe est illustré sur la Figure 3.2.

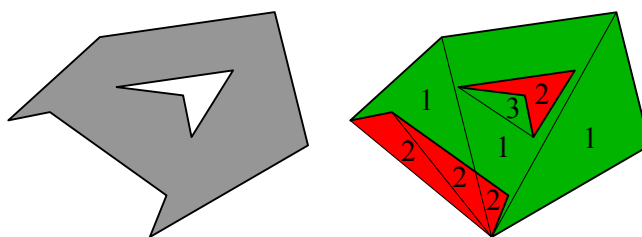


FIGURE 3.2 – Génération d'une texture de découpe selon la méthode de Guthe et al. [GBK05]. La découpe est constituée ici de deux boucles de découpe, chacune composée d'une ou plusieurs courbe. Les courbes sont discrétisées et, le long de chaque boucle, des triangles sont générés et rasterisés, en éventail. Les chiffres indiquent le nombre de recouvrement à la rasterisation.

La texture obtenue est utilisée par la suite comme texture de découpe. Pendant la rasterisation d'une surface, les coordonnées u, v associées à chaque fragment sont converties en coordonnées de texel dans la texture de découpe et la classification est effectuée.

3.2.2 Représentation vectorielle

Nishita et al. [NSK90] sont les premiers à proposer une méthode de découpe vectorielle. La classification d'un point dans le domaine de définition de la surface au travers de cette structure de découpe s'appuie sur le fait qu'un point S peut être classifié comme étant dans la zone de découpe si un rayon R partant de S intersecte un nombre impair de boucles de découpe (théorème de Jordan).

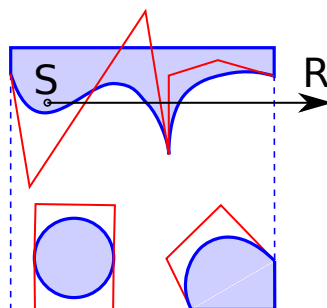


FIGURE 3.3 – Bleu : courbes de découpe, organisées en trois boucles. Rouge : polygones de contrôle de chaque courbe (ceux des courbes de degré 1 omis). La classification d'un point S dans la structure de découpe s'appuie sur le fait que tout rayon R partant de S intersecte un nombre impair de fois les boucles de découpe (ici 3 fois).

Pour calculer l'intersection de R avec chacune des courbes de découpe, leur méthode s'appuie sur un procédé dénommé *Bézier clipping*. Une courbe de Bézier C est définie par son équation paramétrique

$$C(u) = \sum_{i=0}^n C_i B_i^n(u) \quad (3.3)$$

avec des points de contrôle de coordonnées $C_i = (s_i, t_i)$. R repose sur une ligne L dont l'équation implicite normalisée est

$$as + bt + c = 0, \quad a^2 + b^2 = 1 \quad (3.4)$$

L'intersection de L et C , où $C_i = (s_i, t_i)$ sont les points de contrôle de C , peut être trouvée en substituant l'équation (3.3) dans l'équation (3.4) :

$$d(u) = \sum_{i=0}^n d_i B_i^n(u) = 0, \quad d_i = as_i + bt_i + c \quad (3.5)$$

Nous avons défini dans l'équation ci-avant une fonction d telle que $d(u) = 0$ pour toutes les valeurs de u pour lesquelles C intersecte L . d est la distance à L d'un point du plan. Définissons maintenant la courbe de Bézier suivante, dite *explicite* :

$$D(u) = (u, d(u)) = \sum_{i=0}^n D_i B_i^n(u) \quad (3.6)$$

Suivant la définition de cette courbe, les points de contrôle de D sont $D_i = (u_i, d_i)$, avec $d_i = as_i + bt_i + c$ et $u_i = i/n$, ce qui signifie qu'ils sont uniformément espacés sur l'axe u et ont une coordonnée u toujours croissante (Figure 3.4). De plus, puisque $\sum_{i=0}^n \frac{i}{n} B_i^n(u) \equiv u[(1-u) + u]^n \equiv u$, la coordonnée u de $D(u)$ est égale au paramètre u de $C(u)$.

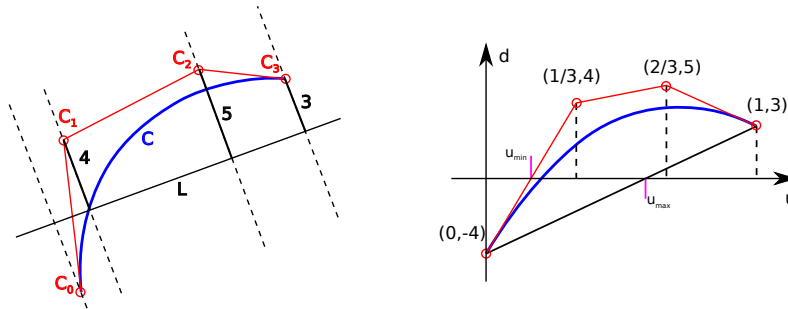


FIGURE 3.4 – Gauche : courbe de Bézier C , ligne sur laquelle repose le rayon R servant de support. Droite : représentation explicite D de C . L'axe u représente la droite L .

Sur la Figure 3.4, droite, la coquille (ou enveloppe) convexe du polygone de contrôle de D intersecte l'axe u en $u_{min} = 1/6$ et $u_{max} = 4/7$. Puisque D est contenue dans cette coquille convexe, il est donc établi que l'axe u ne peut intersecter la courbe qu'entre u_{min} et u_{max} . Il ne reste plus qu'à subdiviser notre courbe de Bézier en trois segments définis sur $0 \leq u \leq u_{min}$, $u_{min} \leq u \leq u_{max}$ et $u_{max} \leq u \leq 1$, et ne conserver que le second segment. L'algorithme de De Casteljau peut être utilisé pour la subdivision. Une intersection est trouvée lorsque la courbe en cours de traitement est suffisamment petite. La dimension maximale ϵ de sa boîte englobante peut être utilisée à cet effet. L peut avoir plusieurs intersections avec C . Si c'est le cas, le segment central extrait aura une amplitude paramétrique peu réduite par rapport à celle de la courbe originale. Pour déterminer les intersections multiples, Nishita et al. recommandent de subdiviser la courbe en deux segments si le segment central extrait après le Bézier clipping a au moins 80% de la taille du segment original, et de relancer des itérations sur ces deux sous-segments.

Pour déterminer u_{min} et u_{max} , l'axe u de la courbe explicite doit être intersecté avec la coquille convexe du polygone de contrôle. Or, l'extraction de cette coquille est coûteuse. Pabst et al. [PSS*06] proposent une méthode permettant de calculer u_{min} et u_{max} sans avoir à l'extraire.

Notons pour finir que l'algorithme du Bézier clipping peut être généralisé aux courbes rationnelles. Une courbe rationnelle est définie par

$$C(u) = \frac{\sum_{i=0}^n w_i C_i B_i^n(u)}{\sum_{i=0}^n w_i B_i^n(u)} \quad (3.7)$$

avec des points de contrôle de coordonnées $C_i = (s_i, t_i)$ et des poids associés w_i . En substituant l'équation (3.8) dans l'équation (3.4) et en éliminant le dénominateur, on obtient

$$d(u) = \sum_{i=0}^n d_i B_i^n(u) = 0, \quad d_i = w_i(as_i + bt_i + c) \quad (3.8)$$

Schollmeyer et Fröhlich [SF09] proposent une structure de découpe pouvant traiter n'importe quel type de découpe, même complexe, comme celle proposée par Nishita et al. Leur structure permet d'obtenir des performances d'exploitation considérablement accrues, notamment sur GPU. Les courbes de découpe d'entrée, des NURBS, sont tout d'abord transformées et décomposées en segments de Bézier rationnels. Leur algorithme décompose ensuite les courbes de Bézier résultantes en segments uv -monotones. Ces segments sont divisés à nouveau de sorte que chaque segment résultant de la subdivision soit classifiable dans une seule bande v horizontale de l'espace de découpe. Cette transformation est illustrée sur la Figure 3.5.

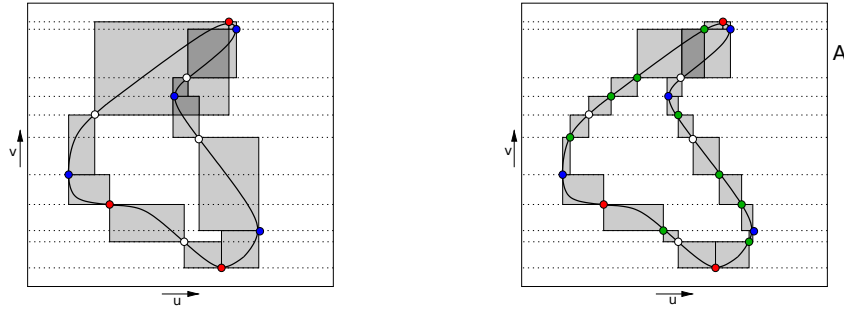


FIGURE 3.5 – *A gauche : les courbes de découpe d'entrée, des Bézier rationnelles, sont décomposées en segments uv -monotones. A droite : les segments uv -monotones sont divisés en vue d'une classification sur l'axe v , par intervalle.*

Les segments de courbes finalement obtenus sont ensuite organisés dans un double arbre binaire. Un premier arbre binaire permet d'accéder à une bande horizontale à partir d'une valeur de v donnée, dans laquelle la valeur v du point u, v à classifier se trouve. Un second arbre binaire organise le contenu d'une bande v . Il référence une suite de rectangles parcourant l'ensemble de l'espace paramétrique sur u . Sur la Figure 3.6, l'espace paramétrique est organisé en 3 zones. La zone 2, qui est une union entre les rectangles englobants de deux courbes consécutives sur u , référence deux courbes. Les zones 1 et 3 ne référencent aucune courbe.

Les zones sont parcourues dans un ordre croissant sur u , en regard de la valeur u minimale de leur rectangle englobant. Un bit d'information est maintenu pour chaque zone indiquant si son bord gauche se trouve dans ou hors de la découpe. Point de départ, la zone la plus à gauche a toujours son bord gauche hors de la découpe. Ce bit d'information indiquant la classification du bord gauche est inversé lorsqu'une zone comprenant au moins une courbe est parcourue (eg. la zone 2 sur la Figure 3.6). Cette règle pair-impair permet de classer les zones qui ne référencent aucune courbe.

Une fois la structure de découpe construite, elle peut être utilisée au moment du rendu pour effectuer la classification d'une coordonnée $P(u_p, v_p)$ de l'espace paramétrique. La classification avec les zones qui ne référencent aucune courbe est effectuée avec le seul bit d'information (zone

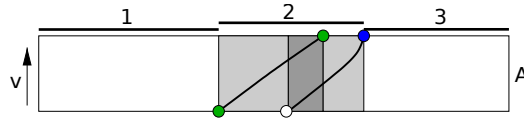


FIGURE 3.6 – Organisation de la bande v identifiée avec la lettre A sur la Figure 3.5. La zone 2 référence deux courbes.

dans ou hors de la découpe). La classification dans les zones référençant une seule courbe est réalisée avec le bit d'information, et en déterminant si (u_p, v_p) se trouve à gauche ou à droite de la courbe au moyen d'une bisection. La courbe entre dans son rectangle englobant à une valeur de paramètre $t_{entrant}$, et sort à une valeur de paramètre $t_{sortant}$. La courbe est tout d'abord évaluée à une valeur de $t_0 = \frac{t_{entrant} + t_{sortant}}{2}$. P réside à ce stade dans un des quadrants autour de $C(t)$. Si P se trouve dans un quadrant ne contenant pas la courbe ($Q1$ ou $Q4$ sur la Figure 3.7), il peut être classifié comme étant à gauche ($Q1$) ou à droite ($Q4$) de la courbe. Sinon, le processus de classification reprend sur la section de courbe $[t_{entrant}, t_0]$ ou $[t_0, t_{sortant}]$ en fonction du quadrant de résidence $Q3$ ou $Q2$ de P , respectivement.

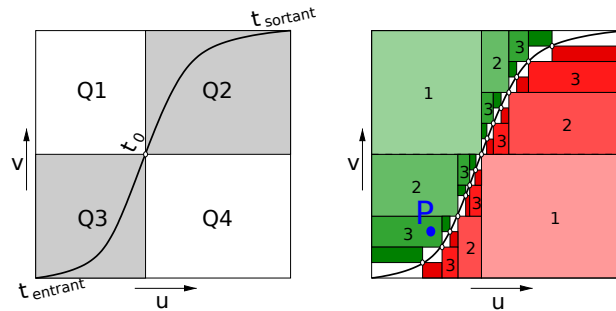


FIGURE 3.7 – Classification d'un point par rapport à une courbe. La classification du point P (en bleu) nécessite ici 3 itérations.

Le nombre moyen d'évaluations pour un ensemble de points uniformément répartis dans la boîte englobante de la courbe peut être estimé. Pour une courbe dont le paramètre t est aligné sur l'axe v de paramétrisation, comme c'est le cas sur la Figure 3.7, la première itération de la bisection permet de classer la moitié de l'espace paramétrique, la seconde permet de classer un quart, la troisième un huitième et ainsi de suite. La moitié de l'espace paramétrique peut ainsi être classifié avec 1 itération, un quart de l'espace paramétrique peut l'être avec 2 itérations et ainsi de suite. Le nombre moyen d'évaluations, où N est le nombre d'itération, est donné par la formule suivante :

$$\sum_{i=1}^N \frac{i}{2^i} < 2 \quad (3.9)$$

Hanniel et Haller [HH11] réalisent la classification de coordonnées u, v dans l'espace paramétrique en passant par les coordonnées correspondantes sur la surface support, c'est-à-dire converties en espace objet. La classification d'un point en espace objet est effectuée en se servant des surfaces adjacentes aux courbes de l'espace de découpe, et non en utilisant les courbes elles-mêmes qui ne sont jamais utilisées. Leur approche est originale puisqu'elle permet potentiellement de se débarrasser des jours géométriques (Partie I, Section 2.2.1) entre les faces.

Leur méthode consiste tout d’abord à construire une grille dans laquelle sont référencées les surfaces de découpe associées aux courbes traversant chaque cellule de la grille. La dimension de la grille est calculée de sorte que chaque case référence un maximum de 2 surfaces. Pour les découpes complexes, ils proposent d’utiliser un quadtree à la place de la grille, dont la résolution peut augmenter afin de respecter la contrainte de 2 surfaces par cellule (Figure 3.8).

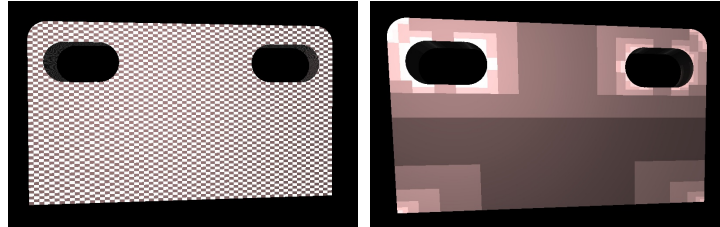


FIGURE 3.8 – Hanniel et Haller [HH11] organisent leur découpe dans une grille ou dans un quadtree. Les cellules de la grille ou du quadtree peuvent référencer jusqu’à deux surfaces de découpe, adjacentes à la surface de référence.

Une fois la structure de découpe créée, elle est utilisée au moment du rendu pour faire la classification. La cellule de la coordonnée u_p, v_p est tout d’abord extraite à partir de la grille ou du quadtree de la structure. u_p, v_p est ensuite évalué sur sa surface support. La coordonnée en espace objet x, y, z obtenue est projetée sur la ou les surfaces présentes dans la cellule. S’il n’y a qu’une seule surface, P est classifié dans la zone de découpe si le vecteur $P - S^p$ et la normale associée à S^p vont en sens opposé (produit scalaire négatif). Si deux surfaces sont présentes, le résultat de la classification pour chaque surface est combiné logiquement avec un opérateur d’union ou d’intersection, suivant les cas.

Outre un problème de performance lié à la projection très coûteuse de points sur les surfaces pour effectuer la classification, l’algorithme proposé par Hanniel et Haller présente d’importantes limitations. Ces limitations se traduisent par des artéfacts visuels montrés sur la Figure 3.9. D’une part, la découpe ne peut pas toujours être représentée localement dans une cellule par une composition logique du résultat de la classification avec deux surfaces. D’autre part, même pour les cellules n’ayant qu’une seule surface, la classification ne produit pas toujours le résultat escompté dans la mesure où la projection d’un point P sur la surface adjacente à la surface de référence (pour laquelle la découpe doit être faite) produit parfois un résultat inattendu. Ce type de problème est illustré sur la partie gauche de la Figure 3.9, plus spécifiquement au niveau de l’affichage du plan en rose sombre. Le plan relie le cône situé au dessus en passant sur un chanfrein toroïdal. Les cellules représentant une jonction avec le tore sont ici traitées intentionnellement comme des cellules pleines bien qu’elles référencent le tore, produisant l’artéfact visuel. Ceci est dû au fait que la définition du tore étant prolongée au-delà de la jonction avec le plan, la projection des points pour faire la découpe est faite sur la partie prolongée du tore, qui se trouve à l’intérieur de l’objet. Il en résulte une classification incorrecte et des artéfacts visuels. Hanniel et Haller détectent ces situations et transforment les cellules concernées en cellules pleines, ce qui produit un effet de discrétisation fort désagréable.

Les méthodes d’affichage de texte et de **rendu vectoriel au sens large** peuvent servir de source d’inspiration pour réaliser la découpe de faces B-Rep, découpe qui n’est rien d’autre qu’une forme vectorielle.

Loop et Blinn [LB05] proposent une méthode d’affichage de formes vectorielles efficace sur GPU. Elle prend en charge les formes linéaires mais également les formes quadratiques et cubiques, ce qui leur permet d’assurer un rendu fidèle de fontes TrueType et de dessins SVG par

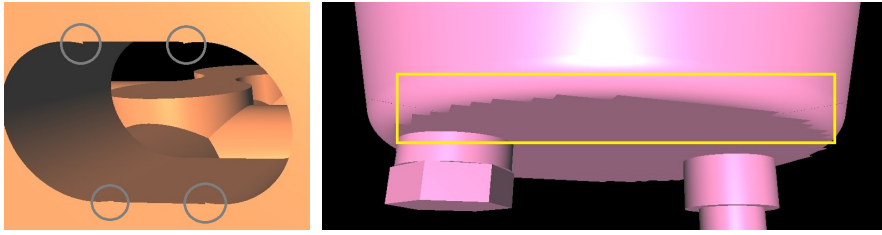


FIGURE 3.9 – Artéfacts causés par la méthode de découpe de Hanniel et Haller. Les artéfacts de gauche sont expliqués sur la Figure 3.10

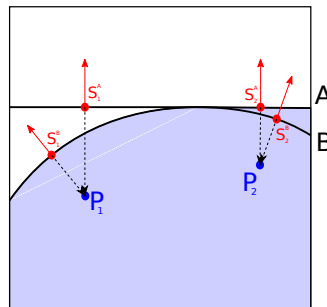


FIGURE 3.10 – Cellule de grille ou de quadtree référençant deux surfaces A (un plan) et B (un cylindre ou portion de cylindre) utilisées pour la classification. La zone bleue identifie la partie de la cellule dans la découpe, la zone blanche celle hors de la découpe. Pour que la classification de P_1 fonctionne, une union du résultat de classification avec A et B doit être réalisé, alors que pour P_2 , une intersection doit être opérée.

exemple. Nous ne présentons ici que les formes linéaires et quadratiques et invitons le lecteur à se référer à leurs travaux pour des informations relatives aux courbes cubiques, plus complexes à prendre en charge.

Une tessellation des formes à représenter est tout d’abord effectuée en utilisant une triangulation de Delaunay. La triangulation est réalisée avec les polygones de contrôle des courbes quadratiques (Figure 3.11). Les triangles hors de la forme à représenter sont éliminés (la partie enclavée de la lettre e sur la Figure 3.11).

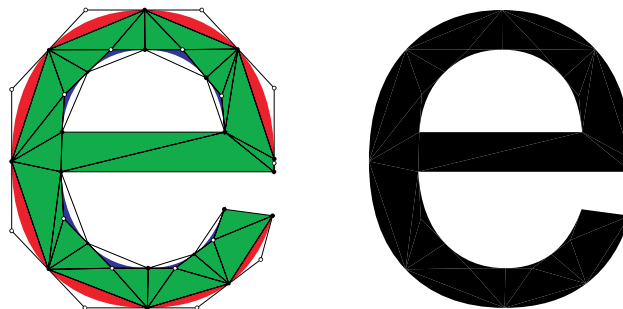


FIGURE 3.11 – Rendu d’une lettre de fonte TrueType avec la méthode de Loop et Blinn [LB05].

Le rendu des formes quadratiques dans les triangles associés est assuré par l’utilisation d’un shader de fragment approprié. A chaque fois qu’un fragment est traité, il doit être classifié afin de déterminer s’il doit être affiché ou non. Le résultat de la classification dépend du côté de la courbe

sur lequel il se trouve. Loop et Blinn proposent donc un moyen de représenter d'une part la courbe de manière efficace dans son polygone de contrôle, et un moyen d'exprimer les coordonnées du fragment en cours de traitement compatible avec cette représentation. C'est une représentation implicite qu'ils ont choisi, d'une part, et l'utilisation de coordonnées barycentriques, de l'autre.

Une courbe paramétrique est une fonction vectorielle d'une variable unique. Une courbe paramétrique rationnelle de degré n peut être exprimée sous la forme

$$C(t) = \mathbf{t} \cdot \mathbf{C} \quad (3.10)$$

où, pour $n = 2$,

$$\mathbf{t} = [1 \ t \ t^2], \quad \text{et} \quad \mathbf{C} = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} \quad (3.11)$$

Les courbes de Bézier quadratiques peuvent être représentées sous forme de combinaison linéaire de bases de puissances $\mathbf{C} = \mathbf{M}_2 \cdot [b_0 \ b_1 \ b_2]$ où b_0, b_1 et b_2 sont les points de contrôle de la courbe, et où

$$\mathbf{M}_2 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix}, \quad \text{dont l'inverse est } \mathbf{M}_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & \frac{1}{2} & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (3.12)$$

Selon Sederberg [Sed83], toute courbe paramétrique définie avec les équations

$$x(t) = \frac{x^n(t)}{w^n(t)}, \quad y(t) = \frac{y^n(t)}{w^n(t)} \quad (3.13)$$

admet une équation implicite de la forme

$$c^n(x(t), y(t)) = 0 \quad (3.14)$$

la notation $c^n(\cdot)$ désignant une fonction polynomiale quelconque de degré n . Loop et Blinn montrent dans leur travail que la projection sur le plan de toute courbe paramétrique rationnelle de degré 2 peut être représentée sous la forme implicite suivante :

$$f(u, v) = u^2 - v = 0 \quad (3.15)$$

Cette courbe peut être interprétée sous la forme $f_u(u) = u^2$, $f_v(v) = v^2$, et peut être formulée avec le produit matriciel suivant :

$$F(t) = \mathbf{t} \cdot \mathbf{F} = [t \ t^2 \ 1], \quad \text{avec} \quad \mathbf{F} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad (3.16)$$

Loop et Blinn cherchent par conséquent une transformation Ψ telle que

$$\mathbf{M}_2 = \mathbf{F} \cdot \Psi^{-1} \quad (3.17)$$

en isolant Ψ^{-1} on obtient $\Psi^{-1} = \mathbf{F}^{-1} \cdot \mathbf{M}_2$. On en déduit Ψ :

$$\Psi = \mathbf{M}_2^{-1} \cdot \mathbf{F} = \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (3.18)$$

La courbe de Bézier quadratique $C(t)$ avec les points de contrôle

$$\left[\begin{array}{c} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} \frac{1}{2} \\ 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \end{array} \right] \quad (3.19)$$

a donc la même empreinte dans le plan que le 0-set de la fonction implicite $f(u, v) = u^2 - v = 0$. Pour généraliser la prise en charge de courbes quadratiques quels que soient les emplacements des points de contrôle \mathbf{b}_0 , \mathbf{b}_1 et \mathbf{b}_2 , Loop et Blinn affectent les coordonnées $(0, 0)$, $(\frac{1}{2}, 0)$ et $(1, 1)$ aux points de contrôles sous forme d'attributs additionnels avant de les envoyer au shader de sommet (Partie 3, Chapitre 3.1). Ce dernier interpole ensuite ces coordonnées, ramenant ainsi automatiquement les coordonnées du fragment à celles en vigueur dans l'enceinte du polygone de contrôle défini en (3.19) (Figure 3.12).

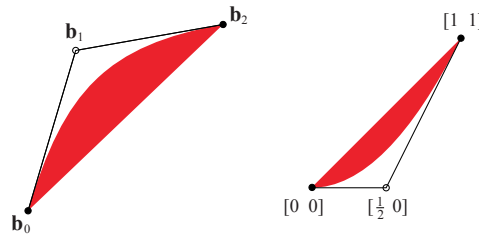


FIGURE 3.12 – L'utilisation de coordonnées additionnelles passées au vertex shader au niveau des points de contrôle permet de faire le rendu de n'importe quelle forme basée sur une courbe quadratique, en ramenant les points de contrôle dans le système de coordonnées de référence (à droite) avant de faire la classification.

Nehab et Hoppe [NH08] s'intéressent également au rendu vectoriel à base de courbes linéaires et quadratiques. Ils partitionnent leur espace de découpe en $n \times m$ cellules, en utilisant une grille avec une taille de cellule uniforme. La représentation des formes dans chaque cellule est basée chemin, où les formes à remplir au moment du rendu sont délimitées par des primitives linéaires (lignes ou courbes quadratiques), représentant un espace fermé (des courbes linéaires additionnelles sont créées pour fermer les cellules, le cas échéant). Leur travail se focalise également sur d'autres points importants, inapplicables à la découpe mais pertinents pour du rendu de forme vectorielle (tracé de courbe avec une épaisseur spécifique, anti-aliasing etc.).

Comme Nishita et al. [NSK90], ils utilisent la règle pair-impair pour effectuer la classification d'un point dans une cellule. Un rayon horizontal partant vers la droite est tracé à partir de la coordonnée u_p, v_p à classifier, et le point est classifié comme étant dans la zone de découpe si le nombre d'intersection avec les primitives linéaires de la cellule est impair. Nous omettons ici le développement pour les primitives linéaires. Pour les quadratiques, une courbe avec les points de contrôle (b_{i-1}, b_i, b_{i+1}) est définie sur l'intervalle $0 \leq t \leq 1$ avec l'équation

$$b(t) = (1 - t)^2 b_{i-1} + 2(1 - t)t b_i + t^2 b_{i+1} \quad (3.20)$$

Les racines t_1 et t_2 sont déterminées pour $b_v(t) = v_p$. Les cellules contenant un nombre variable et potentiellement illimité de primitives et d'ordres de tracé, les informations doivent être rangées intelligemment en mémoire pour être efficacement exploitées par le GPU par la suite (Figure 3.13). Nehab et Hoppe proposent pour gérer cette organisation aussi bien une table d'indirection qu'une méthode de hashage spatial optimal [NH07], basée sur le travail de Lefebvre et Hoppe [LH06]. Avec cette dernière méthode, une table de présence indique si une cellule de

la grille contient effectivement des données. Si c'est le cas, une table de hashage H et une table d'offset Φ sont utilisées pour accéder à un emplacement mémoire où sont rangées les primitives contenues dans la cellule, de longueur variable. Seuls trois accès mémoire au total sont nécessaires pour accéder aux informations d'une cellule, quelle que soit le modèle d'entrée et la taille $n \times m$ de la grille utilisée.



FIGURE 3.13 – A gauche : rendu basé chemin restreint à chaque cellule pour Nehab et Hoppe [NH08], effectué avec un nombre variable de primitives par cellule. A droite : quantité d'informations présentes dans chaque cellule (plus foncé signifie un nombre d'informations accru).

Ray et al. [RCL05] proposent enfin une représentation vectorielle baptisée *Vector Texture Map*. L'espace de découpe est organisé dans un arbre de subdivision adaptatif, où chaque cellule est subdivisée si les données vectorielles associées ne peuvent être représentées suivant une définition précise. Cette définition contient une ou deux courbes cubiques représentant une reformulation des contours de la forme vectorielle, restreints à la cellule. Si la reformulation ne peut être définie, la cellule est subdivisée. Elle repose sur une courbe de Hermite :

$$t' = \tan(\alpha_A)s'^2(1 - s') - \tan(\alpha_B)s'(1 - s')^2 \quad (3.21)$$

dont la forme implicite est

$$f(s, t) = t' - (\tan(\alpha_A)s'^2(1 - s') - \tan(\alpha_B)s'(1 - s')^2) \quad (3.22)$$

où α_A , α_B , $s'(s, t)$ et $t'(s, t)$ sont définis sur la Figure 3.14.

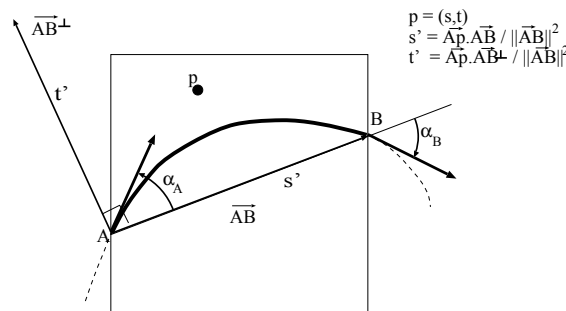


FIGURE 3.14 – Reconstruction implicite d'un contour dans une cellule. A et B sont ici les points d'entrée et de sortie de la cellule. α_A et α_B représentent les tangentes aux points A et B . La courbe est reconstruite par rapport à son repère local \vec{AB} , \vec{AB}^\perp .

A chaque courbe approximante est associée une information booléenne déterminant si la zone à l'intérieur de l'espace de découpe se trouve au dessus ou en dessous de la courbe par rapport à l'axe s' de son repère de définition. Lorsque deux courbes sont référencées dans une cellule, la

découpe est représentée par une combinaison logique du résultat de classification par rapport à chaque courbe (Figure 3.15). Pour pouvoir prendre en charge une minification au moment du rendu, Ray et al. construisent une pyramide de mipmap d’une résolution correspondant à 2^n où n représente la profondeur maximale de subdivision. Chaque mipmap est une texture 8bpp permettant de représenter 256 niveaux de couverture de la cellule (par exemple 0 signifiant que la cellule est hors de la zone de découpe, et 255 qu’elle est totalement dedans). Le consommation mémoire est non négligeable.

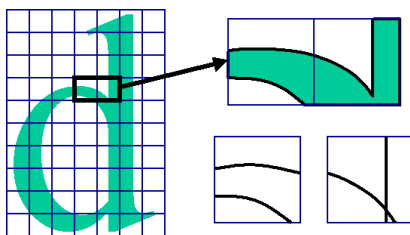


FIGURE 3.15 – La classification d’un point dans les cellules recevant deux courbes implicites est effectué en combinant logiquement le résultat de classification du point par rapport à chaque courbe, et en utilisant le booléen propre à chaque courbe indiquant si l’espace de découpe est situé au dessus ou en dessous de la courbe.

3.3 Rendu des surfaces support

Le rendu basé découpe, outre la découpe, doit assurer le rendu des surfaces support. Nous le développons dans cette section.

3.3.1 Tessellation uniforme

Nous avons vu dans le Chapitre 1 qu’un certain nombre de méthodes [FMM87, ZS00, LP99] existent pour calculer, à partir d’une déviation ϵ exprimée en espace objet ou ϵ_{img} exprimée en espace écran, des pas paramétriques de discrétisation $STEP_u$ et $STEP_v$ pouvant être utilisés pour échantillonner des points uniformément sur les surfaces. Un maillage reliant ces points entre eux, représentant une approximation linéaire de la surface, est appelé une *tessellation uniforme*.

La tessellation uniforme a deux vertus. D’une part, elle est efficacement mise en œuvre en tirant parti des shaders de tessellation, disponibles sur les GPU depuis 2009, et dont le mode de fonctionnement a été expliqué dans la Partie I, Chapitre 3.1. En s’appuyant sur ces shaders, Yeo et al. [YBP12] effectuent la tessellation uniforme de patches de Bézier de degrés arbitraires non découpés (Figure 3.16). Nous avons détaillé leur méthode dans le chapitre précédent. Régulant la déviation écran à 1 pixel, la qualité de leur rendu s’approche de celle obtenue avec le lancer de rayon, mais avec des performances largement supérieures.

L’autre vertu de la tessellation uniforme vient directement de l’échantillonnage uniforme réalisé sur la surface, qui se prête très bien à une méthodologie de discrétisation dénommée *forward differencing* [Wal90]. Cette méthode de calcul repose sur la détermination initiale d’une discrétisation de la surface en $u = 0, v = 0$ et sur une méthode permettant de rapidement mettre à jour l’emplacement du point discrétisé, incrémentalement, en fonction de $STEP_u$ et $STEP_v$. Le désavantage étant que les points discrétisés doivent être calculés séquentiellement, et non en parallèle. Schwartz et Stamminger [SS09], ne disposant pas de GPU équipé d’une unité de

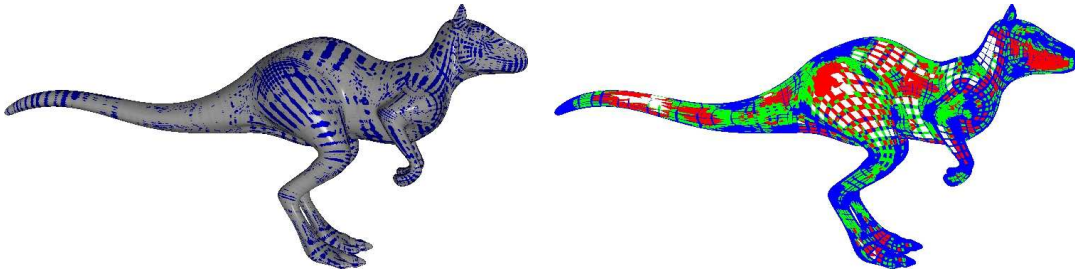


FIGURE 3.16 – Modèle du killeroo rendu avec la méthode de Yeo et al. A gauche : précision paramétrique de chaque pixel, représentée avec des couleurs. En gris, inférieure à 0.1 pixel. En bleu, inférieure à 0.5 pixel. A droite : taille moyenne des triangles produits par la tessellation, en nombre de pixels. Bleu < 5 , vert < 10 , rouge < 20 , blanc ≥ 20 .

tessellation, choisissent cette approche pour faire le rendu de patches de Bézier bicubiques non découpés (Figure 3.17). Les performances qu'ils obtiennent avec un traitement générique parallèle (GPGPU) sont là encore excellentes.

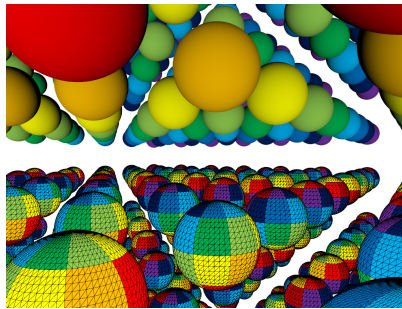


FIGURE 3.17 – Tessellation produite avec la méthode de Schwarz et Stamminger, reposant sur une implémentation GPGPU et utilisant la méthode du forward differencing.

3.3.2 Lancer de rayon

Le rendu par lancer de rayon se propose de générer pour chaque pixel de l'image un rayon partant du point de vue, c'est-à-dire de la position de la caméra, et passant par l'emplacement correspondant au pixel sur le plan *proche* de la pyramide de vue (on parle souvent de *near plane*), comme montré sur la Figure 3.18.

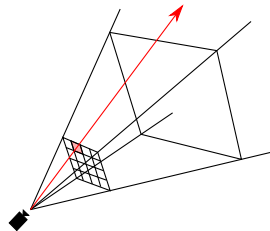


FIGURE 3.18 – Principe du rendu par lancer de rayon. Le pixel dont la couleur doit être déterminée sert de support à un rayon, passant en son centre et partant du point de vue. Ce rayon est intersecté avec la scène.

Une fois qu'un tel rayon est généré, une intersection avec le modèle B-Rep est réalisée. Pour chaque face du modèle, on teste la ou les intersections potentielles avec le rayon. Si aucune intersection n'est trouvée, le pixel prend la couleur du fond d'écran (Figure 3.19, rayon *c*; le fond d'écran est bleu). Pour chaque intersection avec une surface correspondant à une face B-Rep trouvée, la coordonnée paramétrique u, v où siège l'intersection est déterminée. Si cette coordonnée paramétrique se trouve en dehors du domaine de définition de la surface, l'intersection est ignorée. Sinon, la découpe (Section 3.2) permet de déterminer si la coordonnée u, v se trouve dans la zone de découpe de la face. Si ce n'est pas le cas (Figure 3.19, rayon *b*), l'intersection est là encore ignorée et les tentatives d'intersections prennent place avec d'autres surfaces. Si la coordonnée u, v est dans la zone de découpe (Figure 3.19, rayon *a*), les opérations liées au rendu prennent place. Ces opérations nécessitent l'obtention d'une normale, déterminée en évaluant la surface avec la coordonnée u, v extraite avant la découpe.

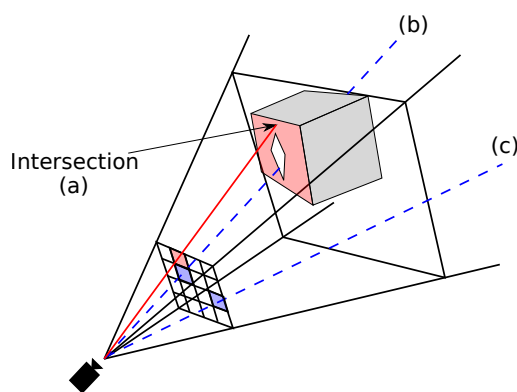


FIGURE 3.19 – Lancé de trois rayons, *a*, *b* et *c*.

Les surfaces des modèles B-Rep sont toutes définies paramétriquement. Il est possible de calculer analytiquement les intersections avec certains types de surfaces paramétriques simples en passant par une représentation implicite de ces surfaces. L'intersection entre le rayon et la primitive implicite est alors calculée en utilisant les équations définissant cette primitive. Si une intersection est trouvée, on détermine les coordonnées paramétriques du point trouvé dans le domaine de définition de la surface B-Rep associée. La dualité de définition des surfaces support, à la fois implicitement et paramétriquement définies, permet à ce titre de bénéficier de la simplicité du modèle implicite pour réaliser des intersections, et de la capacité de la représentation paramétrique permettant de réaliser la découpe.

3.3.2.1 Méthode analytique

Toledo et Lévy [TL08] font du rendu de primitives géométriques simples définies sous forme implicite (cônes, cylindres, tores etc.). Ils effectuent un lancer de rayon pour chaque pixel contenu dans une zone polygonale de l'écran correspondant à une approximation de l'empreinte de la primitive à afficher (Figure 3.20).

Leur méthode présente des performances très bonnes, tant au niveau de la rapidité de rendu que de la consommation mémoire (Figure 3.21), attestant que l'utilisation de représentation implicite pour les primitives simples peut être grandement utile pour le rendu. Toutefois, seules des primitives géométriquement simples peuvent être prises en charge de cette manière.

D'une manière générale, les méthodes analytiques sont complexes à mettre en œuvre. Kajiyama [Kaj82] calcule les intersections entre un rayon et un patch de Bézier bicubique avec une

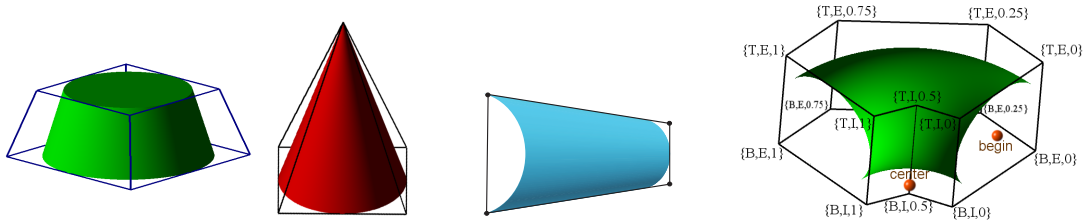


FIGURE 3.20 – Volumes englobants à l’intérieur desquels le lancer de rayon est effectué. Seules les faces faisant face à la caméra sont passées au rasteriseur. Pour le cylindre, troisième figure en partant de la gauche, une enceinte 2D épousant étroitement le contour de la primitive à afficher est utilisée.

méthode analytique, en trouvant les racines d’un polynôme de degré 18. Leur procédé semble difficilement généralisable à des surfaces plus complexes, hélas fréquentes dans le monde de la CAO.

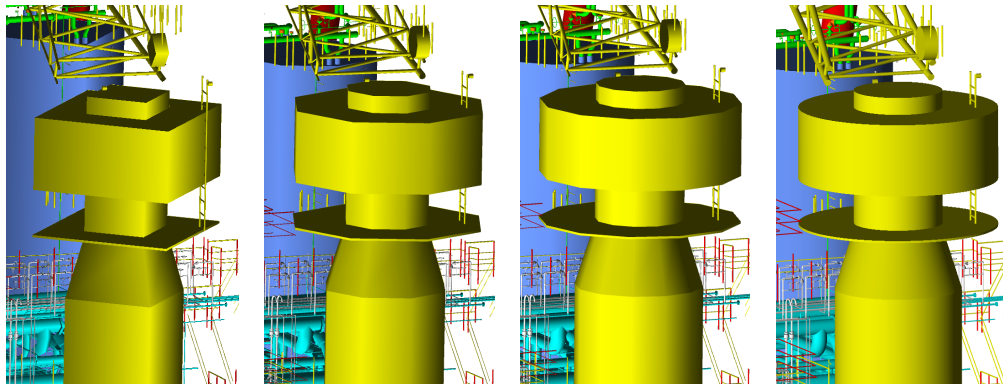


FIGURE 3.21 – Modèle d’usine rendu avec une discrétisation des primitives (3 images de gauche, où chaque primitive est tessellée avec 4, 8 ou 12 sommets support) et la méthode de Toledo et Lévy (droite), tirant parti de la définition implicite. Le taux de rafraîchissement de l’affichage pour ces images est respectivement de 40, 21, 13 et 22 fps. L’utilisation mémoire est respectivement de 283, 464, 740 et 89 Mo.

3.3.2.2 Subdivision uniforme

Le lancer de rayon peut se faire directement sur des surfaces définies paramétriquement. Lorsqu’une intersection ne peut être trouvée par voie analytique, il est possible de subdiviser un patch jusqu’à ce que la section de patch considérée soit suffisamment petite pour que le centre de ce patch puisse être considéré comme étant l’intersection entre le rayon et le patch de départ. Reste à déterminer comment subdiviser la surface.

La subdivision uniforme jusqu’au pixel est la méthode la plus simple, choisie par Catmull [Cat74]. Le rayon est représenté par une intersection de deux plans orthogonaux. La surface paramétrique est récursivement subdivisée, typiquement en quatre sections uniformes. Pour chaque sous section, on teste la position des points de contrôle par rapport aux deux plans constituant le rayon. On élimine les sous patches dont les points de contrôle se trouvent tous d’un seul côté d’au moins un des deux plans, indiquant que le rayon n’intersecte pas le sous patch en

question. On ne conserve ainsi que les sous-patches intersectant le rayon. Ce processus continue récursivement jusqu'à ce que la taille de l'empreinte d'un patch à l'écran soit d'un seul pixel. La taille de l'empreinte est mesurée en prenant pour référence le rectangle englobant de la coquille convexe du patch. L'algorithme de De Casteljau est utilisé pour subdiviser ces derniers.

L'inconvénient de cette méthode est qu'un nombre très important de subdivision doit être effectué avant de trouver une intersection. L'approche de subdivision est récursive et se prête donc mal à une implémentation sur GPU. Enfin, pour les patches de Bézier, la décomposition avec l'algorithme de De Casteljau impose l'utilisation d'un nombre de variables temporaires égales au degré minimum du patch, ce qui peut poser des problèmes de parallélisme (Partie I, Section 3.2.4).

3.3.2.3 Bézier clipping

La méthode du Bézier clipping, mentionnée dans le chapitre précédent sur la découpe des faces, permet de calculer les intersections non seulement entre une droite et une courbe dans le plan, mais également entre un rayon et un patch de Bézier rationnel dans l'espace 3D.

La méthode est expliquée par Nishita et al. [NSK90]. Un patch de Bézier rationnel est défini par l'équation paramétrique suivante :

$$\hat{P}(s, t) = \frac{\sum_{i=0}^n \sum_{j=0}^m B_i^n(s) B_j^m(t) w_{ij} \hat{P}_{ij}}{\sum_{i=0}^n \sum_{j=0}^m B_i^n(s) B_j^m(t) w_{ij}} \quad (3.23)$$

où $\hat{P}_{ij} = (\hat{x}_{ij}, \hat{y}_{ij}, \hat{z}_{ij})$ sont les points de contrôle avec les poids associés w_{ij} . Selon Kajiya [Kaj82], un rayon peut être défini avec l'intersection de deux plans. Considérons un rayon défini avec l'équation paramétrique

$$r(t) = o + \hat{d} \cdot t \quad (3.24)$$

Ce rayon peut être représenté avec l'équation de deux plans P_k , $k = 1, 2$. Ces plans sont définis avec des normales N_k et un scalaire associé d_k ($P_k = (N_k, d_k)$). N_1 peut être défini de la manière suivante :

$$N_1 = \begin{cases} (\hat{d}_y, -\hat{d}_x, 0) & \text{si } |\hat{d}_x| > |\hat{d}_y| \text{ et } |\hat{d}_x| > |\hat{d}_z| \\ (0, \hat{d}_z, -\hat{d}_y) & \text{dans les autres cas} \end{cases} \quad (3.25)$$

On peut ensuite poser $N_2 = N_1 \times \hat{d}$. Les deux plans, orthogonaux, contenant l'origine de notre rayon o , on a

$$\begin{aligned} d_1 &= -N_1 \cdot o \\ d_2 &= -N_2 \cdot o \end{aligned}$$

Représentons maintenant nos plans par leur équation implicite normalisée

$$a_k \hat{x} + b_k \hat{y} + c_k \hat{z} + e_k = 0, \quad k = 1, 2 \quad a_k^2 + b_k^2 + c_k^2 = 1 \quad (3.26)$$

L'intersection d'un plan k et du patch \hat{P} peut être représentée en substituant l'équation (3.23) dans l'équation (3.26) et en éliminant le dénominateur :

$$d^k(s, t) = \sum_{i=0}^n \sum_{j=0}^m B_i^n(s) B_j^m(t) d_{ij}^k = 0 \quad (3.27)$$

où

$$d_{ij}^k = w_{ij} \times DISTANCE(\hat{P}_{ij}, \text{plan } k) \quad (3.28)$$

avec

$$DISTANCE(\hat{P}_{ij}, \text{plan } k) = (a_k \hat{x}_{ij} + b_k \hat{y}_{ij} + c_k \hat{z}_{ij} + e_k) \quad (3.29)$$

$\hat{P}(s, t)$ se trouve alors sur le plan k si $d^k(s, t) = 0$. Nous allons maintenant nous servir des distances d_{ij}^k de l'équation (3.27) par rapport aux deux plans k_1 et k_2 pour définir de nouveaux points de contrôle :

$$P_{ij} = (x_{ij}, y_{ij}) = (d_{ij}^1, d_{ij}^2) \quad (3.30)$$

ainsi qu'un nouveau patch de Bézier

$$P(s, t) = \sum_{i=0}^n \sum_{j=0}^m B_i^n(s) B_j^m(t) P_{ij} \quad (3.31)$$

Notons que P est toujours non rationnel que \hat{P} le soit ou non. Dans l'équation (3.31), le plan k_1 devient l'axe y , le plan k_2 devient l'axe x et notre rayon se projette sur l'origine du système de coordonnées $(0, 0)$ (Figure 3.22).

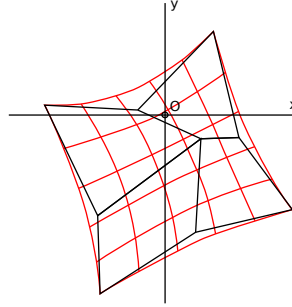


FIGURE 3.22 – Patch explicite utilisé pour la méthode du Bézier clipping.

L'intersection entre \hat{P} et notre rayon peut être déterminée en trouvant les valeurs (s, t) avec $0 \leq s, t \leq 1$ pour lesquelles $P(s, t) = 0$. Nous allons maintenant détailler la méthode pour calculer les valeurs de (s, t) . Une ligne L_s passant par $(0, 0)$ et parallèle au vecteur $V_0 + V_1$ (Figure 3.23) est définie avec son équation implicite normalisée

$$ax + by + c = 0, \quad a^2 + b^2 = 1 \quad (3.32)$$

A noter que, comme le remarque Campagna et al. [CSS97], L_s doit séparer les deux courbes situées au bord du patch dans la direction de t (celles longeant les vecteurs V_0 et V_1 sur la Figure 3.23), sans quoi l'algorithme peut identifier de fausses intersections. L'intersection de L_s et P peut ensuite être trouvée en substituant l'équation (3.31) dans l'équation (3.32) :

$$d(s, t) = \sum_{i=0}^n \sum_{j=0}^m B_i^n(s) B_j^m(t) d_{ij}, \quad d_{ij} = ax_{ij} + by_{ij} + c \quad (3.33)$$

On a $d(s, t) = 0$ pour toutes les valeurs de (s, t) où P intersecte L_s . d_{ij} dénote la distance à la ligne L_s (Figure 3.24).

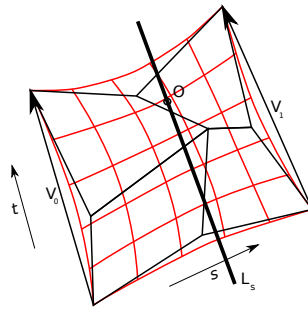


FIGURE 3.23 – Ligne L_s utilisée comme référentiel de distance.

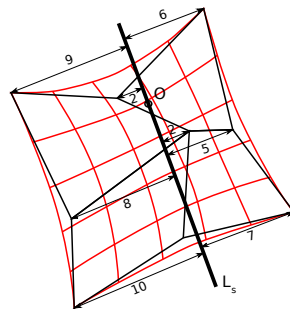


FIGURE 3.24 – Distance d_{ij} des points de contrôle D_{ij} à la ligne L_s .

La fonction $d(s, t)$ peut se représenter en base de Bernstein sous forme de surface de Bézier dite *explicite* de la manière suivante :

$$D(s, t) = (s, t, d(s, t)) = \sum_{i=0}^n \sum_{j=0}^m B_i^n(s) B_j^m(t) D_{ij} \quad (3.34)$$

Les points de contrôle $D_{ij} = (s_{ij}, t_{ij}, d_{ij})$ sont uniformément répartis sur s et t ($s_{ij} = \frac{i}{n}, t_{ij} = \frac{j}{m}$). Si nous traçons le patch D dans un repère où s représente l'abscisse et $d(s, t)$ représente l'ordonnée, nous pouvons voir que l'empreinte de D tient dans la coquille convexe de son polygone de contrôle (Figure 3.25).

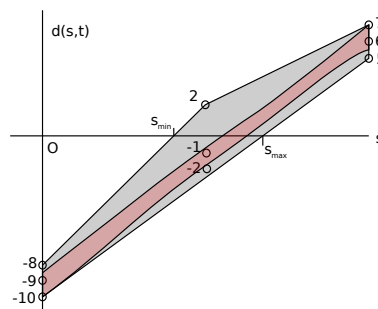


FIGURE 3.25 – Visualisation de $d(s, t)$ par rapport à s . L'empreinte du patch P apparaît en rouge.

$d(s, t)$ (et du même coup $P(s, t)$) ne peut potentiellement s'annuler ($= (0, 0)$) que pour les valeurs de s situées dans le polygone de contrôle, c'est-à-dire pour $s_{min} \leq s \leq s_{max}$. Il ne reste plus qu'à extraire la région de \hat{P} située entre s_{min} et s_{max} avec l'algorithme de De Casteljau

(Figure 3.26, gauche). Comme pour les courbes, si le segment paramétrique s_{min}, s_{max} couvre 80% du patch original — suggérant que des intersections multiples existent — le patch est subdivisé en deux sous-patches et l’algorithme reprend sur les deux sous-patches. Une fois l’itération sur s terminée, le processus de *Bézier clipping* recommence sur l’axe t . L’algorithme s’arrête lorsque la taille maximale des boîtes englobantes des patches projetés à l’écran n’excède pas une tolérance donnée, correspondant à une précision d’affichage d’un pixel (Figure 3.26, droite).

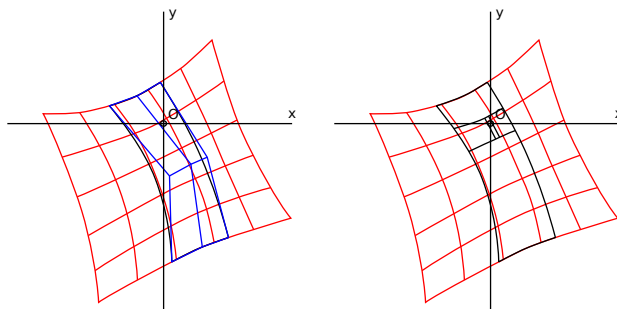


FIGURE 3.26 – A gauche : extraction de la région du patch située entre s_{min} et s_{max} avec l’algorithme de De Casteljaou. En noir, le patch obtenu et en bleu, son polygone de contrôle. A droite : itérations du Bézier clipping successives, à tour de rôle sur s et t . L’algorithme converge lorsque le sous-patch obtenu a une taille déterminée.

Au final, la méthode du Bézier clipping est efficace et permet de détecter des intersections multiples entre un rayon et le patch. Outre Campagna et al. [CSS97], des améliorations ont également été apportées par Efremov et al. [EHS05] pour la rendre plus robuste. Plus de détails sur le Bézier clipping sont disponibles dans les thèses d’Alexander Efremov [Efr04] et Joakim Löw [LÖ6]. Reposant sur la subdivision de De Casteljaou, la méthode du Bézier clipping, bien que relativement efficace à implémenter aussi bien sur CPU que GPU, se heurte aux contraintes associées à la subdivision des polygones de contrôle, lourde, et nécessitant des variables temporaires (Partie I, Section 3.2).

3.3.2.4 Méthode de Newton

Le calcul d’intersection entre un rayon et une surface peut être vu comme un problème numérique où les racines du système suivant doivent être trouvées (nous reprenons ici la définition d’un rayon de Kajiya présentée plus haut, où le rayon est l’intersection de deux plans $P_k = (N_k, d_k)$, $k = 1, 2$) :

$$F(u, v) = \begin{pmatrix} N_1 \cdot S(u, v) + d_1 \\ N_2 \cdot S(u, v) + d_2 \end{pmatrix} \quad (3.35)$$

Plusieurs méthodes existent pour approximer ces racines. Celle de Newton [FP79] est appropriée dans le cas du rendu graphique pour plusieurs raisons. Tout d’abord, les algorithmes d’éclairage nécessitent le calcul des normales des surfaces et ces normales sont généralement obtenues en calculant le produit vectoriel des vecteurs représentant les dérivées partielles sur ces surfaces, en un point donné. La méthode de Newton requiert les dérivées partielles. Ensuite, elle converge à un rythme quadratique vers une racine. Elle marche également pour tout type de surface paramétrique, pas seulement les patches de Bézier.

Une itération de Newton débute avec une coordonnée paramétrique initiale u_i, v_i sur la surface. Nous reviendrons sur cette coordonnée initiale plus tard. L’itération est définie comme

suit

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - J^{-1}(u_n, v_n) \times F(u_n, v_n) \quad (3.36)$$

où J est la matrice Jacobienne de F définie comme suit

$$J = \begin{pmatrix} N_1 \cdot S_u(u, v) = J_{11} & N_1 \cdot S_v(u, v) = J_{12} \\ N_2 \cdot S_u(u, v) = J_{21} & N_2 \cdot S_v(u, v) = J_{22} \end{pmatrix} \quad (3.37)$$

L'inverse de la matrice Jacobienne peut être calculée de la manière suivante

$$J^{-1} = \text{adj}(J) \cdot \frac{1}{\det(J)} = \begin{pmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{pmatrix} \cdot \frac{1}{J_{11}J_{22} - J_{12}J_{21}} \quad (3.38)$$

L'itération prend fin si l'une des conditions suivantes est rencontrée. Tout d'abord, nous pouvons avoir convergé d'une manière suffisamment proche de la racine, ce qui se vérifie par

$$\|F(u_n, v_n)\| < \epsilon \quad (3.39)$$

avec une valeur ϵ suffisamment petite pour permettre au processus de converger sur des pixels voisins mais non identiques localement, même à fort niveau de zoom sur les surfaces. Si l'itération ne permet pas d'extraire une racine, une nouvelle itération prend place. Le processus de convergence est décrété comme ayant échoué si une itération nous éloigne de la racine, c'est-à-dire si

$$\|F(u_{n+1}, v_{n+1})\| > \|F(u_n, v_n)\| \quad (3.40)$$

ou si un nombre d'itérations maximum est atteint. La méthode de Newton convergeant à un rythme quadratique, ce nombre peut être limité à 4 ou 5. A noter que si la matrice J est singulière, l'itération peut rentrer dans une boucle autour de la racine, sans jamais s'en rapprocher. Pour déterminer la singularité, le test $|\det(J)| < \epsilon_{\text{singularité}}$ peut être effectué. Si le test de singularité est positif, la valeur de u_n, v_n peut être légèrement perturbée pour éviter l'effet de *ping-pong* autour de la racine. Les itérations peuvent également être abandonnées si u_n, v_n nous fait sortir du domaine de définition de la surface, comme le note Martin et al. [MCFS00], mais ce critère d'arrêt ne vaut surtout que pour les surfaces mathématiquement non définies hors de leur domaine de définition, ce qui n'est pas le cas des surfaces de type NURBS ou les produits tensoriels de Bézier. Dans leur cas, il est plus judicieux de permettre une itération isolée hors du domaine [BS93] ou bien de laisser le processus de convergence arriver à son terme et, le cas échéant, déterminer si la racine trouvée est réellement dans le domaine de définition de la surface.

La méthode de Newton souffre d'un défaut, où une racine ne peut être trouvée avec elle que si la coordonnée initiale u_i, v_i est suffisamment proche de cette racine. De même, elle ne permet de trouver qu'une seule racine, quand bien même $F(u, v)$ peut en admettre plusieurs. Les conséquences en terme de rendu se traduisent par la présence d'artéfacts plus ou moins gênants pour l'utilisateur. Pour éviter ces problèmes il faut choisir, pour un rayon donné, une coordonnée de départ garantissant que la convergence puisse se produire si une racine existe effectivement, et d'autre part, qu'une et seule racine n'existe.

Pour ce faire, Toth [Tot85] propose une méthode basée sur l'arithmétique d'intervalle et la subdivision du domaine de définition de la surface considéré pour l'intersection. Ce dernier est subdivisé jusqu'à ce que les conditions précédemment citées soient réunies. Toth cherche à déterminer si, pour une surface définie sur un domaine U, V , la méthode de Newton peut

converger vers une racine unique quel que soit la coordonnée u, v de départ prise sur U, V . Sa méthode se base sur un test prenant en entrée la surface, le domaine de définition considéré et le rayon utilisé pour le lancer. Giger [Gig89] propose une approche comparable. Bien qu'efficaces et surtout fiables, ces méthodes sont assez coûteuses et ne sont qu'exceptionnellement utilisées dans la littérature.

D'autres méthodes utilisent des volumes englobants les surfaces, à partir desquels le lancer de rayon est effectué. Aucune garantie n'existe pour une convergence fiable à 100% mais, en pratique, ces méthodes donnent de bons résultats. C'est par exemple le cas avec Barth et Stuürzlinger [BS93] qui utilisent des parallélogrammes. Les surfaces sont subdivisées jusqu'à ce qu'elles puissent être approximées par un parallélogramme suffisamment plat. Une coordonnée u, v de départ est ensuite déterminée à partir de l'intersection entre le rayon et le parallélogramme, en rapport avec le domaine de définition de la surface dans le parallélogramme. Martin et al. [MCFS00] utilisent une méthode similaire mais avec des boîtes englobantes. Yen et al. [YSSP91] travaillent avec des dalles orientées. Pabst et al. [PSS*06] s'appuient sur la coquille convexe du polygone de contrôle d'un patch, en s'assurant que ce dernier soit suffisamment plat, et en subdivisant les patches lorsque ce n'est pas le cas. Pour tous ces algorithmes dont le fonctionnement est illustré sur la Figure 3.27, les surfaces sont subdivisées dans un prétraitement, par opposition à la méthode de Toth, beaucoup plus lourde, où la subdivision dépend du rayon et doit donc être faite en temps réel. Il est possible d'associer cette dernière méthode à une présubdivision sur un critère de platitude des patches, mais les performances restent faibles du fait de la subdivision dynamique pour des raisons déjà mentionnées (lourdeur d'implémentation de De Castelneau sur GPU).

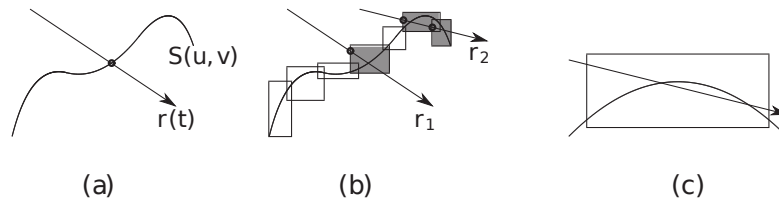


FIGURE 3.27 – (a) intersection entre un rayon et une surface. (b) en utilisant des volumes englobant des parties de la surface, des coordonnées u, v initiales peuvent être trouvées pour l'itération de Newton. (c) intersections multiples à l'intérieur d'un volume englobant.

Etant donné que la méthode de Newton n'est fiable que si des méthodes lourdes basées sur l'arithmétique d'intervalle sont utilisées [Tot85, Gig89], certains travaux utilisent d'autres approches pour allier fiabilité et rapidité. Wang et al. [WSC01] utilisent à la fois la méthode du Bézier clipping et sur celle de Newton, en s'appuyant sur la cohérence des rayons pendant le lancé, d'un pixel à l'autre. La méthode du Bézier clipping est utilisée à chaque fois que celle de Newton ne converge pas. Le fait que leur méthode s'appuie sur la cohérence des rayons rend difficile son implémentation sur GPU où les rayons doivent typiquement être traités indépendamment les uns des autres, en parallèle.

3.3.3 Tessellation adaptative

Lane et al. [LCWB80] proposent de subdiviser paramétriquement une surface jusqu'à ce qu'une section de surface considérée ait une certaine platitude, auquel cas elle peut être rendue avec deux triangles représentant l'approximation bilinéaire de la surface. La platitude est estimée en évaluant la déviation entre les courbes délimitantes aux extrémités du patch et le segment

de ligne reliant les deux extrémités. Cette distance peut être estimée en utilisant les points de contrôle des courbes délimitantes dans le cas des courbes de Bézier. La distance entre les points de contrôle du patch et sa « base » bilinéaire (les quatre coins) sert à évaluer la platitude globale. Clark [Cla79] reprend l'idée de Lane et al. en l'améliorant. La platitude est cette fois estimée avec la courbure de la surface en calculant les dérivées secondes aux quatre coins d'un patch. La méthode proposée par Lane et al. fait apparaître des cracks entre les polygones créés (Figure 3.28). Ces cracks sont évités avec la méthode de Clark, en identifiant les courbes délimitantes sur u ou v considérées comme étant plates, et en les prenant comme telles pour le reste de la subdivision et la génération de triangles qui en découle (Figure 3.31).

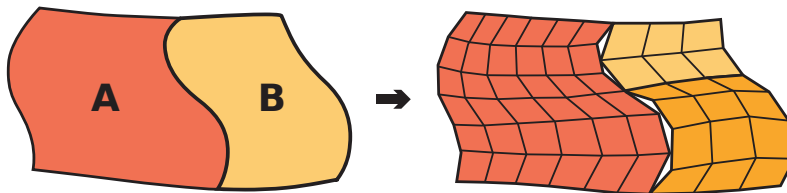


FIGURE 3.28 – Le patch A est tessellé en 6×6 polygones. Le patch B est d'abord subdivisé en deux. Le premier sous-patch est tessellé en 3×2 polygones, le second en 3×4 polygones. Une zone de cracks apparaît ici entre A et B une fois la tessellation effectuée.

Eisenacher et al. [EML09] reprennent le test de platitude proposé par Lane et al. basée sur la distance des points de contrôle, l'idée de Clark pour s'affranchir des cracks, et proposent une implémentation GPU (Figure 3.29). Ils s'inspirent du travail de Patney et Owens [PO08] pour implémenter la subdivision récursive sur GPU au moyen d'un algorithme, récursif par nature, fonctionnant en *largeur d'abord* (de l'anglais *breadth-first*). Un *prefix scan* est utilisé pour ajouter un nombre variable de patches subdivisés d'une manière contiguë en mémoire et en parallèle [HSO08] (Figure 3.30).

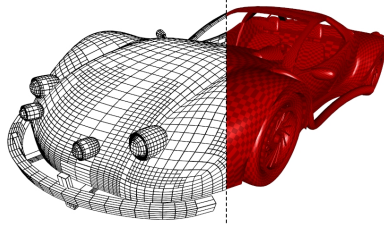


FIGURE 3.29 – Subdivision adaptative d'Eisenacher et al. [EML09]

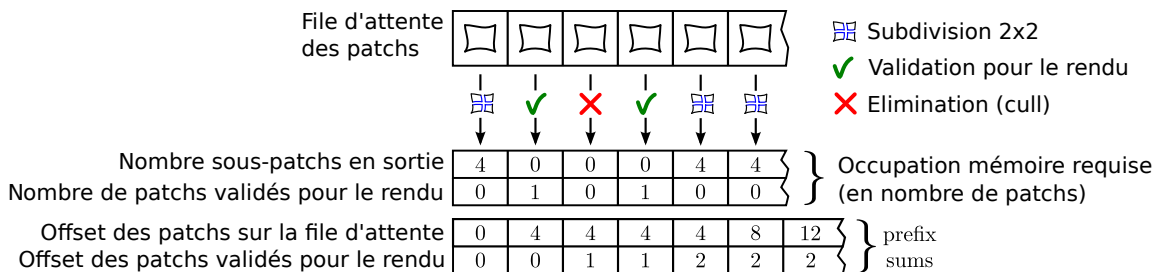


FIGURE 3.30 – Eisenacher et al. [EML09] effectuent un prefix scan pour ajouter un nombre variable de patches à la file d'attente de subdivision, depuis un grand nombre de threads, en parallèle.

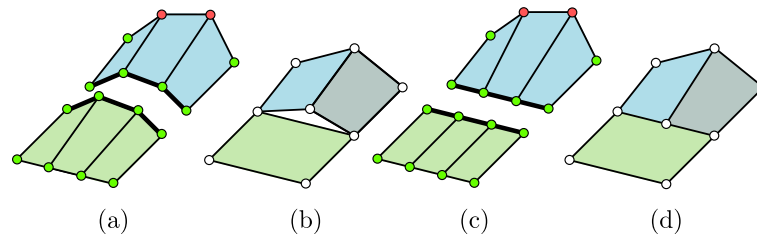


FIGURE 3.31 – Eisenacher et al. [EML09] s’inspirent de la méthode proposée par Clark [Cla79] pour éviter l’apparition de cracks entre les sous-patches d’une même surface, dont les niveaux de subdivision et de tessellation diffèrent. Les bordures de patch presque plates sont modifiées pour réellement devenir plates, en abaissant les points de contrôle intermédiaires.

3.3.4 Micro-polygonisation

La méthode de rendu basée sur la micro-polygonisation a été introduite par Cook et al. [CCC87]. Leur idée se calque sur la subdivision récursive introduite par Lane et al., utilisée pour la tessellation adaptative. Cook et al. proposent un pipeline de rendu dénommé *Reyes*, destiné à produire des images de très haute qualité. Comme nous l’avons vu auparavant, la tessellation adaptative s’arrête suivant un facteur de platitude, avec par exemple une platitude définie en espace écran et où la subdivision s’arrête lorsqu’un patch et son approximation bilinéaire ne dévient pas de plus d’un pixel à l’écran. Avec la méthode de Reyes, une précision d’affichage au demi pixel est recherchée. Au lieu de subdiviser les patches paramétriques jusqu’à ce que leur taille projetée à l’écran soit d’un demi pixel, comme pour l’algorithme de Catmull [Cat74] où aucun critère de platitude n’est utilisé, la méthode de Reyes se propose de subdiviser récursivement les patches jusqu’à ce qu’il soit jugé plus opportun d’avoir recours à la subdivision uniforme pour finir le travail de tessellation (Figure 3.32).

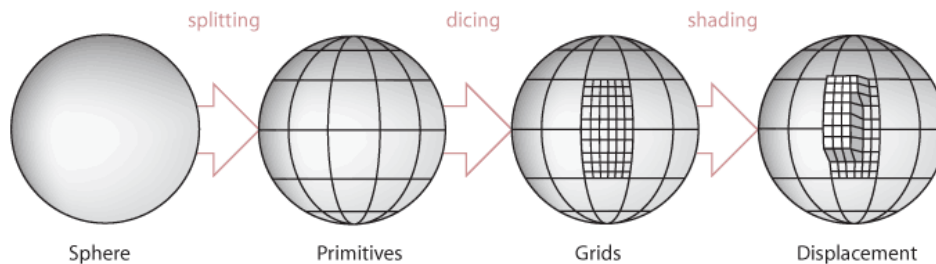


FIGURE 3.32 – Pipeline Reyes et algorithme split and dice. Une sphère, modélisée d’une manière unitaire, est représentée, en entrée du moteur de rendu, par un ensemble de patches de Bézier quadratiques rationnels. Ces patches sont ensuite subdivisés (split), puis finalement tessellés uniformément (dice). L’éclairage final (shading) se fait avec les micro-polygones ainsi créés, dont la taille d’une arête n’excède pas un demi-pixel écran.

La génération de micro-polygones repose ainsi sur une méthode baptisée *split and dice*, qui peut se traduire littéralement par *subdivision et partitionnement en carreaux uniformes*. La division finale en carreaux uniforme est avantageuse car elle permet d’avoir recours à des méthodes de *forward differencing* d’un point discrétisé à l’autre et donc de réduire la quantité de calcul. D’autre part, elle limite l’apparition de cracks entre les patches voisins, la tessellation uniforme limitant le nombre de patches créés et ne créant pas en elle-même de cracks à l’intérieur de chaque patch tessellé. Reyes est depuis longtemps utilisé dans le moteur de rendu *RenderMan*

de Pixar [CFLB06]. Offrant une très grande qualité de rendu et des facilités pour réaliser des effets visuels comme la profondeur de champ, le dé-focus ou le flou directionnel [CCC87], il est très répandu dans le monde du cinéma d’animation. Il offre un temps de calcul avantageux comparé aux méthodes basées sur le lancer de rayon pour les scènes de grandes tailles, ne pouvant tenir en mémoire [LCWB80, CCC87], situation courante dans le monde du cinéma. Patney et Owens [PO08] se sont essayés à une implémentation partielle de Reyes sur GPU, et de sa partie critique, le *split and dice*. Ils implémentent la subdivision récursive avec une queue et utilisent un algorithme permettant d’ajouter de nouveaux éléments sur la queue en parallèle et d’une manière contigüe en mémoire après compactage [SHZO07] (Figure 3.33).

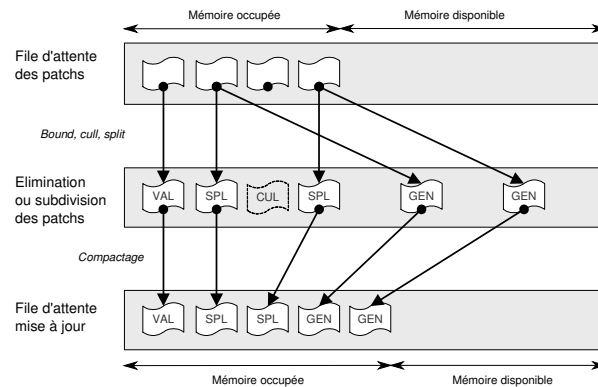


FIGURE 3.33 – Patney et Owens [PO08] ajoutent et suppriment des patches, créant une file d’attente avec des trous. Un algorithme est ensuite utilisé pour compacter l’espace mémoire.

Fisher et al. [FFB*09] s’intéressent aux cracks laissés entre les patches, tessellés indépendamment les uns des autres. Leur approche est basée sur le calcul de discrétisation des courbes en bordure de patches, commune d’un patch voisin à l’autre. Pour éviter qu’une subdivision uniforme d’un patch en 4 morceaux ne vienne créer un crack entre les patches à cause d’un sommet « flottant » utilisé sur un seul côté de la subdivision et pas sur l’autre, ils s’assurent que la subdivision ait lieu à l’emplacement d’un point discrétisé pour chaque courbe de bordure (Figure 3.34, à droite). Cette méthode les oblige à manipuler des patches non quadrangulaires (domaine non isoparamétrique), ce qui vient légèrement surcharger les calculs pour chaque patch. Leur méthode de calcul des pas de discrétisation le long de courbes de bordure leur permet de maîtriser la création de sommets en bordure de patches, quel que soit le niveau de subdivision, de sorte que les cracks n’apparaissent pas entre ces derniers, même à niveau de subdivision différent (Figure 3.34, à gauche).

3.4 Elimination des cracks entre les faces

Nous avons vu dans la Section 2.2 que le rendu par tessellation statique globale effectue la tessellation face par face en s’efforçant de relier entre eux les sommets des arêtes communes entre les faces pour produire un modèle complètement étanche, fermé. Ce mécanisme s’appuie directement sur les sommets discrétisés le long des courbes de découpe, puisque ces dernières mènent directement à la création de points de discrétisation.

Dans le cas du rendu basé découpe, bien que les surfaces support puissent être tessellées, la découpe se fait fragment par fragment et n’engendre pas la création de sommets. Lorsque la tessellation est utilisée pour les surfaces support, des cracks peuvent néanmoins apparaître. Nous

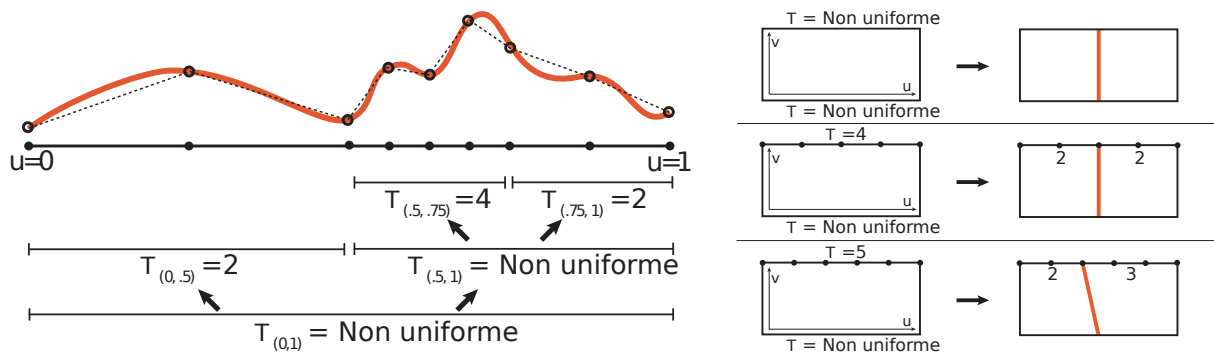


FIGURE 3.34 – A gauche : la discrétisation utilisée par Fisher et al. [FFB*09] le long d'une isoligne de l'espace paramétrique est maîtrisée quel que soit le niveau de subdivision des patches en bordure de cette isoligne. A droite : la subdivision d'un patch se fait au niveau d'un sommet discrétisé en bordure de patch, donc le plus souvent en diagonal. Cette précaution permet d'éviter les cracks.

avons également vu que lorsqu'une méthode rendu précise est utilisée à la fois pour la gestion des surfaces support, comme le lancer de rayon, et pour la découpe [NSK90, SF09], des *jours géométriques* entre les faces, dont nous avons parlé en introduction, peuvent subsister, en raison de leur existence-même dans le modèle.

Certains travaux se proposent d'éliminer les cracks et dans une certaine mesure les jours en effectuant le rendu de géométrie additionnelle venant combler l'espace entre les faces.

Balázs et al. [BGK04] proposent à travers un algorithme dit des *bordures épaisses* (*Fat Borders*) de faire le rendu de bandes de triangles au niveau des courbes de découpe pour remplir les cracks de tessellation et les jours géométriques. Plus exactement, leur objectif est de combler les cracks et jours géométriques ne dépassant pas une taille d'un pixel à l'écran.

Pour chaque face B-Rep, ils discrétisent les courbes de découpe en espace paramétrique, à intervalle régulier, et obtiennent un ensemble de sommets en espace objet. Ils génèrent ensuite une bande de triangles pour chaque courbe de découpe. Au moment du rendu des faces B-Rep, lorsque la tessellation est utilisée pour les surfaces support (l'élimination des cracks reste leur objectif privilégié), ils s'assurent que la déviation maximale entre la définition analytique de ces dernières et leur approximation linéaire utilisée pour le rendu n'excède pas un demi-pixel écran, de sorte que l'éventuel crack créé ne dépasse pas la taille d'un pixel.

Les bandes de triangles doivent ensuite être affichées avec une largeur constante d'un pixel, quel que soit leur emplacement et leur profondeur dans la vue. Trois algorithmes sont proposés pour arriver à cette fin. 2, 4 ou 6 sommets sont pré-générés pour chaque point originellement discrétisé le long d'une courbe de découpe. Ces sommets sont ensuite dynamiquement positionnés avec le shader de sommets de sorte que l'épaisseur de la bande de triangles se rapproche le plus possible de la largeur souhaitée d'un seul pixel. Ce processus est schématisé sur la Figure 3.35.

Les bandes de triangles sont propres aux courbes de découpe de chaque face, ce qui signifie qu'à la jonction entre deux faces, deux bandes sont effectivement créées. Leur idée est basée sur le fait que les géométries additionnelles vont d'une certaine manière étendre d'un demi pixel l'empreinte laissé à l'écran par le rendu d'une face. La jonction entre deux faces adjacentes de différentes couleurs se fera avec deux bandes de triangles de deux couleurs différentes, propres à chaque face. Chaque face peut ainsi contribuer au remplissage du crack ou jour qui existe au niveau d'une courbe spécifique avec sa voisine. Le rendu des bandes de triangle se fait avec les

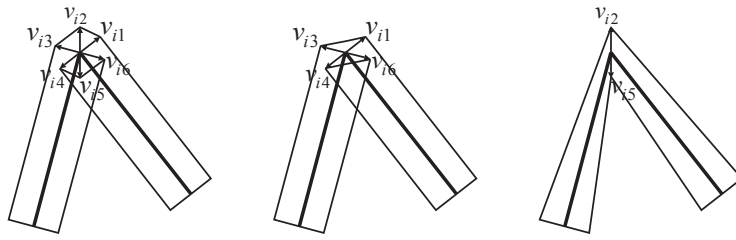


FIGURE 3.35 – 2, 4 ou 6 sommets sont générés par point de discrétisation sur une courbe de découpe. Ces sommets sont positionnés par le shader de sommet pendant le rendu pour produire une ligne d'épaisseur égale à exactement 1 pixel, c'est-à-dire 0.5 pixel de part et d'autre de l'axe d'origine de la discrétisation.

normales obtenues sur la surface au niveau des points discrétisés sur les courbes de découpe, garantissant un bon éclairage.

La génération des sommets v_n de la Figure 3.35, de part et d'autre de l'axe central de discrétisation, se fait en espace objet, et parallèlement au plan de vue. Cela peut produire des artefacts visuels plus ou moins gênants. L'un de ces artefacts est montré sur la Figure 3.36.



FIGURE 3.36 – Artéfact lié à la méthode proposée par Balázs et al. Le positionnement des bandes de triangles pour combler les cracks ou jours reste approximatif et une rastérisation en « en dents de scie » est difficile, sinon impossible, à éviter.

L'artéfact de la Figure 3.36 peut être réduit en déplaçant dans le sens de la profondeur de vue la bande de triangles, par un facteur ϵ exprimé en espace objet. Ce procédé et son effet est illustré sur la Figure 3.37.

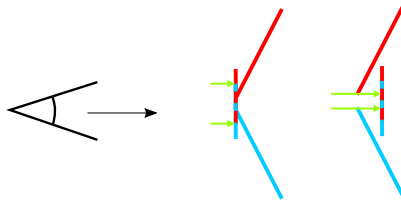


FIGURE 3.37 – Analyse de l'artéfact montré sur la Figure 3.36. En déplaçant légèrement tous les sommets des bandes de triangles (ici deux bandes, une pour la face rouge, l'autre pour la face bleue, adjacente) dans le sens de la profondeur de vue, l'artéfact peut être éliminé.

Après avoir déplacé dans le sens de la profondeur de vue les sommets des bandes de triangle, l'artéfact est éliminé, tel qu'en atteste la Figure 3.38. Cependant, d'autres artefacts ne peuvent être éliminés, notamment ceux causés par le fait que les bandes de triangles d'une face peuvent intersecter une autre face (Figure 3.39).

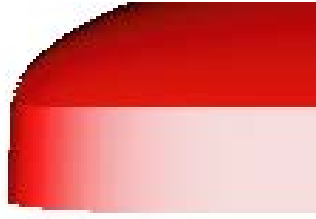


FIGURE 3.38 – L’artéfact de la Figure 3.36 est ici éliminé en déplaçant dans le sens de la profondeur de vue la bande de triangles, au moment du rendu.

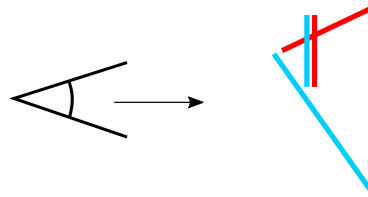


FIGURE 3.39 – L’artéfact ne peut être évité ici. Il est dû au fait que la bande de triangles de la face bleue intersecte visuellement l’affichage de la face rouge.

Pavanaskar et McMains [PM13] s’inspirent de la méthode des *Fat Borders* précédemment décrite. Tout comme Balázs et al., ils discrétisent uniformément des points sur les courbes de découpe et se servent de ces points pour faire le rendu de géométries définies en espace objet et matérialisées là encore sous forme de bandes de triangles. Dans leur algorithme, une seule bande est générée au niveau de la découpe entre deux faces, contre deux pour la méthode de Balázs et al.. Leur méthode permet en théorie de combler les cracks et les jours géométriques de taille arbitraire. L’épaisseur des bandes de triangle, d’un seul pixel avec Balázs et al., est ici déterminée courbe par courbe, pour chaque image. L’épaisseur en pixels de chaque bande est fonction du ratio entre la taille en pixel de l’axe de la bande discrétisé une fois projeté à l’écran (ligne brisée reliant les sommets servant de colonne vertébrale pour la bande), et la longueur de ce même axe en espace objet. La taille en pixel de l’axe de bande est estimé en utilisant une requête d’occlusion, qui permet de compter le nombre de pixels générés par la ligne brisée pendant sa rasterisation.

Les bandes de triangles sont construites de manière similaire à l’algorithme des *Fat Borders*, où seuls 2 sommets sont générés de part et d’autre d’un point de discrétisation sur la courbe découpe (Figure 3.35, droite).

Au moment du rendu, seuls certains fragments issus de ces bandes sont effectivement rasterisés. La rasterisation de ces bandes se fait après celle du modèle d’entrée. Le rasterisation du modèle se fait en mode MRT (*Multiple Render Targets*), et où pour chaque fragment rasterisé est également émis le numéro de surface associé s dans le modèle. Chaque bande de triangles reçoit sous forme de métadonnée, lorsqu’elle est créée, les deux numéros de surface s_1 et s_2 adjacentes. Dans un deuxième temps, lorsque ces bandes de triangle sont rasterisées, un shader de fragment ne valide la rasterisation des fragments individuels que lorsque le numéro de surface s à l’emplacement correspondant dans le framebuffer n’est ni égal à s_1 , ni à s_2 . Ainsi, seuls les fragments identifiés comme cracks ou jours géométriques sont comblés. Ce processus est illustré dans la Figure 3.40.

Balázs et al. affectent aux sommets des bandes de triangles les normales correspondantes sur la surface à laquelle la bande est associée, l’éclairage des fragments étant réalisé avec cette

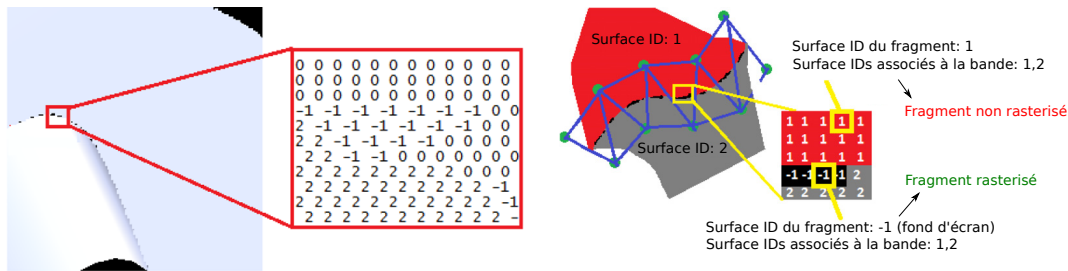


FIGURE 3.40 – Seuls les pixels représentant effectivement des cracks ou des jours doivent être comblés. A gauche, seuls une poignée de pixels dans l'image doit ainsi être mise à jour. Droite : une bande de triangles correspondant à la jonction entre deux surfaces portant les numéros 1 et 2 est rasterisée. Seuls les fragments ayant pour emplacement un pixel dont le numéro de surface n'est ni 1, ni 2, seront effectivement rasterisés, les autres seront ignorés.

normale. Dans le cas de l'algorithme de Pavanaskar et McMains, les normales des deux surfaces associées sont affectées à chaque sommet. Lorsque la rasterisation d'un fragment est validée, l'éclairage est réalisé avec l'une ou l'autre des surfaces s_1 ou s_2 et en prenant la normale associée. Leur algorithme ne précise pas comment déterminer la surface mais il est vraisemblable que le numéro de surface le plus grand ou le plus petit soit choisi (par souci de cohérence, d'un fragment à l'autre le long d'un même crack entre deux faces).

La méthode de validation des fragments basée sur le numéro de surface souffre de limitations, produisant des artefacts visuels si elle est appliquée en l'état. Des bandes de triangles peuvent parfois résider le long des silhouettes en fonction de la configuration du point de vue. Une bande de triangles se trouve alors située sur une sorte de crête, où seule une des deux surfaces adjacente n'est visible, l'autre surface étant située derrière la surface visible, sa normale faisant dos au point de vue. Une partie de la bande se trouve alors sur une surface adjacente, tandis que l'autre recouvre le fond d'écran ou bien une partie du modèle située en arrière plan. Cette seconde partie ne doit absolument pas être rasterisée et le test du numéro de surface est insuffisant pour s'en assurer. Pour être sûr que les fragments associés ne soient pas validés pour la rasterisation, les bandes de triangles ont une épaisseur dynamique le long de leur axe. A chaque fois qu'un sommet d'une bande a des normales n_1 et n_2 correspondant aux surfaces adjacentes s_1 et s_2 caractérisant une silhouette, l'épaisseur de la bande est réduite à 0 du côté de la surface faisant dos à la caméra. Cela permet d'ignorer tous les fragments affectés au côté de la surface non visible et permet de prendre correctement en charge les bandes qui identifient partiellement des silhouettes.

Chapitre 4

Analyse générale de l'état de l'art

En regard des couches logicielles graphiques et de l'architecture matérielle du GPU précédemment présentées et après avoir pris connaissance de la représentation B-Rep ainsi que des contraintes de fonctionnement inhérentes à la programmation sur GPU (Partie I), nous pouvons considérer l'état de l'art dans son ensemble, et l'observer avec un œil critique.

4.1 Des méthodes d'évaluation matures

Relativement à la problématique d'évaluation de points et de dérivées, des méthodes efficaces existent (Chapitre 1) pour prendre en charge les courbes et surfaces de Bézier rationnelles et sont implémentables sur GPU avec un nombre constant de registres, assurant un bon rendement d'exploitation sur GPU (Partie I, Chapitre 3). Rappelons que ces surfaces peuvent représenter l'ensemble des primitives exportées dans les fichiers B-Rep qui nous ont été donnés, avec exactitude. Elles sont très versatiles.

4.2 Limitations de la tessellation statique de modèle

Les méthodes de rendu reposant sur la tessellation globale de modèles (Chapitre 2) sont dans leur ensemble très complexes. En elle-même, la tessellation de face B-Rep est une opération délicate nécessitant un grand nombre d'opérations et de calculs. Bien que le partitionnement de la tessellation passe par une sorte de « grille » de dimension $n_u \times n_v$ plaquée sur la surface support permettant *en théorie* le partitionnement de la problématique de tessellation en zones indépendantes multiples, exécutables en parallèle, la problématique additionnelle d'élimination des cracks vient considérablement alourdir le travail de tessellation. En effet, seule une approche telle que celle proposée par Kumar [Kum96] semble envisageable pour permettre de tesser les faces en temps-réel et de s'affranchir de l'existence de cracks entre les faces. Son approche permet d'éviter de discrétiser les courbes de découpe aux intersections avec les cellules de la grille support, et passe par le calcul d'une représentation d'une courbe commune, en trois dimensions, approximant chaque courbe de découpe définie dans l'espace paramétrique des deux faces adjacentes. La complexité algorithmique de la tessellation est en revanche prohibitive, puisque cette dernière, bien que réalisée pour chaque cellule de la grille, doit être pensée de manière globale en allant chercher des informations situées sur les cellules voisines — voire même au-delà lorsque c'est nécessaire.

En ce qui concerne les autres méthodes de tessellation de face, notamment celles reposant sur une triangulation contrainte de Delaunay, il semble difficile de les réaliser en temps-réel sur

le GPU en raison de leur forte complexité.

Enfin, tous ces algorithmes génèrent un nombre très variable de triangles sur le GPU. L'utilisation du pipeline traditionnel ne permet pas de générer un nombre variable de données, si l'on excepte le shader géométrique (Partie I, Section 3.1) dont l'efficacité est non seulement très limitée puisqu'il n'autorise de générer des données que séquentiellement, mais également réduites en terme de volume, puisque seules des quantités de données inférieures à 1024 sommets sont prises en charge. Passer par un traitement générique parallèle (Partie I, Section 3.1.2) reste envisageable mais le nombre variable de données émises impose l'utilisation d'algorithmes spécifiques qui obligent le développeur à couper en deux son travail, puisqu'il est obligé de prédire à l'avance la quantité de données à générer [HSO08], processus qui peut lui-même impliquer des calculs.

4.3 Opportunités offertes par le rendu basé découpe

Le rendu basé découpe (Chapitre 3) semble au contraire fortement parallélisable. Des méthodes de découpe performantes existent [NSK90] et s'implémentent efficacement sur le GPU [SF09].

Les méthodes de rendu des surfaces de base utilisant le lancer de rayon, la micro-polygonisation ou la tessellation adaptative ne semblent cibler que le rendu *off-line* d'images ou d'animations, et ne pas se prêter au rendu temps-réel. Ces méthodes, coûteuses en terme de temps de calcul peuvent toutefois être utilisées ponctuellement, pour des applicatifs précis (Toledo et Lévy [TL08]). Reste à imaginer d'autres scénarios d'utilisation ciblés pour une utilisation dans le cadre de rendu interactif, permettant d'exploiter intelligemment la qualité de rendu irréprochable offerte par ces technologies. La tessellation des surfaces de base se prête bien aux shaders de tessellation (Partie I, Section 3.1) offerts par les couches logicielles actuelles.

Le rendu basé découpe suit directement la définition naturelle des faces B-Rep. Il est ainsi envisageable de lire le modèle d'entrée dans le logiciel de visualisation et de passer sa définition, aussi brute que possible, au moteur de rendu. Cela représente un avantage appréciable par rapport à la tessellation statique de modèle, où la définition du modèle original est soit perdue, ou alors doit être maintenue en parallèle, une procédure généralement sujette aux erreurs et non souhaitable. Remarquons enfin que ni la conversion des surfaces support en NURBS puis en carreaux de Bézier, ni le traitement associé à la découpe ne sont réellement prohibitifs pour convertir, même en temps réel, des modèles de petite ou moyenne envergure pour une exploitation par un système de rendu basé découpe, au sein même, par exemple, d'un logiciel de modélisation.

4.4 De l'état de l'art aux contributions

Ce chapitre a présenté un premier œil critique sur l'état de l'art. Les Parties suivantes présentent deux contributions. Une plus fine analyse de l'état de l'art y figure au début de chacune d'entre elles, mise dans le contexte de deux problématiques de recherche décrites en détail, préalablement.

Troisième partie

Rendu basé découpe optimisé pour les performances

Chapitre 1

Introduction

1.1 Objectif recherché

La première partie de la thèse s'est faite dans un contexte industriel en collaboration avec un avionneur. La recherche s'est effectuée dans le cadre de la réalisation d'un logiciel permettant de gérer les tests statiques sur les différentes sections d'un avion long courrier en cours de conception.

Dans ce logiciel, les modèles d'avion entiers sont visualisés de manière interactive, sur lesquels un ensemble d'opérateurs appose un certain nombre de capteurs. La Figure 1.4 de la Partie I montre un exemple de capteur utilisé. Ces capteurs sont plats et n'excèdent souvent pas une taille de quelques millimètres. Ils sont placés sur des endroits stratégiques de la maquette numérique et servent à réaliser des mesures. A partir des emplacements sur la maquette virtuelle, des techniciens sont ensuite chargés de placer ces capteurs sur des parties du fuselage dans un hangar, préalablement à la réalisation des tests statiques. Concrètement, une section parfois entière de fuselage réside dans un hangar et les opérateurs s'attendent à voir rigoureusement la même section en intégralité sur la maquette virtuelle sans avoir à sélectionner manuellement différentes sous-parties à afficher.

La taille réduite des capteurs et la grande précision nécessaire à leur emplacement rend l'industriel exigeant en terme de fidélité de rendu. Ces capteurs peuvent être placés sur des chanfreins ou des parties de taille très réduite situées à la jonction de plusieurs pièces, pouvant potentiellement être fragilisées lors de stress répétés appliqués sur la structure. Dans l'absolu, les opérateurs préfèrent avoir une représentation fidèle du modèle pour plus de confort visuel et pour placer d'une manière parfaitement précise leurs capteurs.

Traditionnellement, ces modèles sont rendus dans le logiciel à partir d'une tessellation statique effectuée à une résolution donnée, réalisée dans un prétraitement. Dans le cas du modèle d'avion, cette résolution est fixée à 0.2 millimètre et suffit le plus souvent aux opérations d'ajout de capteur sur la maquette virtuelle. Néanmoins, au moment où notre recherche débute, l'avionneur ne cache pas vouloir accroître cette précision pour gagner en qualité, suivant une exigence du « toujours plus », et pour représenter les surfaces et courbes de la manière la plus lisse possible.

La visualisation de l'ensemble du modèle d'avion ne peut être prise en charge à un niveau de résolution de 0.2 millimètre. Ce niveau épuise les ressources graphiques, tant d'un point de vue occupation mémoire qu'en terme de performance à l'affichage. Seuls les objets en avant plan sont donc affichés avec la résolution de 0.2 millimètre, et les objets en arrière plan utilisent une résolution très inférieure de 5 millimètres. Les objets changent de représentation lorsqu'ils s'approchent suffisamment du point de vue, ce qui induit un effet de *popping* brusque qui peut être gênant pour l'opérateur, notamment en raison du fait que le changement de représentation

se fait en passant par une file d'attente et où les requêtes de changement de représentation sont séquentiellement traitées, souvent avec peine.

L'avionneur est donc intéressé par des méthodes de représentation du modèle permettant de faire tenir l'avion en mémoire en dégradant le moins possible sa représentation visuelle. D'autre part, des méthodes de rendu offrant une grande qualité d'affichage et performantes à l'utilisation sont désirées et recherchées. Enfin, l'utilisation de plusieurs niveaux de résolution n'est pas souhaitable à cause de l'effet de popping.

1.2 Analyse de l'état de l'art

Le rendu par tessellation statique globale est limité par le niveau maximum de discrétisation, qui est fixé à l'avance. Pour répondre aux besoins de l'avionneur en terme de précision à l'affichage, une forte discrétisation est nécessaire. Comme nous l'avons évoqué dans la section précédente, cette forte discrétisation pose aussi bien le problème de l'occupation mémoire comme celui des performances pendant le rendu. Les représentations avec niveau de détail intégré peuvent être utilisées pour améliorer les performances du rendu des objets en arrière plan, au prix d'une certaine latence lors de l'affinage de la résolution des objets proches du point de vue, et d'une consommation mémoire accrue (Partie II, Section 2.4). Deux points noirs dans notre situation. De même, la tessellation statique, quelle que soit sa résolution, produit des arêtes droites et des sommets de discrétisation (Figure 1.1) qui semblent déplaire à l'avionneur. Au final, la méthode de rendu par tessellation statique globale ne semble pas suffisamment bien répondre aux besoins de l'industriel.

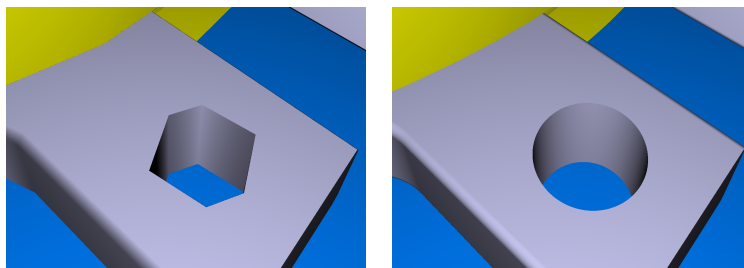


FIGURE 1.1 – La tessellation, et notamment la tessellation statique de modèles entiers, produit des arêtes droites et non lisses, et des points de discrétisation qui n'ont pas lieu d'être.

Les avancées technologiques relatives à la programmabilité des GPU ces dernières années (Partie I, Chapitre 3) permettent chaque jour de se rapprocher d'un idéal : celui de passer au GPU le modèle B-Rep tel qu'il est originellement exporté par le logiciel de modélisation. Nous avons vu dans l'état de l'art de la Partie II des méthodes de rendu *basées découpe* qui permettent d'afficher dynamiquement les modèles avec une résolution variable. Ces méthodes suivent la définition native des faces B-Rep (Partie I, Chapitre 2) et décomposent la problématique de rendu en deux composantes. D'un côté le rendu des surfaces de base (Partie II, Section 2.1.2) et de l'autre, leur découpe (Partie II, Section 2.1.3).

Si le rendu basé découpe offre des perspectives intéressantes, reste à bien maîtriser sa mise en œuvre. Les travaux réalisés ces dernières années montrent que les méthodes de rendu de surface support basées sur le lancer de rayon ou la tessellation adaptative sont bien trop lentes pour prendre en charge de grands modèles. Les travaux tels que celui de Yeo et al. montrent que la tessellation uniforme et dynamique de surfaces de Bézier en temps réel est possible avec de bonnes performances, tirant parti de l'unité de tessellation. Pour prendre en charge les très

grands modèles qui sont les nôtres, ces méthodes restent toutefois insuffisantes. D'une part, de nombreuses surfaces de degré élevé sont impliquées. Les sections d'avion dont nous disposons ont de nombreuses surfaces de degrés 6×6 , voire plus (Chapitre 4, Table 4.1). A l'inverse, de nombreuses faces sont des plans ou de simples formes coniques, certes représentables sous forme de surfaces de Bézier linéaires et quadratiques, mais plus efficacement prises en charge de manière native.

La découpe telle qu'elle est proposée par Schollmeyer et Fröhlich [SF09](Partie II, Chapitre 3) permet de classier des points dans l'espace paramétrique des surfaces d'une manière efficace et d'une manière précise. Pour autant, la structure proposée est relativement coûteuse en terme de temps de calcul pendant le rendu, notamment pour les surfaces proches du point de vue. En effet, pour chaque fragment, le parcours de deux arbres binaires doit être opéré. S'en suit une éventuelle classification via une ou plusieurs section(s) bimonotone(s) de courbes de découpe. Pour les surfaces situées en arrière plan, l'absence de niveau de détail dans la structure rend obligatoire le parcours du double arbre binaire et l'accès aux courbes de découpe, en dépit de la faible résolution recherchée, dépendante de l'échelle d'affichage. Par ailleurs, la découpe des faces B-Rep contient généralement de très grandes zones homogènes, n'intersectant aucune courbe de découpe. Aucune structure ne permet à l'heure actuelle de classier d'un bloc de telles zones, ce qui serait pourtant souhaitable pour les surfaces situées en avant plan où le nombre de fragment à classier explose (notamment si la résolution de l'écran est élevée).

1.3 Vue d'ensemble de la méthode proposée

Nous présentons une méthode de rendu basée découpe permettant de visualiser de très grands modèles B-Rep avec une précision visuelle élevée, et à une cadence d'affichage autorisant l'interaction voire le temps réel. Pour garantir de bonnes performances, nous nous démarquons de l'état de l'art de plusieurs manières.

Tout d'abord, nous prenons en charge les surfaces support avec la tessellation dynamique pour garantir de bonnes performances, mais en *spécialisant* le code d'évaluation des surfaces pour de nombreux types de surfaces connus (plans, cylindres, cones, tores...). Nous approximations les surfaces pour lesquelles nous n'avons pas de code de tessellation dédié sous forme de patches de Bézier cubiques, rapides à prendre en charge pour le GPU. Cette approximation créée des jours entre les faces, d'une taille contrôlable, sur lesquels nous allons revenir un peu plus loin.

Notre stratégie de découpe est, comme pour celles de Nishita et al. [NSK90] et Schollmeyer et Fröhlich [SF09], basée sur une structure vectorielle. Elle permet d'obtenir des courbes lisses, une particularité visuelle à laquelle sont sensibles de nombreux utilisateurs. Contrairement aux papiers précédemment cités, nous choisissons délibérément une approximation du modèle d'entrée, et optons pour une représentation basée sur des courbes implicites. Comme pour les surfaces support, cette approximation créée elle aussi un jour (ou exacerbe un jour déjà existant) entre les faces, d'une taille également contrôlable. En contrepartie, elle permet de gagner en rapidité au moment du processus de classification.

L'approximation des surfaces support et de la découpe dégrade légèrement la qualité de représentation des données avec des conséquences sur l'affichage, quoique minimes, matérialisées, nous l'avons mentionné avant, sous forme de jours. Ce choix difficile d'avoir recours à des approximations afin d'améliorer les performances est basé sur les observations suivantes. Tout d'abord, ces approximations étant de taille contrôlable, elles peuvent être réduites à 0.1 ou 0.01 millimètre, et n'être donc perceptibles qu'à des niveaux de zoom extrêmement forts. Ensuite, avec le rendu basé découpe, des *cracks* se manifestent systématiquement entre les faces à partir du moment

où la tessellation est utilisée pour la prise en charge des surfaces support ; ils sont inévitables (Figure 1.2). Par ailleurs, nous avons rappelé dans la Partie I, Section 2.2.1 que des *jours* géométriques peuvent également exister entre certaines faces. Ces jours résident dans la géométrie même du modèle et ne sont pas causés par la tessellation au moment du rendu. Contrairement aux cracks de tessellation dont on peut limiter l'épaisseur en maîtrisant les facteurs de tessellation, ces jours, dont la taille est physiquement mesurable, sont d'autant plus visibles que le niveau de zoom est fort. Des méthodes comme celles proposées par Balázs et al. [BGK04] ou Pavanaskar et McMains [PM13] permettent dans une certaine mesure de se débarrasser des cracks comme des jours topologiques. Ces méthodes utilisent des bandes géométriques entre les faces pour combler explicitement les espaces causés par ces cracks et ces jours. Dès lors, il nous paraît opportun d'approximer la découpe des faces pour gagner en performances au moment du rendu, quitte à exacerber légèrement les jours à la jonction entre les faces, puisque ces jours ont une taille pouvant être contrôlable, et gardant à l'esprit que s'ils doivent réellement être éliminés, des méthodes, malgré certaines limites, existent pour le faire.

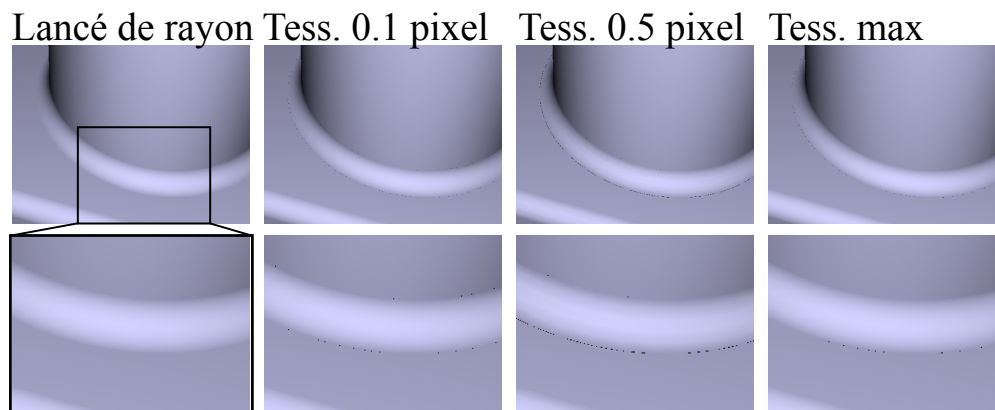


FIGURE 1.2 – Cracks entre deux faces adjacentes, causés par la tessellation. Seule un rendu des surfaces support avec un lancer de rayon permet ici un résultat dénué de crack (à gauche). Pousser les facteurs de tessellation à un niveau très élevé (63 à droite, ce qui correspond à la limite imposée par l'unité de tessellation matérielle) n'est pas suffisant pour faire disparaître tous les cracks.

Enfin, pour accélérer la classification des fragments distants, notre structure de découpe est multirésolution par nature. Cela nous permet de classifier efficacement des fragments en limitant les accès mémoire, coûteux lorsqu'ils sont trop nombreux ou réalisés en dehors des zones opportunément mises en cache par le processeur graphique (Partie I, Section 3.2). Enfin, dans le cas des surfaces planes qui sont particulièrement nombreuses dans les modèles (voir les statistiques sur les modèles Table 4.1 dans le chapitre suivant), nous nous appuyons sur la génération de triangles issus de la tessellation matérielle pour classifier de larges zones de l'espace de découpe, ce qui améliore les performances. La Figure 1.3 permet de se rendre compte des résultats obtenus avec notre méthode.

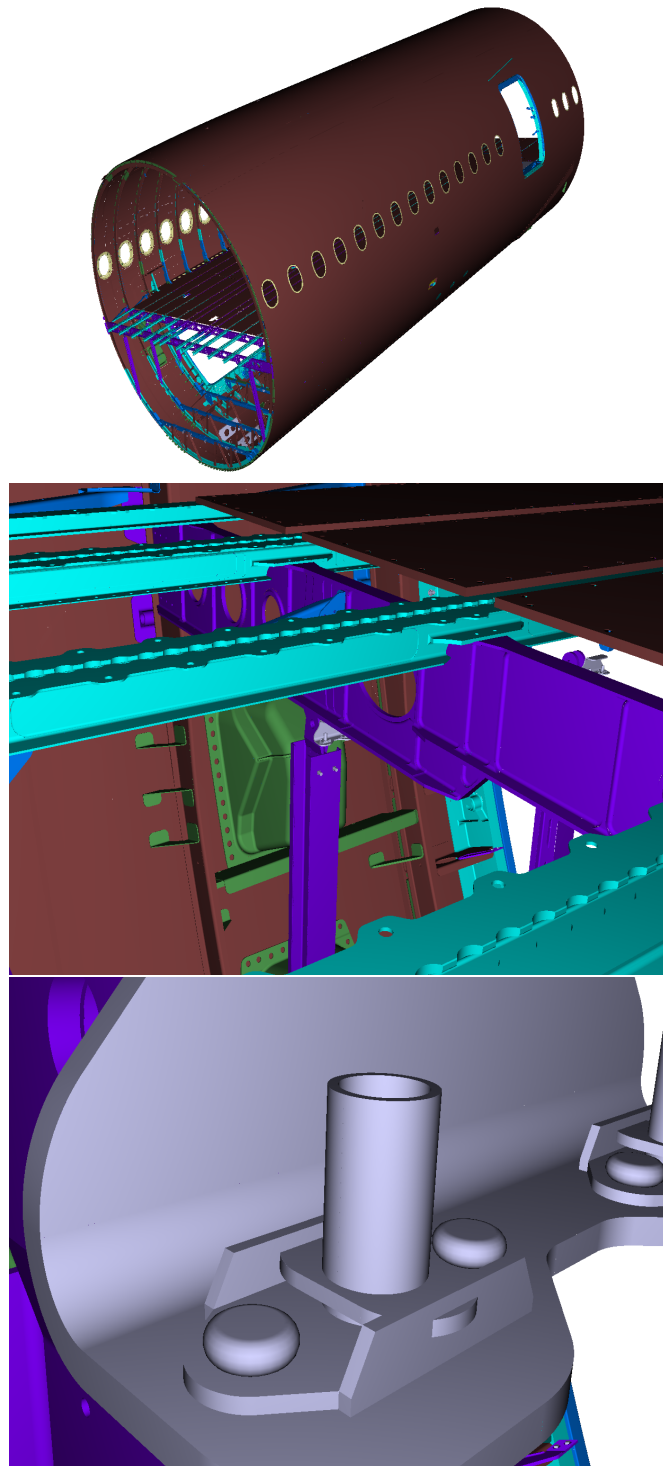


FIGURE 1.3 – Notre méthode de classification de point pour la gestion de la découpe pendant le rendu des surfaces B-Rep s'adapte bien au niveau de zoom. Quand les surfaces sont distantes de la caméra, un mécanisme multirésolution réduit le coût de la classification. Lorsqu'elles sont proches, nous nous appuyons sur la tessellation matérielle et passons par la classification de triangles entiers, lorsque c'est possible, pour accélérer l'affichage. L'utilisation de shaders de tessellation dédiés permet d'effectuer rapidement le rendu des surfaces support. Notre méthode crée un jour additionnel entre les faces, d'une taille paramétrable et suffisamment petite pour être invisible, y compris à fort niveau de zoom.

Chapitre 2

Approximation du modèle d'entrée

Avant d'être passé à notre pipeline de rendu, le modèle B-Rep d'entrée est approximé suivant une procédure décrite dans ce chapitre. Nous décrivons d'abord l'approximation de la découpe des faces, effectuée dans le domaine paramétrique des surfaces support, puis nous intéressons à la représentation des surfaces support par la suite.

2.1 Découpe

2.1.1 Implicitisation des courbes de découpe

2.1.1.1 Echantillonnage et interpolation par des B-Splines

Les courbes de découpe du modèle B-Rep d'entrée sont tout d'abord approximées avec des courbes de Bézier quadratiques, en respectant une erreur d'approximation ϵ , exprimée en espace objet. Nous discrétisons chaque courbe d'entrée et effectuons une interpolation des points, produisant un B-Spline uniforme de degré 2. Une courbe B-Spline quadratique (degré 2, ordre 3) est définie selon l'équation suivante :

$$\mathbf{P}(t) = \sum_{i=1}^{n+1} N_i^3(t) \mathbf{P}_i \quad (2.1)$$

où \mathbf{P}_i sont les $n + 1$ points de contrôle du B-Spline. Nous cherchons les points de contrôle P_i vérifiant

$$\mathbf{P}(v_j) = \sum_{i=1}^{n+1} N_i^3(v_j) \mathbf{P}_i = \mathbf{M}_j \quad (2.2)$$

où \mathbf{M}_j représentent les $j + 1$ points à interpoler. Les valeurs de v_j sont réparties sur le vecteur nodal du B-Spline, à intervalle régulier. Ce vecteur nodal est choisi de sorte que $u_{i+1} - u_i = 1 \forall i$. Développer l'équation (2.2) revient à écrire

$$N_1^3(v_j) P_1 + N_2^3(v_j) P_2 + \dots + N_{n+1}^3(v_j) P_{n+1} = M_j \quad (2.3)$$

ou encore, formulé avec une représentation matricielle :

$$\begin{bmatrix} N_1^3(v_1) & N_2^3(v_1) & \cdots & N_{n+1}^3(v_1) \\ N_1^3(v_2) & N_2^3(v_2) & \cdots & N_{n+1}^3(v_2) \\ \vdots & \vdots & \ddots & \vdots \\ N_1^3(v_{n+1}) & N_2^3(v_{n+1}) & \cdots & N_{n+1}^3(v_{n+1}) \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{n+1} \end{bmatrix} = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n+1} \end{bmatrix} \quad (2.4)$$

Calculer la position des points M_i revient à résoudre le système linéaire (2.4). Néanmoins, la courbe B-Spline produite doit suivre avec le plus de fidélité possible la courbe originelle, y compris les sections situées entre les points échantillonnés. Nous ajoutons donc des conditions de tangentes au système linéaire, en ajoutant un point de contrôle pour la tangente V_1 de début, et un autre pour la tangente V_{n+1} de fin, afin de déterminer les $n + 3$ points de contrôle d'une B-Spline interpolante. Le système linéaire devient

$$\begin{bmatrix} N_1^3(v_1) & N_2^3(v_1) & \cdots & N_{n+1}^3(v_1) \\ -N_2^2(v_1) & N_2^2(v_1) - N_3^2(v_1) & \cdots & N_{n+1}^2(v_1) \\ N_1^3(v_2) & N_2^3(v_2) & \cdots & N_{n+1}^3(v_2) \\ N_1^3(v_3) & N_2^3(v_3) & \cdots & N_{n+1}^3(v_3) \\ \vdots & \vdots & \ddots & \vdots \\ N_1^3(v_{n+1}) & N_2^3(v_{n+1}) & \cdots & N_{n+1}^3(v_{n+1}) \\ -N_2^2(v_{n+1}) & N_2^2(v_{n+1}) - N_3^2(v_{n+1}) & \cdots & N_{n+1}^2(v_{n+1}) \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ \vdots \\ P_{n+2} \\ P_{n+3} \end{bmatrix} = \begin{bmatrix} M_1 \\ V_1 \\ M_2 \\ V_2 \\ \vdots \\ M_{n+1} \\ V_{n+1} \end{bmatrix} \quad (2.5)$$

La résolution de ce système linéaire nous permet de dégager la définition de notre courbe. Rappelons qu'avec un vecteur normal uniforme vérifiant $u_{i+1} - u_i = 1 \forall i$, on a

$$\frac{d}{du} N_i^k(u) = \frac{m}{u_{m+i-1} - u_{i-1}} N_i^{k-1}(u) - \frac{m}{u_{m+i} - u_i} N_{i+1}^{k-1}(u) = N_i^{k-1}(u) - N_{i+1}^{k-1}(u) \quad (2.6)$$

2.1.1.2 Approximation sous contrôle d'erreur

Une fois la courbe B-Spline interpolante construite, elle contient un très grand nombre de points de contrôle. Nous cherchons à réduire ce nombre de points de sorte que la courbe résultante ne devie pas de plus de ϵ_u unités sur l'axe paramétrique u et de ϵ_v unités sur l'axe paramétrique v de la courbe originale. Nous utilisons donc un algorithme de suppression de nœuds soumis à une contrainte de déviation maximale indépendante sur les axes u et v , tel que celui de Lyche et Morken [LM87, LM88]. Il est intéressant de traiter les axes paramétriques u et v de manière indépendante, dans la mesure où le domaine de définition de la surface sur laquelle les courbes de découpe résident est parfois fortement anisotropique. Pour déterminer ϵ_u et ϵ_v , nous estimons par le biais d'un échantillonnage la longueur maximale en espace objet des isolignes paramétriques à u ou v constant le long de la surface, desquelles nous déduisons respectivement ϵ_v et ϵ_u .

2.1.1.3 Décomposition pièce à pièce en courbes de Bézier

Une fois que les courbes B-Spline quadratiques approximées sont créées, l'algorithme d'Oslo [CLR80] est utilisé pour décomposer les B-Splines en courbes de Bézier quadratiques. Ces courbes quadratiques vont être référencées dans la structure de découpe, mais sous une forme implicite, pour permettre la classification.

2.1.1.4 Reconstruction implicite locale

En suivant la convention utilisée pour les courbes de découpe d'entrée, nos courbes quadratiques sont toujours orientées de telle sorte que la partie *sur la face* se situe toujours à gauche de la courbe. La classification de point consiste à déterminer de quel côté de la courbe un point se situe. A l'intérieur de l'enveloppe convexe du polygone de contrôle de la courbe quadratique, nous effectuons la classification de point avec la méthode introduite par Loop et Blinn [LB05].

Loop et Blinn utilisent une fonction $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, définie par $f(x, y) = y - x^2$ si la face est localement convexe et $f(x, y) = x^2 - y$ si la face est localement concave, pour représenter une courbe de Bézier quadratique, où (x, y) est exprimé dans l'espace quadratique défini par un repère quadratique (Figure 2.1). Dans ce repère, la courbe quadratique est l'ensemble des valeurs nulles de f . Le repère est aussi défini de sorte que les points de contrôle successifs p_i , $i = 0, 1, 2$ de la courbe de quadratique aient des coordonnées (x_i, y_i) successivement égales à $(0, 0)$, $(0.5, 0)$ et $(1, 1)$ (Figure 2.1). Dans l'enceinte de l'enveloppe convexe du polygone de contrôle de la courbe quadratique, les coordonnées (x, y) évaluées en $f(x, y) \geq 0$ sont *sur la face*.

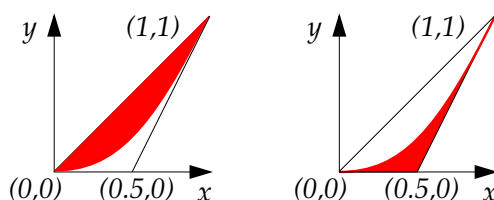


FIGURE 2.1 – Illustration de la partie *sur la face* (en rouge) définie par la représentation implicite de la courbe de Bézier dans l'espace quadratique. À gauche : une zone sur la face convexe vérifiant $y - x^2 \geq 0$. À droite : une zone sur la face concave vérifiant $x^2 - y \geq 0$.

Pour classifier un point de coordonnées (u, v) dans l'espace paramétrique, ses coordonnées (x, y) en espace quadratique sont calculées comme suit :

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 2(u_{p_1} - u_{p_0}) & u_{p_0} - 2u_{p_1} + u_{p_2} & -u_{p_0} \\ 2(v_{p_1} - v_{p_0}) & v_{p_0} - 2v_{p_1} + v_{p_2} & -v_{p_0} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

où (u_{p_i}, v_{p_i}) , $i = 0, 1, 2$ sont les coordonnées respectives des points de contrôle de Bézier dans l'espace paramétrique. Ensuite, évaluer $f(x, y)$ indique si le point est *sur la face* ($f(x, y) \geq 0$) ou *hors de la face* ($f(x, y) < 0$).

2.1.2 Structure multirésolution d'accès à l'espace de découpe

La représentation multirésolution de la découpe de la face s'appuie sur une subdivision récursive de l'espace paramétrique, matérialisée par un quadtree.

Les cellules feuilles de l'arbre de subdivision identifient une zone complètement *sur la face* ou *hors de la face*, ou bien une zone qui est traversée par deux courbes quadratiques au plus. La subdivision de cellule continue jusqu'à ce que chaque cellule respecte ces contraintes. Dans les feuilles sont stockées jusqu'à deux références de courbes quadratiques intersectantes.

La présence de deux références à des courbes quadratiques dans une cellule est inévitable, les points de jonction entre des courbes quadratiques adjacentes n'ayant quasiment aucune chance de tomber sur des frontières de cellule. Un quadtree plutôt qu'un kd-tree, qui pourrait nous faire contourner ce problème, est utilisé pour les possibilités d'accès multirésolution, qui seront

expliquées au chapitre suivant. Par ailleurs, deux courbes de découpe distinctes peuvent parfois se trouver tellement proches l'une de l'autre qu'une division récursive de la cellule doit être faite d'une manière très profonde avant que les contraintes de cellule ne soient respectées. La Figure 2.2 montre un quadtree construit sur le domaine paramétrique pour deux faces B-Rep complexes.

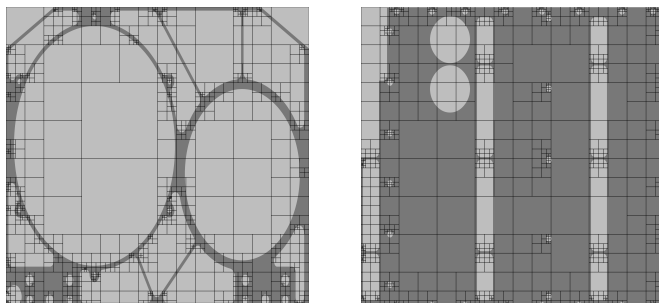


FIGURE 2.2 – *Quadtrees générés pour deux faces complexes comportant de nombreuses boucles de découpe.*

Lorsque deux courbes quadratiques traversent une cellule, nous précalculons une ligne séparatrice. En considérant seulement les portions de courbe qui traversent effectivement les cellules, une ligne séparatrice est une ligne qui ne coupe aucune des deux courbes quadratiques, avec une courbe de chaque côté (voir Figure 2.5(c)) ; elle peut être construite à l'aide du théorème de l'axe de séparation [Ebe06, p. 393] : étant donnés deux polygones convexes disjoints, nous cherchons une ligne avec un polygone de chaque côté. Un segment d'un des deux polygones peut servir de support à cette ligne de séparation (Figure 2.5(c)). Nous cherchons un tel segment en subdivisant les polygones de contrôle jusqu'à ce qu'ils ne soient pas sécants entre eux. Les lignes construites à partir des segments des polygones de contrôle, progressivement subdivisés (par exemple avec le schéma de De Casteljau) sont testées en vérifiant que l'autre polygone de contrôle n'est pas coupé. La première ligne qui convient est choisie comme ligne de séparation pour la cellule. Si aucune ligne ne peut être trouvée après 100 itérations, nous en déduisons qu'aucune ligne de séparation n'existe pour séparer les deux segments de courbe intersectant la cellule, et nous subdivisons cette dernière.

La Figure 2.3 montre une structure de découpe finalement obtenue.

La subdivision récursive descendante des cellules est suivie par un calcul de la valeur de couverture, obtenue en remontant le quadtree, de bas en haut (Figure 2.4). Cette valeur de couverture, binaire, est utilisée pendant le rendu comme mécanisme multirésolution (détaillé dans la Section 3.2.1 du chapitre suivant).

La construction de notre structure est résumée sur la Figure 2.5.

2.2 Surfaces support

2.2.1 Primitives simples

Les primitives telles que les plans, les cylindres, les cônes et les tores, pour lesquelles nous avons des shaders de tessellation spécialisés, ne sont pas approximées. Elles sont enregistrées en tant que telles dans le modèle de sortie.

Les fillets sont approximés. Un fillet est, dans sa définition B-Rep, défini sous forme d'une sorte de section de tube, et de diamètre variable sur sa longueur. L'axe paramétrique u suit le fillet sur sa longueur, tandis que l'axe paramétrique v suit la définition du tube sur sa révolution.

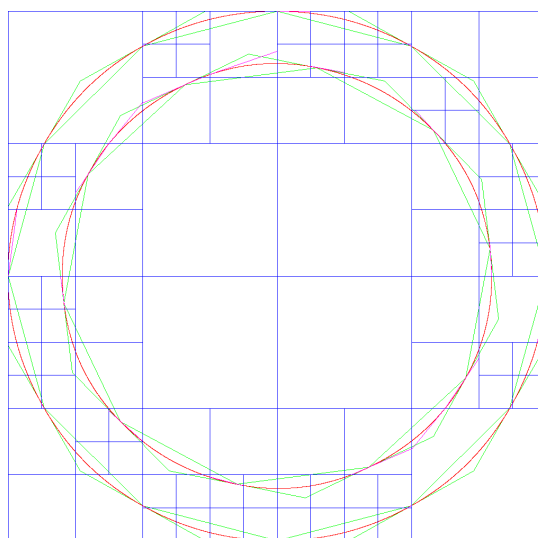


FIGURE 2.3 – Structure de découpe obtenue avec notre procédure de construction. Les lignes de séparation des cellules du quadtree sont en bleu. Les courbes de découpe (Bézier quadratiques) sont en rouge, leur polygone de contrôle en vert. Les lignes de séparation apparaissent en violet.

Un fillet est défini avec 3 courbes. La première courbe est la colonne vertébrale et définit l'axe de révolution. Les deux autres sont appelées *courbes d'adjacence* et délimitent le rayon de révolution du fillet sur sa longueur. Le fillet est défini comme étant l'ensemble des arcs de cercle dont le centre se trouve sur la colonne vertébrale et dont les extrémités se trouvent sur les deux courbes d'adjacence pour une valeur de paramètre donnée. La paramétrisation de ces trois courbes est donc homogène sur toute la longueur du fillet. Nous approximations la colonne vertébrale et les deux courbes d'adjacence avec des courbes de Bézier cubiques suivant un procédé analogue à celui décrit pour les courbes de découpe en Section 2.1.1. L'erreur d'approximation ϵ choisie représente la déviation maximale à respecter entre une courbe d'origine et son approximation avec une polyligne brisée.

2.2.2 Autres primitives

Les autres primitives rencontrées dans les modèles B-Rep sont approximées avec des patches de Bézier bicubiques, moyennant une erreur d'approximation ϵ représentant la déviation maximale à respecter entre la surface support d'origine et les patches de Bézier l'approximant. Cette approximation entraîne la perte d'une éventuelle continuité C^2 d'une face à l'autre, mais ce n'est pas gênant dans la mesure où l'affichage des modèle CAO dans un contexte industriel ne requiert pas le rendu de reflets lumineux. Les patches bicubiques sont choisis car il est aisé d'évaluer des points et des dérivées avec le GPU sur ces derniers ; ceci est effectué au moyen d'opérations matricielles et avec des matrices 4×4 pour lesquelles les langages de shader disposent d'instructions spécialisées.

Chaque surface est d'abord échantillonnée à intervalles paramétriques u et v réguliers. Les points de contrôle d'un patch B-Spline de degré 3×3 sont ensuite recherchés de sorte que le patch résultant interpole les points échantillonnés et respecte les dérivées partielles sur u et v pour les points situés en bordure de l'échantillon. Le patch B-Spline est défini avec l'équation suivante :

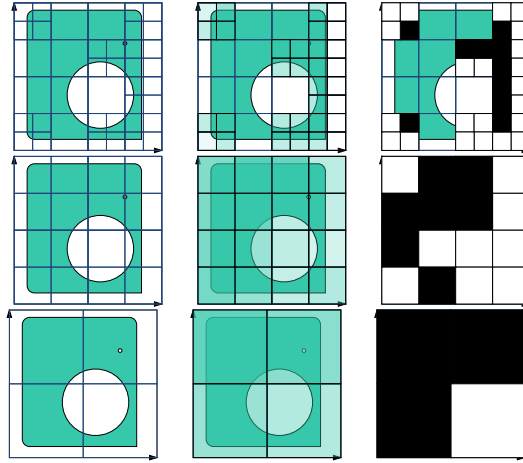


FIGURE 2.4 – Une valeur binaire de couverture est calculée pour chaque cellule. Cette valeur est définie à 1 (cellules noires) si la cellule couvre majoritairement ($\geq 50\%$) une zone dans l'espace de découpe, 0 dans le cas contraire (cellules blanches).

$$\mathbf{P}(s, t) = \sum_{i=1}^{n+1} \sum_{j=1}^{m+1} N_i^3(s) N_j^3(t) \mathbf{P}_{ij} \quad (2.7)$$

Pour trouver ses points de contrôle, une méthode similaire à celle décrite dans la Section 2.1 est utilisée, où le problème est exprimé sous la forme d'un système linéaire. Nous cherchons les points de contrôle P_{ij} vérifiant

$$\mathbf{P}(s_k, t_l) = \sum_{i=1}^{n+1} \sum_{j=1}^{m+1} N_i^3(s_k) N_j^3(t_l) \mathbf{P}_{ij} = M_{kl} \quad (2.8)$$

où M_{kl} représentent les $k \times l$ points de contrôle à interpoler. Les valeurs s_k et t_l sont réparties uniformément sur les vecteurs nodaux, avec $s_{k+1} - s_k = 1 \forall k$ et $t_{l+1} - t_l = 1 \forall l$. Le système linéaire se définit par (en posant $n+1 = m+1 = 3$ et en omettant ici les conditions de tangentes pour plus de clarté) :

$$\begin{bmatrix} N_1^3(s_1) & N_2^3(s_1) & N_3^3(s_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ N_1^3(s_2) & N_2^3(s_2) & N_3^3(s_2) & 0 & 0 & 0 & 0 & 0 & 0 \\ N_1^3(s_3) & N_2^3(s_3) & N_3^3(s_3) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & N_1^3(s_1) & N_2^3(s_1) & N_3^3(s_1) & 0 & 0 & 0 \\ 0 & 0 & 0 & N_1^3(s_2) & N_2^3(s_2) & N_3^3(s_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & N_1^3(s_3) & N_2^3(s_3) & N_3^3(s_3) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & N_1^3(s_1) & N_2^3(s_1) & N_3^3(s_1) \\ 0 & 0 & 0 & 0 & 0 & 0 & N_1^3(s_2) & N_2^3(s_2) & N_3^3(s_2) \\ 0 & 0 & 0 & 0 & 0 & 0 & N_1^3(s_3) & N_2^3(s_3) & N_3^3(s_3) \end{bmatrix} \cdot \begin{bmatrix} P_{1,1} \\ P_{2,1} \\ P_{3,1} \\ P_{1,2} \\ P_{2,2} \\ P_{3,2} \\ P_{1,3} \\ P_{2,3} \\ P_{3,3} \end{bmatrix} = \begin{bmatrix} T_{1,1} \\ T_{2,1} \\ T_{3,1} \\ T_{1,2} \\ T_{2,2} \\ T_{3,2} \\ T_{1,3} \\ T_{2,3} \\ T_{3,3} \end{bmatrix} \quad (2.9)$$

où $T_{k,l}$ représentent des évaluations intermédiaires sur le paramètre s , et

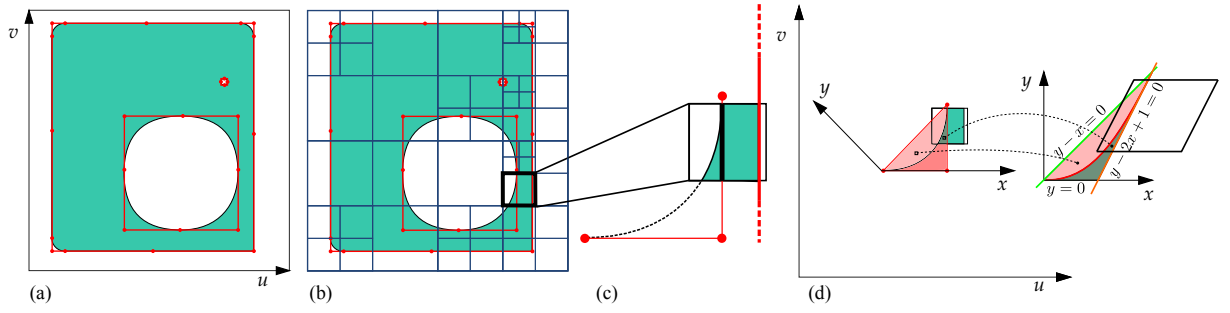


FIGURE 2.5 – (a) Approximation quadratique des courbes de découpe. Les points et polygones de contrôle sont affichés en rouge. (b) Quadtree : chaque cellule peut recouvrir une zone complètement sur la face, hors de la face, ou partiellement sur et hors, avec une ou deux courbes quadratiques qui traversent la cellule. (c) Lorsqu’une feuille a deux courbes quadratiques qui la traverse, les portions de courbes intersectantes sont isolées par une ligne séparatrice qui est stockée dans la cellule. (d) Pour évaluer une courbe quadratique d’une manière implicite, nous transformons ses coordonnées paramétriques (u, v) en coordonnées quadratiques (x, y) .

$$\begin{bmatrix}
 N_1^3(t_1) & 0 & 0 & N_2^3(t_1) & 0 & 0 & N_3^3(t_1) & 0 & 0 \\
 N_1^3(t_2) & 0 & 0 & N_2^3(t_2) & 0 & 0 & N_3^3(t_2) & 0 & 0 \\
 N_1^3(t_3) & 0 & 0 & N_2^3(t_3) & 0 & 0 & N_3^3(t_3) & 0 & 0 \\
 0 & N_1^3(t_1) & 0 & 0 & N_2^3(t_1) & 0 & 0 & N_3^3(t_1) & 0 \\
 0 & N_1^3(t_2) & 0 & 0 & N_2^3(t_2) & 0 & 0 & N_3^3(t_2) & 0 \\
 0 & N_1^3(t_3) & 0 & 0 & N_2^3(t_3) & 0 & 0 & N_3^3(t_3) & 0 \\
 0 & 0 & N_1^3(t_1) & 0 & 0 & N_2^3(t_1) & 0 & 0 & N_3^3(t_1) \\
 0 & 0 & N_1^3(t_2) & 0 & 0 & N_2^3(t_2) & 0 & 0 & N_3^3(t_2) \\
 0 & 0 & N_1^3(t_3) & 0 & 0 & N_2^3(t_3) & 0 & 0 & N_3^3(t_3)
 \end{bmatrix} \cdot \begin{bmatrix} T_{1,1} \\ T_{2,1} \\ T_{3,1} \\ T_{1,2} \\ T_{2,2} \\ T_{3,2} \\ T_{1,3} \\ T_{2,3} \\ T_{3,3} \end{bmatrix} = \begin{bmatrix} M_{1,1} \\ M_{2,1} \\ M_{3,1} \\ M_{1,2} \\ M_{2,2} \\ M_{3,2} \\ M_{1,3} \\ M_{2,3} \\ M_{3,3} \end{bmatrix} \quad (2.10)$$

En substituant l’équation (2.9) dans l’équation (2.10) et en développant le produit de matrice obtenu, le système linéaire peut être dégagé. Une fois le patch B-Spline interpolant défini, il est simplifié de sorte que la version simplifiée ne dévie pas de plus de ϵ unités par rapport à la surface d’origine, en espace objet. La méthode proposée par Lyche et Morken est une nouvelle fois utilisée [LM87, LM88].

Pour finir, le patch B-Spline est décomposé en carreaux de Bézier avec un algorithme d’insertion de nœuds, comme celui d’Oslo [CLR80].

Chapitre 3

Rendu

3.1 Rendu des surfaces support

Nos primitives sont prises en charge par des shaders de tessellation dédiés pour améliorer les performances lors de l'évaluation des points et des normales, mais aussi pour rendre le calcul des facteurs de tessellation plus efficace. Nous avons des shaders dédiés pour les cylindres, les cônes, les tores, les fillets et les patches de Bézier bicubiques. Les primitives sont traitées par le GPU type après type, de sorte qu'un changement de programme GPU ne soit opéré qu'une seule fois par type. Ainsi, tous les plans sont rendus, puis tous les cylindres, les cônes etc. Nous avons vu dans le Chapitre 2 comment ces primitives sont converties, ou approximées dans le cas des fillets et des patches de Bézier cubiques, à partir du modèle d'entrée. Pendant le rendu, nous tessellons chaque surface support de sorte que le maillage produit et projeté à l'écran ne dévie pas de plus de ϵ_s pixels de l'empreinte de la définition analytique correspondante.

Pour les cylindres et les cônes, le facteur de tessellation sur v vaut toujours 1 puisque les surfaces sont linéaires sur cet axe. Le facteur de tessellation sur u est calculé en se référant à la projection écran d'une boule définie en espace objet, dont le centre est situé sur l'axe de révolution et dont la position sur l'axe est égale à l'une ou l'autre des extrémités sur v (Figure 3.1, à gauche). Le rayon de la boule est dans les deux cas le rayon maximal du cylindre ou du cône — ce rayon est constant pour le cylindre mais variable pour le cône.

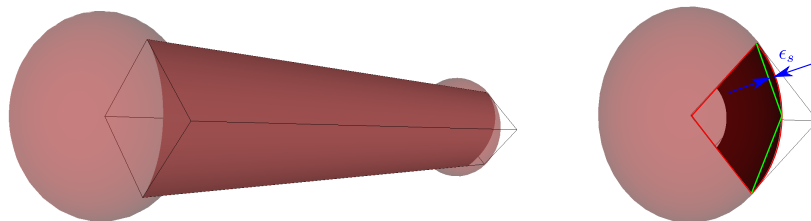


FIGURE 3.1 – A gauche : boules dont la projection sert de référence pour le calcul du facteur de tessellation sur l'axe paramétrique u . A droite : la boule la plus proche est projetée et le domaine paramétrique à tesseller est matérialisé sous forme d'un arc de cercle. L'erreur cordale correspond à la déviation écran maximale ϵ_s à respecter.

Une fois les deux boules situées aux extrémités v de l'axe de révolution construites, nous les projetons et obtenons deux empreintes circulaires dans l'espace écran. Le domaine paramétrique u à tesseller est ensuite matérialisé sous la forme d'un arc de cercle, qui sert de base pour calculer le facteur de tessellation à utiliser. Nous utilisons une erreur cordale, comme illustré sur

la Figure 3.1, à droite.

Pour les tores, le facteur de tessellation sur l'axe de révolution u est calculé en se référant à une boule, dont le centre est celui du tore, et dont le rayon correspond à la somme des rayons *mineur* et *majeur* (Figure 3.2, gauche). Une fois projeté à l'écran sous forme de cercle, nous considérons un arc de cercle dont la longueur est celle correspondant à l'étendue paramétrique sur u (Figure 3.2, centre).

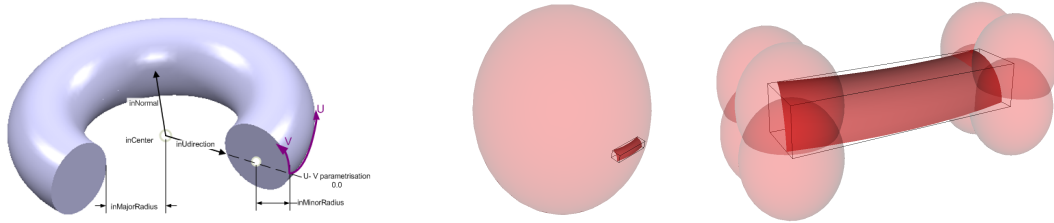


FIGURE 3.2 – *A gauche : définition d'un tore. Les rayons majeur et mineur sont identifiés. Au centre : boule de rayon $inMajorRadius + inMinorRadius$ servant de référence pour la tessellation sur la direction paramétrique u . A droite : boules dont la projection sert de référence pour le calcul du facteur de tessellation sur l'axe v .*

Pour l'axe v , nous considérons la boîte englobante du tore etinstancions 8 boules, à chaque coin de la boîte, chaque boule ayant un rayon égal au diamètre mineur du tore. Nous gardons la boule la plus proche et prenons sa projection (Figure 3.2, droite). Le domaine paramétrique v à tesseller est là encore matérialisé sous la forme d'un arc de cercle, qui sert de base pour calculer le facteur de tessellation à utiliser. Une erreur cordale est une nouvelle fois utilisée.

Pour les fillets, le facteur de tessellation sur l'axe paramétrique v , correspondant à la révolution du fillet sur son axe, est calculé avec la méthode de la boule, instanciée aux 8 coins de la boîte englobante du fillet, et ayant pour rayon le rayon maximal du fillet sur sa longueur. Ce dernier est précalculé. Le facteur de tessellation sur la longueur du fillet (axe paramétrique u) est calculé en utilisant ce que nous appelons la *hauteur* des polygones de contrôle de chaque courbe de Bézier constituant la colonne vertébrale du fillet (Figure 3.3).

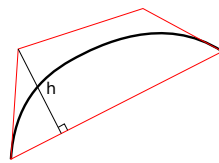


FIGURE 3.3 – *Hauteur h d'une courbe de Bézier cubique, dont la valeur dans l'espace objet et l'espace écran permet de calculer ϵ_s .*

Nous déduisons d'abord une déviation objet ϵ entre une courbe de Bézier et son approximation linéaire à partir de la déviation écran désirée ϵ_s , fixée par l'utilisateur. Pour ce faire, nous partons du principe que le rapport entre les déviations exprimées en espace objet et écran, d'une part, et les hauteurs des courbes de Bézier, d'autre part, est similaire. En d'autres termes, que

$$\frac{\epsilon}{\epsilon_s} = \frac{h}{h_s} \quad (3.1)$$

où h représente la *hauteur* en espace objet d'une courbe de Bézier cubique, et h_s la longueur de son empreinte à l'écran. En isolant ϵ , nous pouvons calculer le facteur de tessellation à appliquer

pour chaque courbe de Bézier dans le sens de la longueur du fillet avec la méthode de Filip et al. [FMM87](Partie II).

Pour les patches de Bézier bicubiques, nous reprenons la méthode utilisée pour la colonne vertébrale des fillets, mais l'appliquons pour un patch entier, h devenant la *hauteur* de patch de Bézier tout entier.

3.2 Classification

3.2.1 Classification de fragments

La classification de points pour les cellules complètement *sur la face* ou *hors de la face* est triviale, et s'effectue en utilisant la valeur de couverture propre à chaque cellule. La coordonnée paramétrique (u, v) associée au fragment est classifiée comme étant *sur la face* si une cellule ne comportant aucune courbe de découpe a une valeur de couverture égale à 1, et 0 dans le cas contraire.

3.2.1.1 Cellules avec une seule courbe quadratique

Dans une cellule traversée par une seule courbe définissant une découpe locale convexe, les coordonnées paramétriques (u, v) d'un point sont converties en coordonnées quadratiques (x, y) comme expliqué en Section 2.1.1.4. On teste ensuite si le point se trouve dans le polygone de contrôle. Il est classifié comme étant *hors de la face* si $y < 0$ ou $y - 2x + 1 < 0$, *sur la face* si $y - x \geq 0$; en dehors du polygone de contrôle, on classifie le point avec une évaluation implicite de la fonction f présentée en Section 2.1.1.4 (Figures 3.4 et 2.5(d)). Dans le cas concave, les classifications *sur la face* et *hors de la face* sont inversées.

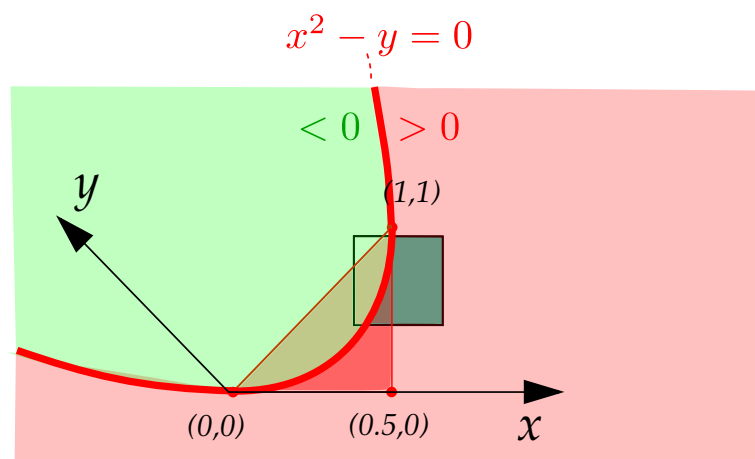


FIGURE 3.4 – Classification d'une coordonnée (u, v) dans une cellule avec une courbe quadratique. Le repère non-orthonormé sert à passer des coordonnées paramétriques à celles quadratiques.

3.2.1.2 Cellules avec deux courbes quadratiques

Pendant le rendu nous testons d'abord de quel côté de la ligne un point se situe (Figure 3.5), puis nous procédons à la classification de point décrite précédemment.

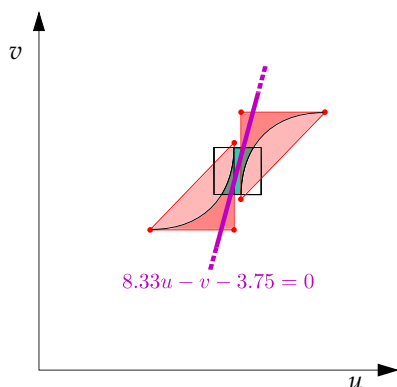


FIGURE 3.5 – Classification d’une coordonnée (u, v) dans une cellule avec deux courbes quadratiques, en classifiant d’abord le fragment avec la ligne séparatrice.

3.2.1.3 Mécanisme d’accès multirésolution

La valeur de couverture, binaire, détermine si une cellule occupe une zone de l’espace paramétrique majoritairement *sur la face* ou *hors de la face*. Les cellules feuilles qui sont complètement *sur* ou *hors* de la face sont identifiées dans le quadtree par une valeur de couverture égale à 1 ou 0, respectivement, sans référence à des courbes quadratiques. Pendant le rendu, la traversée du quadtree est arrêtée quand un nœud recouvre moins d’un pixel à l’écran. Lorsqu’une feuille de l’arbre est atteinte et que son empreinte à l’écran recouvre plusieurs pixels, la classification de point est effectuée suivant la procédure décrite précédemment. Cette méthode d’accès multirésolution nous permet d’augmenter les performances à l’affichage en limitant la profondeur d’accès au quadtree pour de très nombreux fragments (Figure 3.6). La découpe effectuée dans le fragment shader et l’utilisation du mot-clé *discard* rend moins efficace le mécanisme de culling précoce effectué par la carte graphique pendant la rasterisation, mécanisme qui permet de s’affranchir de l’exécution du fragment shader dans certaines situations. Limiter les calculs par fragment n’en est que plus souhaitable.

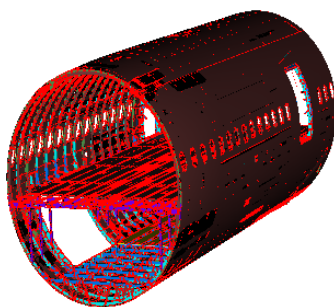


FIGURE 3.6 – Les fragments rouges tirent bénéfice de l’accès multirésolution. Ils identifient des fragments pour lesquels la traversée du quadtree s’est arrêtée à un nœud intermédiaire, ou à un nœud feuille mais où l’utilisation des courbes quadratiques n’a pas été nécessaire. Cette capture ne permet pas de voir les fragment cachés par d’autres qui leur sont superposés, la moindre face *B-Rep* donnant naissance à la rasterisation de plusieurs fragments, aussi petite soit-elle.

L’accès multirésolution permet également de limiter l’effet de moiré visuellement inélégant (Figure 3.7) sur les faces distantes.

Pour parcourir le quadtree et s’arrêter à une cellule dont l’empreinte écran couvre moins d’un

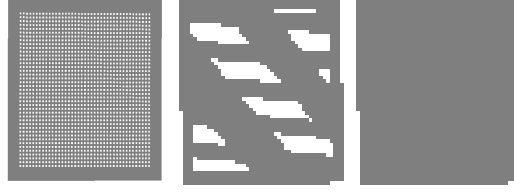


FIGURE 3.7 – *A gauche : exemple de face B-Rep avec de nombreux petits trous. Au centre : moiré visible à faible niveau de zoom. A droite : en utilisant l'accès multirésolution, les trous disparaissent, mais il n'y a pas de moiré.*

pixel, nous utilisons une méthodologie illustrée sur la Figure 3.8. Nous approximons d'abord l'empreinte du pixel dans l'espace paramétrique de la découpe en utilisant un parallélogramme P défini avec les deux vecteurs suivant :

$$q = \left(\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x} \right) \text{ et } r = \left(\frac{\partial u}{\partial y}, \frac{\partial v}{\partial y} \right)$$

Les dérivées partielles nous sont fournies par les instructions spécialisées $dFdx$ et $dFdy$ dans le fragment shader. Nous recherchons ensuite la longueur d'arête s du plus grand carré aligné sur les axes paramétriques et tenant dans P . Un tel carré peut être défini de la manière suivante :

- son centre C est situé à l'intersection des deux diagonales de P ,
- chaque diagonale a pour demie-longueur la longueur minimale des segments partants de C et de direction $(\pm 1, \pm 1)$, s'arrêtant à l'intersection avec P .

Considérons C comme étant le centre d'un repère orthonormé sur les axes paramétriques u et v . Les quatre points P_0, P_1, P_2 et P_3 correspondants aux extrémités de l'empreinte P ont, dans ce repère, les coordonnées suivantes :

$$\begin{aligned} P_0 &= -a - b \\ P_1 &= a - b \\ P_2 &= a + b \\ P_3 &= -a + b \end{aligned}$$

avec $a = .5q$ et $b = .5r$. Nous déduisons de cette définition les intersections entre les droites (P_0, P_1) , (P_1, P_2) , (P_2, P_3) et (P_3, P_0) , d'une part, et les deux diagonales partant de C et de direction $(\pm 1, \pm 1)$ de l'autre. Après simplification, nous obtenons les longueurs d'arêtes des cubes correspondant aux quatre intersections :

$$\begin{aligned} t_1 &= 2 \left| -a_x - b_x + a_x \frac{a_x + b_x - a_y - b_y}{a_x - a_y} \right| \\ t_2 &= 2 \left| -a_x - b_x + b_x \frac{a_x + b_x - a_y - b_y}{b_x - b_y} \right| \\ t_3 &= 2 \left| -a_x - b_x + a_x \frac{a_x + b_x + a_y + b_y}{a_x - a_y} \right| \\ t_4 &= 2 \left| -a_x - b_x + b_x \frac{a_x + b_x + a_y + b_y}{b_x - b_y} \right| \end{aligned}$$

Nous recherchons la longueur minimale, c'est-à-dire celle vérifiant

$$s = \min(t_1, t_2, t_3, t_4).$$

Le niveau du quadtree sur lequel nous devons nous arrêter est donné par la formule

$$l = \lceil \log_2(1/l_c) \rceil$$

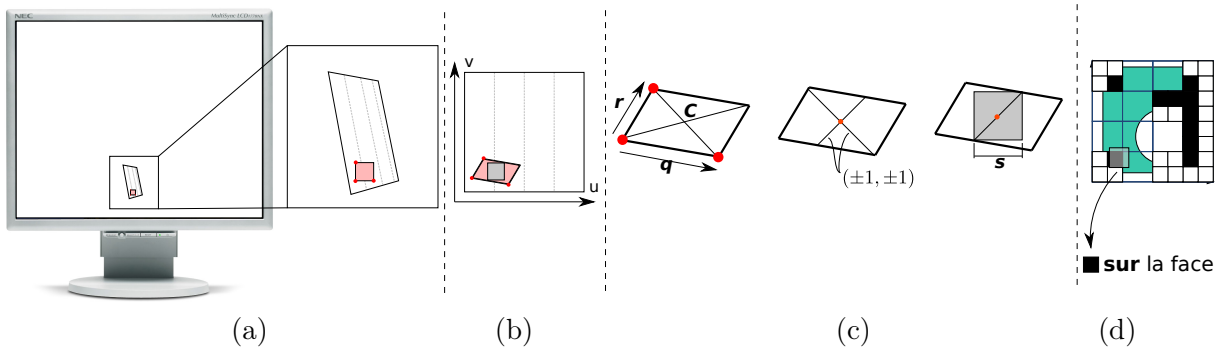


FIGURE 3.8 – (a) Empreinte d'un fragment dans l'espace écran. Le domaine paramétrique est déformé pour tenir compte de la projection et de la perspective de la vue. (b) Empreinte du même fragment dans l'espace paramétrique. Nous calculons la taille s de l'arête du carré le plus grand (en gris) qui tient dans l'empreinte pour limiter la profondeur de traversée du quadtree. (c) s peut être calculée en calculant les intersections entre les droites support des diagonales du carré et celles délimitant l'empreinte. (d) nous en déduisons le niveau maximum du quadtree au-delà duquel il n'est pas nécessaire d'aller chercher des informations pour effectuer la classification.

3.2.2 Classification de triangles

La structure B-Rep d'un modèle exporté en CAO permet d'avoir un indice sur la façon dont les différentes composantes ont été conçues. Des faces B-Rep de très grande taille, mais ne contenant que quelques petites zones éparses aussi bien *sur la face* que *hors de la face*, sont monnaie courante, en dépit de la taille parfois très grande des surfaces support. Ces faces sont enclines à une dégradation des performances puisqu'elles mènent au traitement d'un grand nombre de fragments, qui pourraient idéalement être classifiés par « blocs » de manière relativement triviale, aussi bien *sur la face* qu'*hors de la face*. Cf. le modèle d'avion de la Figure 3.9.

Lorsque les faces B-Rep sont proches de la caméra, les performances peuvent diminuer d'une manière spectaculaire puisque la classification de point doit être faite pour un nombre très élevé de fragments ; même les petites surfaces génèrent un grand nombre de fragments lorsqu'elles sont fortement zoomées.

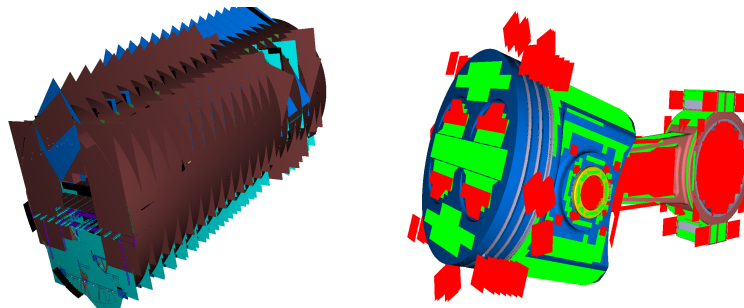


FIGURE 3.9 – A gauche : modèle d'avion rendu sans la découpe des faces activée, révélant de très nombreuses surfaces planes dans le sens transversal du fuselage. Le fuselage a beaucoup de sections circulaires qui ont été conçues avec un seul objet dans le modèle CAO. A droite : les zones vertes et rouges dans le modèle du piston correspondent à des triangles pour lesquels la classification a été réalisée avec succès. Les triangles rouges sont éliminés dans le shader géométrique, avant que la rasterisation ne prenne place.

Le coût de la classification de point dans notre méthode vient principalement de la traversée du quadtree, un coût qui est proportionnel à la profondeur de la cellule à utiliser. La classification avec les courbes est relativement rapide, puisqu'elle utilise des évaluations implicites. Pour réduire la quantité de calculs et d'accès mémoire à effectuer, nous proposons de grouper la classification de fragments et de faire la classification des triangles issus de la tessellation dynamique. Pour les triangles que nous ne pouvons pas classifier d'un bloc, dans leur intégralité, nous accélérons le temps de classification des fragments sous-jacents.

Les surfaces support sont tessellées sur leur domaine paramétrique u, v pendant le rendu, en respectant une erreur écran. Pour chaque surface, nous prédéterminons une profondeur (ou *niveau*) de quadtree k_i à laquelle nous savons que nous pourrions classifier pendant le rendu au moins 40% du domaine paramétrique quand ce dernier est couvert de triangles issus de la tessellation, avec des facteurs valant $t_u = 2^{k_i}$, $t_v = 2^{k_i}$. Pendant le rendu, nous voulons que les triangles tessellés respectent à la fois l'erreur écran que nous nous sommes fixée, et qu'ils soient alignés sur les frontières des cellules du quadtree, au moins jusqu'au niveau k_i , garantissant alors qu'au moins 40% d'entre eux seront classifiés « d'un bloc » pendant le rendu. Nous calculons donc pendant le rendu des facteurs de tessellation qui respectent l'erreur écran, puis nous ajustons ces facteurs à des puissances de deux supérieures, au moins égales à 2^{k_i} , ce qui nous donne des facteurs finaux $2^{k_{uf}}$, $2^{k_{vf}}$. La Figure 3.10 montre comment nous ajustons les facteurs de tessellation de sorte que suffisamment de triangles soient classifiés.

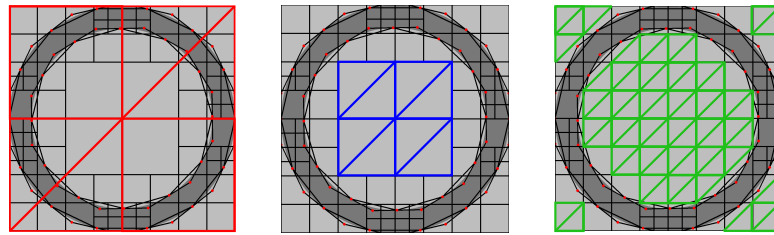


FIGURE 3.10 – Les triangles rouges alignés sur le niveau 1 du quadtree ne peuvent être classifiés, alors que les triangles de niveau 2 et 3 = k_i affichés en bleu et vert le peuvent, couvrant respectivement 25% et 50% du domaine paramétrique.

Pour chaque triangle tessellé, nous procédons à la classification avant que la rasterisation ne prenne place, comme montré sur les Figures 3.11 et 3.13.

La cellule d'appartenance du triangle à classifier est identifiée dans le quadtree jusqu'à un niveau de $k_f = \min(k_{uf}, k_{vf})$, une fois par triangle, en utilisant son barycentre. A ce stade, un triangle peut être classifié comme étant *sur* ou *hors* de la face si la cellule est une cellule feuille qui

- identifie une zone complètement *sur la face* ou *hors de la face*,
- référence une courbe quadratique, et le triangle réside du côté convexe de cette courbe,
- référence deux courbes quadratiques, et les points du triangle résident tous du même côté de la ligne de séparation, et du côté convexe de la courbe quadratique correspondante.

Autrement, lorsque le triangle ne peut être classifié, la référence au nœud du quadtree est passée au shader de fragment. La classification des fragments reprendra à partir de ce nœud, et non de la racine du quadtree, augmentant au passage les performances et rentabilisant le traitement effectué dans le shader géométrique, partagé pour de nombreux fragments.

Le gain de performance obtenu avec la classification de triangle doit être supérieur à la diminution de performance liée à la tessellation additionnelle effectuée, une tessellation qui serait nor-

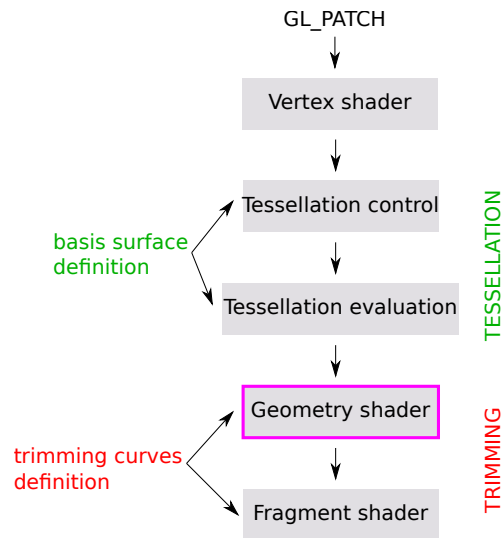


FIGURE 3.11 – Notre pipeline de rendu. La classification est effectuée dans le shader géométrique, qui est situé à l'emplacement idéal pour exercer les fonctions qui lui sont assignées.

malement effectuée avec des facteurs respectant seulement l'erreur écran que nous nous sommes fixée. Surtesseller une surface n'est pas forcément intéressant, comme en atteste la Figure 3.12.

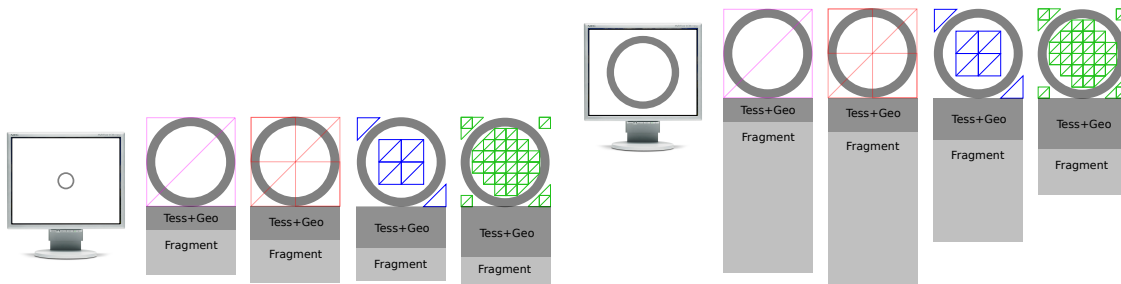


FIGURE 3.12 – A gauche : à faible niveau de zoom, une simple tessellation de la surface support en 2 triangles assure un rendu d'une rapidité optimale. Toute surtessellation est pénalisante pour les performances. A droite : à fort niveau de zoom, cette face peut idéalement être tessellée en 128 triangles (en vert, complètement à droite).

La classification de triangle est proportionnellement intéressante à mesure que le nombre moyen de fragments par triangle augmente. En moyenne, le coût de la classification de n fragments sur un triangle T , considéré comme un ensemble de fragments rastérisés de l'espace écran, doit être inférieur lorsque la tessellation a été réalisée, que lorsqu'elle n'a pas été faite, c'est-à-dire lorsque

$$C^{T(k_{uf},k_{vf})} + \sum_{i=1}^n C_f^{T(k_{uf},k_{vf})}(i) < \sum_{i=1}^n C_f(i)$$

où $C^{T(k_{uf},k_{vf})}$ représente le coût de la classification de triangle, C_f le coût de la classification des fragments contenus dans l'espace écran de ce triangle et $C_f^{T(k_{uf},k_{vf})}$ le coût de cette même classification en prenant en compte les calculs supplémentaires effectués pour le triangle, réutilisés pour chaque fragment du triangle.

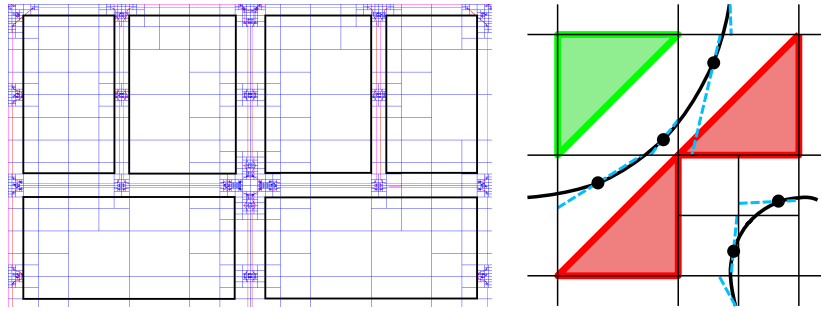


FIGURE 3.13 – *A gauche : surface plane avec de grandes zones hors de la face affichées en noir. A droite : classification de triangle. La classification des triangles rouges échoue puisque leurs sommets sont soit des deux côtés de la ligne séparatrice de la cellule (à droite), ou du côté concave de la courbe (en bas). Les triangles verts pour qui la classification a pu être effectuée seront soit rapidement classifiés en tant que zones sur la face, ou éliminés par le shader géométrique lorsqu'ils sont hors de la face.*

Nous avons expérimenté la classification de triangle avec des facteurs de tessellation uniformes $t_u = t_v$ pour des surfaces planes, étant donnée leur très grande quantité dans les modèles CAO (cf. Chapitre 4, Table 4.1), vu que le facteur de tessellation n'a pas d'impact sur l'erreur écran et parce que nous pouvons calculer le nombre de fragments tôt dans le pipeline de rendu pour ces surfaces (eg. shader de contrôle des facteurs de tessellation, Partie I, Chapitre 3). Nous activons notre méthode lorsque les triangles couvrent en moyenne, pour une primitive, au moins 200 pixels à l'écran. C'est assez pour augmenter considérablement les performances dans le cas où les surfaces à afficher sont modérément ou fortement zoomées, et sans les réduire pour les surfaces éloignées du fait d'une sur-tessellation superflue.

Chapitre 4

Résultats et limitations

4.1 Modèles utilisés

Nous utilisons trois modèles CATIA V5 pour nos tests, un piston, un satellite et une section d'avion. Les propriétés de ces modèles sont détaillées dans les Tables 4.1 et 4.2.

Type de surface support	Satellite	Avion	Piston
Plans	36518	276788	210
Cylindres	41832	295262	390
Cônes	9722	20488	26
Sphères	667	1595	24
Tores	6517	34266	66
Fillets	166	22125	74
Nurbs	853	20871	0
Autres types	173	30669	0
Total	96448	702064	790
Patchs de Bézier cubiques	1271	71894	0
Nombre total de patchs	96693	722418	790

TABLE 4.1 – Nombre de surfaces support classifiées par type de surface. La ligne Bézier cubiques compte le nombre de patchs de Bézier créés après le processus d'approximation des surfaces support originales. Toute surface support qui n'est pas d'un type explicitement pris en charge (plan, cône, cylindre, tore, sphère ou fillet) est approchée par un ou plusieurs patch(s) de Bézier cubique(s). Sinon, à chaque surface support correspond un patch sur le GPU.

4.2 Qualité visuelle

Notre méthode de rendu fournit une précision visuelle élevée dans les plans rapprochés. La tessellation statique affiche à fort niveau de zoom des arêtes droites et des points discrétisés quand elle est conjointement utilisée pour l'affichage de grands modèles (imposant une limite sur la finesse de discrétisation en raison de l'occupation mémoire) et pour visualiser des détails d'une grande finesse. Notre approche permet d'afficher des courbes de découpe d'une manière lisse (Figure 4.1).

Type de courbe	Courbes de découpe		
	Satellite	Avion	Piston
Ligne	354197	2767813	2680
Arc d'ellipse	58797	438295	294
Nurbs (total)	58236	496808	606
Nurbs de degré 1	12479	2251	0
Nurbs de degrés 2,3,4	98	407	8
Nurbs de degrés 5	45571	493723	598
Nurbs de degrés 6,7,8,11	88	427	0
Nombre total de courbes	471230	3702916	4186
Quadratiques (0.1mm)	690140	5975794	5695
Quadratiques (0.01mm)	941476	8661440	7854

TABLE 4.2 – Nombre de courbes de découpe classifiées par type de courbe. Les deux dernières lignes listent le nombre de courbes obtenues après l'approximation effectuée en prétraitement.

4.3 Performances

La Table 4.3 montre les résultats des tests de performance obtenus pour les modèles et vues de la Figure 4.2 ainsi que de la Figure 1.3 du chapitre précédent, en comparant notre méthode de découpe et celle de Schollmeyer et Fröhlich. La tessellation est effectuée avec une bibliothèque commerciale, à partir des modèles CATIA (Datakit). Dans ces tests, nous fixons la tolérance d'approximation à 0.1 mm pour les surfaces de Bézier et pour les courbes support des fillets. Nous configurons l'approximation des courbes de découpe de sorte que ces dernières ne dévient pas de plus de 0.1 ou 0.01 mm, en espace objet, des courbes d'origine. Les résultats sont obtenus avec une carte d'entrée/moyenne gamme (hélas courantes dans l'industrie) de type GeForce GTS 450 avec 1 Go de mémoire vidéo, sur un processeur Intel Core i7-860 avec 8 Go de RAM. Notre méthode permet d'obtenir un gain de performance allant de 10% à 240% suivant le modèle et la vue utilisés.

4.4 Occupation mémoire

Les modèles statiquement tessellés occupent un espace mémoire qui augmente proportionnellement avec les facteurs de discrétisation. Pour les données qui servent à définir les surfaces support, l'occupation mémoire de la méthode proposée est affectée par la tolérance à l'erreur utilisée pour approximer les surfaces complexes avec des patches de Bézier cubiques, et les courbes support pour les fillets, approximées avec des courbes de Bézier cubiques. L'occupation mémoire de la structure de découpe est affectée par l'erreur d'approximation utilisée pour les courbes de découpe. La Table 4.4 détaille l'occupation mémoire pour notre méthode et celle de Schollmeyer et Fröhlich, ainsi que pour des modèles statiquement tessellés. Avec une erreur d'approximation des courbes de découpe de $\epsilon = 0.1$ mm, nous consommons 30-45% moins de mémoire que Schollmeyer et al. Avec $\epsilon = 0.01$ mm, nous obtenons à peu près la même occupation.

4.5 Limitations

Dans certaines circonstances rares, le processus de subdivision des cellules peut dégénérer quand trois courbes quadratiques ou plus sont très rapprochées, ce qui se produit principalement

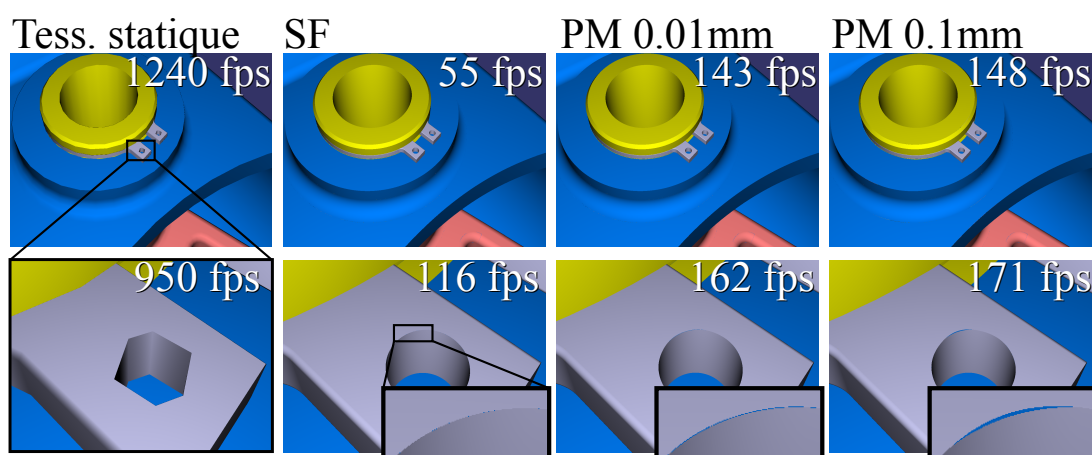


FIGURE 4.1 – Examen de la qualité du rendu pour le modèle de piston. Tess. statique : tessellation statique de 0.1mm, SF : Schollmeyer et Fröhlich, PM : méthode proposée avec $\epsilon = 0.01\text{mm}$ ou 0.1mm. Rendu effectué en tessellation dynamique avec une erreur d'écran de $\epsilon_s = 0.1$. Temps de rendu fourni de manière indicative, exprimé en images par seconde (fps, de l'anglais frames per second).

lorsqu'une extrémité de courbe quadratique appartenant à une première boucle de découpe est très proche d'une seconde boucle (Figure 4.3, à gauche). Dans ce cas, la subdivision doit se faire très profondément dans le quadtree puisque l'extrémité de la courbe quadratique crée une situation multi-courbe que nous ne pouvons prendre en charge dans notre modèle de cellule, limité à deux courbes. Cette situation se produit le plus souvent parce que nous ne contrôlons pas les endroits où des points de jonction sont créés entre les courbes quadratiques, pendant le processus d'approximation. La Table 4.5 montre que cette situation est rare cependant, même avec le modèle de section d'avion, contenant un très grand nombre de faces.

Une solution envisageable pour ce problème serait de décaler les points de contrôle des courbes quadratiques le long des courbes de découpe de sorte que ces derniers s'éloignent des boucles gênant la subdivision. Il est également possible de générer une nouvelle courbe quadratique, juste pour le segment traversant une cellule. Enfin, les cellules problématiques pourraient être subdivisées d'une manière différente qu'une simple subdivision 2×2 uniforme ; toutefois cette approche rendrait plus complexe le prétraitement, comme le code utilisé pour le rendu.

La méthode que nous utilisons pour calculer la ligne séparatrice (Section 3.2.1.2 du chapitre précédent) peut prendre un certain temps avant de converger vers une solution. Quand les segments support sont trop rapprochés, la subdivision des polygones de contrôle doit dans certains cas être assez poussée avant qu'un segment ne puisse être utilisé pour construire la ligne séparatrice (Figure 4.4 à droite). Rappelons que si aucune ligne séparatrice ne peut être trouvée après 100 itérations, nous en déduisons qu'aucune ligne séparatrice n'existe dans l'absolu pour séparer les deux sections de courbes et nous subdivisons la cellule.

	Temps de rendu (ms)					
	(a)	(b)	(a)	(b)	(a)	(b)
Avion	Vue 1		Vue 2		Vue 3	
SF	145	130	115	101	50	56
P.M. 0.1mm	108	98	103	92	38	37
P.M. 0.01mm	109	99	93	92	40	37
Satellite	Vue 1		Vue 2		Vue 3	
SF	44	28	45	31	87	22
P.M. 0.1mm	34	21	30	21	65	14
P.M. 0.01mm	33	20	30	19	65	15
Piston	Vue 1		Vue 2		Vue 3	
SF	15	8.3	31	15	53	25
P.M. 0.1mm	8.8	3.1	19	4.5	31	8
P.M. 0.01mm	8.9	3.1	19	4.6	33	8

TABLE 4.3 – Temps de rendu en millisecondes pour notre méthode (P.M.) et celle de Schollmeyer et Fröhlich (SF). Les vues utilisées sont celles montrées sur la Figure 1.3 du chapitre précédent, ainsi que sur la Figure 4.2. Approximation de surface $\epsilon = 0.1\text{mm}$; Erreur de tessellation écran $\epsilon_s = 5$ pixels. Résultats présentés aussi bien en lancer de rayon + tessellation (a), en utilisant la méthode de Toledo et Lévy [TL08] pour les sphères, cônes et cylindres, ou en tessellation seulement (b). Notons que le lancer de rayon désactive totalement le culling précoce des fragments sur le GPU et peut expliquer la chute parfois importante de performances pour (a).

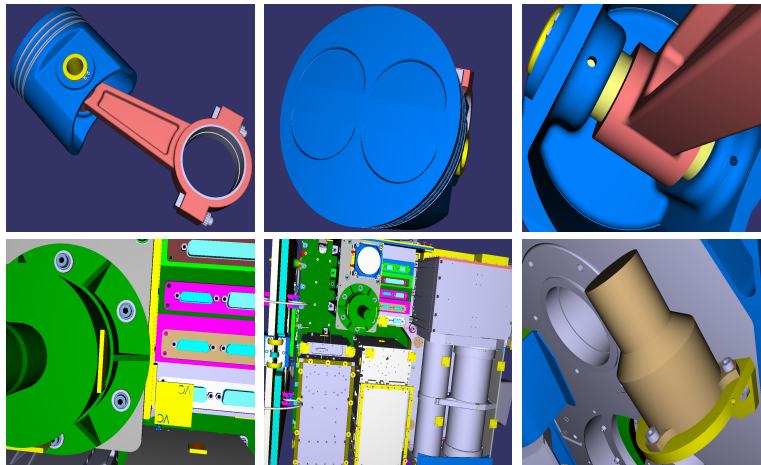


FIGURE 4.2 – Vues 1, 2 et 3 utilisées pour l'évaluation de la performance de rendu pour le modèle de Piston (en haut) et celui du Satellite (en bas). Résultat des tests de performance montrés dans la Table 4.3.

Section d'avion	Surface	Découpe	Total
SF	64	296	360
P.M. 0.1mm	64	197	261
P.M. 0.01mm	64	294	358
Discrétisation 0.1mm			246
Discrétisation 0.01mm			1010
Satellite	Surface	Découpe	Total
SF	10.5	48.1	58.6
P.M. 0.1mm	10.5	31	41.5
P.M. 0.01mm	10.5	43.7	54.2
Discrétisation 0.1mm			42.5
Discrétisation 0.01mm			139.3
Piston	Surface	Découpe	Total
SF	0.119	0.656	0.775
P.M. 0.1mm	0.119	0.309	0.428
P.M. 0.01mm	0.119	0.440	0.559
Discrétisation 0.1mm			0.629
Discrétisation 0.01mm			2.683

TABLE 4.4 – Occupation mémoire, en mégaoctets, pour la méthode de Schollmeyer et Fröhlich (SF), pour la méthode proposée (P.M.), et pour des modèles statiquement tessellés (pour lesquels chaque sommet prend 36 octets, en prenant en compte l'espace requis pour les coordonnées des sommets, leur normale et couleur). L'approximation des surfaces en patches de Bézier cubiques, effectuée pour la prise en charge de surfaces complexes, utilise ici dans tous les cas une tolérance de 0.1 mm.

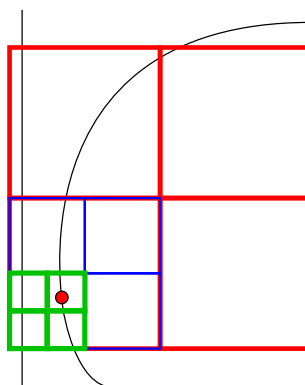


FIGURE 4.3 – Le cercle rouge identifie un point de jonction entre deux courbes quadratiques. La subdivision doit parfois être effectuée en profondeur avant d'obtenir des cellules respectant nos contraintes (2 courbes quadratiques au maximum par cellule).

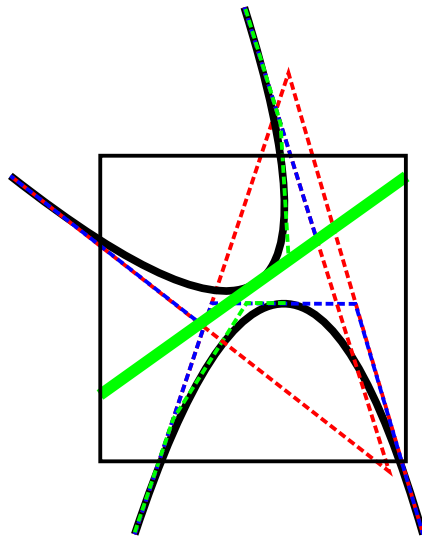


FIGURE 4.4 – Un nœud du quadtree référencant deux courbes. Une ligne séparatrice peut ici être trouvée après 3 subdivisions d'un des deux polygones de contrôle. Dans certains cas, le nombre de subdivision requis est très grand.

Section d'avion	0.1mm	0.01mm
Nombre de patches	365	668
% du nombre de patches total	0.12	0.22
Nœuds par patch (moyenne)	2.5	4.4
Satellite		
Number of patches	147	333
% du nombre de patches total	0.23	0.5
Nœuds par patch (moyenne)	1.2	3
Piston		
Number of patches	0	0
% du nombre de patches total	0	0
Nœuds par patch (moyenne)	0	0

TABLE 4.5 – Nombre de patches pour lesquels la profondeur de subdivision du quadtree a atteint le niveau 13, entraînant un abandon (Voir Figure 4.3 à gauche). La table montre aussi le nombre moyen de nœud problématique trouvé dans une structure de découpe où au moins un problème (eg. un épuisement de la limite de profondeur fixée) a été détecté. 1.2 signifie que, en moyenne, une structure de découpe problématique possède 1.2 nœud problématique.

Chapitre 5

Conclusion

Nous avons présenté une structure de découpe optimisée pour les performances permettant d'afficher de grands modèles B-Rep d'une manière interactive. Notre implémentation avec des shaders dédiés pour chaque type de surface support nous permet d'afficher une section d'avion entière avec une haute qualité de rendu, et un taux de rafraîchissement de l'image correct.

La nature multirésolution de notre structure de découpe basée sur un quadtree et la présence de la valeur de couverture dans chaque cellule nous permet de faire la classification de coordonnées u, v sur la surface support en limitant autant que possible la profondeur d'accès aux données lorsque les surfaces à afficher sont éloignées du point de vue. Les performances du rendu s'en trouvent globalement améliorées dans la mesure où les surfaces distantes sont généralement majoritaires dans la vue et que le nombre de fragments (cachés ou non) à traiter devient vite important. L'utilisation du mot-clé *discard* désactivant la rasterisation d'un fragment a tendance à limiter l'efficacité du culling précoce sur Z pour les fragments et donc à accroître la nécessité d'accélérer leur classification associée.

Notre structure permet en outre la classification rapide des triangles issus de la tessellation dynamique lorsque ceux-ci sont contenus ou alignés sur les cellules de notre quadtree. Il en résulte une classification globalement accélérée pour les surfaces support. Notre classification peut facilement s'intégrer dans le shader géométrique pour classifier à la volée les différentes primitives et offre la possibilité de ne pas les envoyer au rasteriseur (*discard* de triangle) si la classification est hors de la zone de découpe. Nous avons expérimenté la classification de triangles avec les surfaces planes, généralement très nombreuses dans une scène, et obtenons un gain de performance significatif par rapport à une classification effectuée avec les seuls fragments.

Notre choix d'approximer les courbes de découpe et les surfaces support avec une erreur contrôlable par l'utilisateur nous a permis de nous détacher de l'état de l'art en terme de performances, en exacerbant de manière relativement marginale les jours artificiellement créés par la tessellation entre les faces. La complexité de la transformation du modèle imposée par cette approximation n'est pas un problème dans la mesure où notre recherche portait sur une solution de rendu *après-coup* d'exports B-Rep représentant des assemblages finaux de milliers de pièces, effectués de manière ponctuelle.

Quatrième partie

Rendu sans crack de modèles tessellés
dynamiquement

Chapitre 1

Introduction

1.1 Limitations des méthodes de l'état de l'art pour le rendu haute qualité

Nous avons résumé dans la Partie II les principales méthodes existantes pour réaliser le rendu des modèles B-Rep. La tessellation individuelle des faces B-Rep ne pouvant être réalisée en temps-réel, les méthodes de rendu basées découpe sont devenues populaires sur GPU. Elles ouvrent la porte à un rendu très précis, où il est possible de s'affranchir de toute discrétisation aussi bien lors de la prise en charge des surfaces support que pour réaliser la découpe. Des structures vectorielles adaptées au GPU existent ainsi, et permettent de réaliser une découpe en temps-réel. Citons les structures de Schollmeyer et Fröhlich [SF09] ou celle Nishita et al. [NSK90] (Partie II), et celle présentée dans la Partie III.

Un rendu de haute qualité passe obligatoirement par une prise en charge précise des surfaces support. Nous avons détaillé dans la Partie II, Chapitre 3 les différentes manières de faire le rendu de ces surfaces. Le rendu par lancer de rayon offre une très grande précision mais a plusieurs faiblesses. Une lenteur d'exécution tout d'abord, réellement prohibitive pour les grands modèles, omniprésente quel que soit l'algorithme utilisé. Elle est due à la recherche itérative d'une racine pour chaque fragment rendu dans le cas de la méthode de Newton, ou de la réduction du domaine paramétrique du patch jusqu'à une taille suffisamment petite dans le cas de la méthode du Bézier clipping. Le lancer de rayon ne permet pas de tirer parti des fonctions dédiées à la rasterisation et à la tessellation matérielle des GPUs, ce qui est peu avantageux. Il oblige à spécifier pour chaque fragment une profondeur de vue associée, ce qui a pour effet de désactiver le culling précoce des GPUs. Une qualité de rendu réellement optimale et dénuée d'artefacts ne peut être réalisée en pratique qu'en utilisant la méthode du Bézier clipping. La méthode de Newton, même si elle est efficace, peut souffrir de problèmes de convergence et provoquer l'apparition d'artefacts visuels. Pour s'affranchir des problèmes de convergence, les étapes de subdivision de surface décrites par Toth [Tot85] peuvent être utilisées mais elles ralentissent encore (considérablement) le rendu de chaque fragment.

Les méthodes basées sur la tessellation dynamique uniforme montrent d'excellentes performances comme le prouve le travail de Yeo et al. [YBP12]. La qualité de rendu est très proche de celle obtenue en lancer de rayon pour peu que la tolérance de discrétisation soit suffisamment petite. Pour autant, et c'est le cas dès lors que la tessellation est utilisée, des cracks apparaissent entre les faces (Partie III, Figure 1.2). Quand bien même les facteurs de tessellation peuvent être augmentés pour limiter leur apparition, ces cracks restent toujours présents et sont d'autant plus présents que les patches à tesseller sont nombreux. C'est notamment le cas lorsque les surfaces

NURBS sont décomposées en patches de Bézier multiples. Pour les supprimer, Yeo et al. propose une méthode pour le rendu de surfaces non découpées, basée sur un facteur de tessellation affecté aux bordures des surfaces, inapplicable pour nos modèles B-Rep. Les méthodes de Pavanaskar et al. [PM13] et Balázs et al. [BGK04] peuvent être utilisées mais montrent trop vite leurs limites. Basées sur une discrétisation et sur la génération de primitives géométriques venant combler explicitement les cracks laissés par la tessellation au moment du rendu, ces méthodes semblent trop à même de produire des artéfacts visuels.

Nous proposons une méthode pour s'affranchir des cracks de tessellation d'une manière plus fiable, sans artéfacts visuels, et sans nécessiter la création de primitives géométriques dédiées au remplissage. Une création de géométrie additionnelle va pour nous à contre courant de la vocation même du GPU, qui est de pouvoir afficher un modèle de données d'une manière la plus brute possible, si possible sans transformation trop importante des données (Partie I, Section 1.5.1). Nous détaillons également dans cette partie une méthode simple pour éliminer les cracks d'une manière plus approximative mais également plus rapide, qui peut être utilisée lors des déplacements de caméra si les performances de la première méthode sont jugées insuffisantes, par exemple lors de l'affichage des plus grands modèles.

1.2 Vue d'ensemble de la méthode proposée

Nous proposons un algorithme [CBV*14] pour réaliser un rendu interactif de haute qualité, dénué de crack, et suffisamment performant pour permettre la prise en charge de grands modèles. Notre algorithme est divisé en plusieurs étapes (Figure 2.1 du chapitre suivant).

Une première étape effectue un rendu initial du modèle en utilisant une tessellation dynamique des surfaces support et une découpe directe sur GPU, en utilisant une tolérance permettant de contrôler précisément la taille des cracks induits par la tessellation (Section 2.1).

Après le rendu initial suit une étape de détection des cracks (Section 2.2), au cours de laquelle des pixels sont sélectionnés pour un éventuel remplissage ultérieur.

Les étapes suivantes réalisent le remplissage et dépendent de la manière de le réaliser. Nous proposons deux méthodes. Pendant les déplacements de caméra où les temps de rendu doivent être réduits, nous proposons une procédure de remplissage de crack simple, en une seule étape, travaillant dans l'espace image et basée sur l'information de géométrie et de couleur au voisinage des pixels identifiés comme cracks (Section 2.3). Cette méthode de remplissage peut laisser apparaître des artéfacts mineurs. Pour les rendus statiques, ou si une plus grande qualité de rendu est désirée pendant les déplacements de caméra, nous proposons également une méthode de remplissage plus robuste, en trois étapes, opérant en espace objet. Avec cette méthode, une troisième étape (Section 2.4.1) crée un masque de profondeur pour limiter le travail effectué par le GPU dans une quatrième étape (Section 2.4.2), cette dernière réalisant le remplissage à proprement parler en lançant des rayons sur le modèle à l'emplacement des cracks, et considérant uniquement les pixels filtrés ainsi que les surfaces dignes d'intérêt pour leur remplissage.

Nous tirons parti de la tessellation et la rastérisation rapides sur GPU pour limiter la quantité de rayons lancés sur le modèle, diminuant l'impact sur les performances, tout en s'assurant que les cracks vont être effectivement remplis. Une étape terminale (Section 2.4.3) combine pour finir les informations de couleur et de profondeur obtenues à la première (rendu initial) et la quatrième (remplissage) étapes et produit l'image finale.

Chapitre 2

Algorithme

Avant d'être passé à notre moteur de rendu, le modèle d'entrée doit être converti. Cette conversion, faite une fois pour toute dans un prétraitement avec le CPU, est une transformation du modèle d'entrée, réalisée sans perte de données. Chaque surface support du modèle B-Rep est convertie en NURBS, qui est à son tour décomposée en un ou plusieurs patches de Bézier rationnels [CLR80]. Les courbes de découpe subissent le même traitement. Nous construisons la structure de découpe de Schollmeyer et Fröhlich [SF09], qui nous servira à réaliser la classification de chaque fragment sur le GPU, chaque fois que cela sera nécessaire.

Une vue d'ensemble de notre pipeline de rendu tournant intégralement sur le GPU ainsi que les différents buffers de données utilisés est montré sur la Figure 2.1. La Figure 2.2 illustre le résultat des opérations de détection et de remplissage de cracks. Les différentes étapes de rendu sont décrites ci-après.

2.1 Etape 1 : rendu initial

Dans la première étape, le modèle est une première fois rendu en utilisant le pipeline graphique programmable. Une tessellation dynamique des surfaces de base est réalisée en utilisant la méthode de Yeo et al. [YBP12]. Avec leur méthode, nos patches de Bézier rationnels sont tessellés et rastérisés de telle sorte que les primitives tessellées ne dévient jamais plus de k pixels par rapport à l'empreinte projetée à l'écran de leur représentation analytique. Yeo et al. appellent ce concept une précision de *couverture* de k pixels. De même, une précision *paramétrique* de k pixels garantit que les fragments d'une coordonnée u, v spécifique ne soient jamais rastérisés à une distance de plus de k pixels de l'emplacement correspondant sur l'empreinte écran de la véritable surface analytique, non approximée (Figure 2.3). Nous définissons $k = 0.5$ pour être sûrs que les fragments une fois projetés ne soient jamais rastérisés à une distance supérieure à 0.5 pixel à rapport à l'emplacement écran de la projection de leur coordonnée u, v sur la surface analytique. k peut avoir une valeur inférieure ce qui amoindrit le nombre de cracks à l'issue de l'étape 1. Changer la valeur de k est discuté en Section 3.1.

Chaque fragment traité par le GPU est classifié [SF09], et les fragments dont la coordonnée u, v est hors de la zone de découpe sont rejetés. Pour les fragments dans la zone de découpe, outre la couleur issue des calculs d'éclairage, nous stockons dans un buffer de métadonnées l'identifiant du patch de Bézier, celui de la face B-Rep associée, ainsi que la coordonnée u, v du fragment.

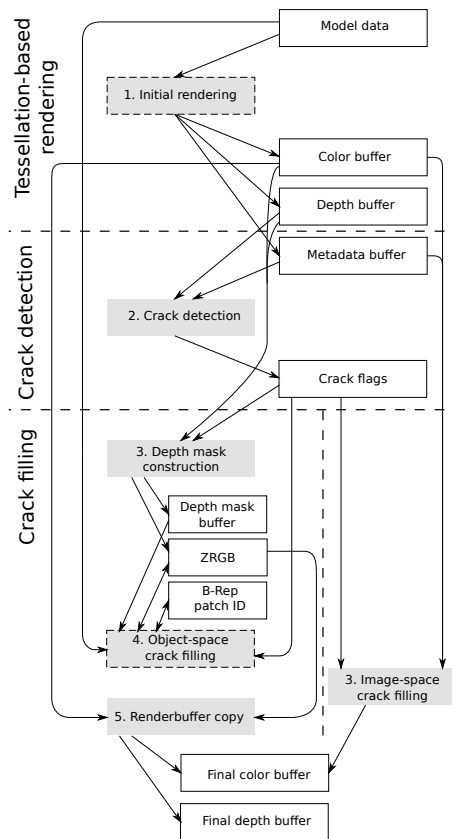


FIGURE 2.1 – Vue d’ensemble de notre pipeline de rendu. Les boîtes avec un fond gris représentent des étapes de rendu. Remarquons sur cette figure que les étapes liées au remplissage sont numérotées de manière commune pour la méthode en espace objet (étapes 3, 4 et 5 à gauche) et celle en espace écran (étape 3, à droite). Les boîtes avec un contour en pointillé effectuent un rendu intégral du modèle, tandis que les autres opèrent en espace écran. Les boîtes avec un fond blanc représentent des buffers de données. Les flèches montrent les données utilisées par les différentes étapes de rendu, leur sens indique leur mode d’utilisation — en lecteur et/ou en écriture.

2.2 Etape 2 : détection des cracks

L’étape 2 travaille en espace écran. Pour chaque pixel du framebuffer, nous identifions un pixel comme étant un crack ou non en se basant sur les informations du premier anneau de voisinage, constitué de 8 pixels. Le résultat de la détection de crack est inscrit dans un buffer booléen dénommé *crack flag buffer* (Figure 2.1), où une valeur *vraie* signifie que le pixel correspondant est un crack.

Le pixel central p est flaggé (i.e. considéré comme étant un crack) lorsqu’un pixel voisin p_i et son opposé $p_{opp(i)}$ sur l’anneau de voisinage sont plus proches de la caméra par un facteur d’au moins d_{min} , où d_{min} est une valeur exprimée en espace objet (encart en haut à droite de la Figure 2.6). L’information de profondeur est récupérée pour chaque pixel à partir du buffer de profondeur créé lors de la première étape. d_{min} doit être égal à la distance minimale entre deux points sur la surface du modèle, se trouvant sur un même rayon (o, \vec{R}) , et où la normale aux deux points de la surface fait dans les deux cas face à la caméra (produit scalaire entre le rayon

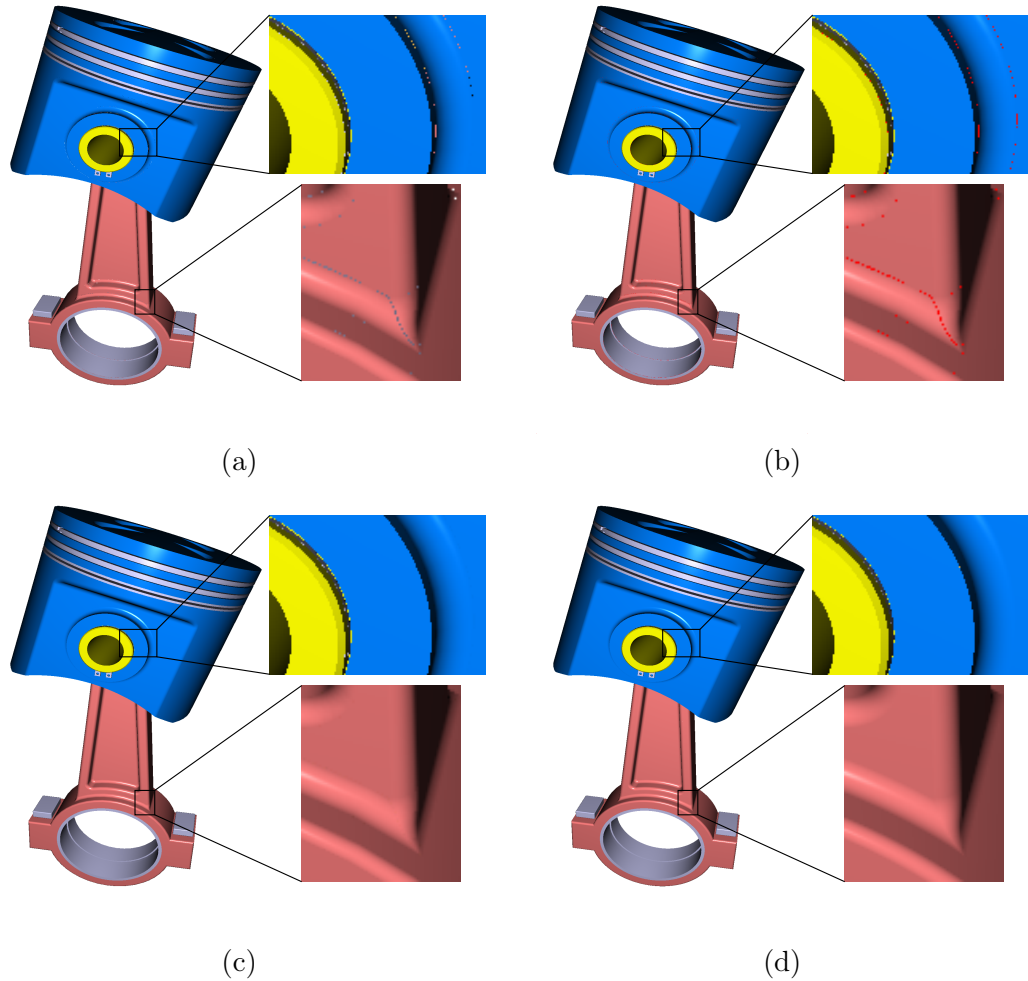


FIGURE 2.2 – (a) résultat de la première étape après tessellation dynamique, avec des cracks entre les faces. (b) identification des cracks (en rouge) de l'étape 2. (c) remplissage des cracks de l'étape 4 en utilisant notre méthode en espace objet, basée sur le lancer de rayon. (d) rendu de référence en utilisant un lancer de rayon sur toute la scène.

et \vec{N}_1 et \vec{N}_2 dans les deux cas négatif, voir Figure 2.4).

Le précalcul de cette valeur est difficile à réaliser en pratique. Pour les modèles que nous avons testés, nous définissons $d_{min} = 1$, équivalent à une distance de 1 millimètre en espace objet. Cette distance réduite aboutit à une détection de crack certes conservative, mais permet de détecter tous les cracks de manière exhaustive, quels que soient la position et l'angle de vue de la caméra. Le pixel p n'est pas identifié comme étant un crack si un pixel voisin (p_i ou $p_{opp(i)}$) a le même identifiant de face B-Rep associé, suggérant que nous sommes dans une zone rasterisée de la même face mais marquée par une forte variation de profondeur, c'est-à-dire une forte perspective locale. Comparer p_i et $p_{opp(i)}$ à p , simultanément, est nécessaire pour que seuls les pixels identifiant réellement des cracks soient détectés, et non les pixels dans les zones de silhouette du modèle (Figure 2.5 et encart en bas à gauche de la Figure 2.6). Insistons sur le fait que, notre détection de crack étant conservative, des pixels n'étant pas visuellement assimilables à des cracks peuvent occasionnellement être identifiés comme tels.

Dans les deux prochaines sections, nous décrivons deux différentes manières de combler les

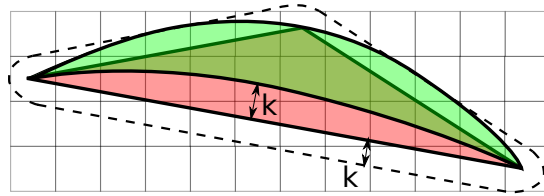


FIGURE 2.3 – Le triangle rouge est une approximation à k pixels de la section de la surface B-Rep (en vert) qu’il représente. La précision de couverture de k pixels définie par Yeo et al. garantit que l’empreinte de la surface B-Rep à l’écran se situe dans un voisinage de k pixels (en pointillé) autour du triangle.

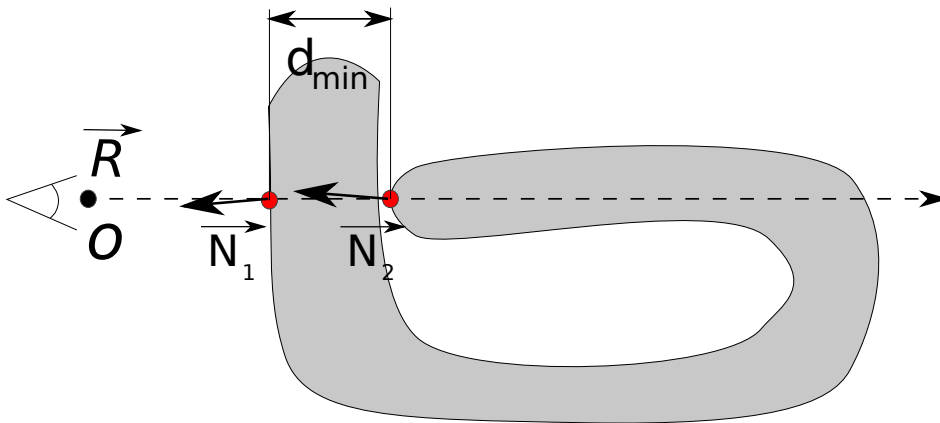


FIGURE 2.4 – Calcul de d_{min} . Cette valeur doit définir la distance minimale entre deux points sur le modèle visibles à l’emplacement de deux pixels adjacents à l’écran. Les modèles B-Rep identifient des objets physiques et seulement la partie externe des faces B-Rep est affichée. d_{min} peut être défini comme étant la distance minimale parcourue par un seul rayon (en pointillé) intersectant le modèle B-Rep et dont la normale aux points d’intersection fait face (en rouge) à la caméra (à gauche).

cracks. La méthode présentée dans la prochaine section travaille exclusivement en espace écran et cible le rendu pendant les déplacements de caméra. La méthode présentée par la suite opère en espace objet et bénéficie de la précision géométrique du lancer de rayon. Bien que plus lente, elle est suffisamment rapide pour être utilisée pendant la navigation dans les modèle et fournit une grande précision et fiabilité.

2.3 Remplissage en espace écran

Pour chaque pixel p du framebuffer, nous mettons à jour sa couleur en fonction des informations des 8 pixels voisins. Nous déterminons d’abord le patch de Bézier B_p qui couvre le plus grand nombre de pixels autour de p . Le pixel p reçoit ensuite la couleur moyenne des pixels associés à B_p .

Le remplissage de crack en espace écran n’utilise aucune information de nature géométrique au-delà de l’identifiant du patch. Il peut dans certains cas combler les jours géométriques (Partie 2, Section 2.2) entre les faces mais également les trous ou d’autres détails, lorsque ces derniers sont visualisés à une certaine distance. Nous discutons la qualité d’image en Section 3.1.

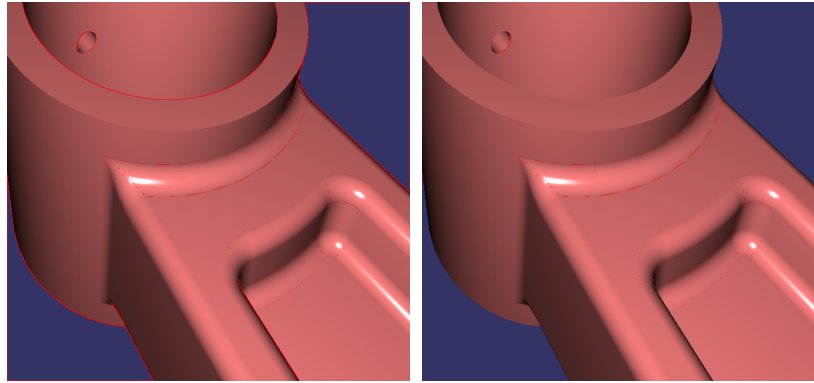


FIGURE 2.5 – Notre algorithme détecte les cracks (en rouge, à droite) en comparant la profondeur de pixels voisins. Le test du « fragment opposé » permet de s’assurer que les pixels le long des silhouettes ne sont pas détectés. Dans la capture d’écran à gauche, ce test a été désactivé, ce qui entraîne la détection de pixels le long des silhouettes, pixels étant considérés comme des cracks par la suite.

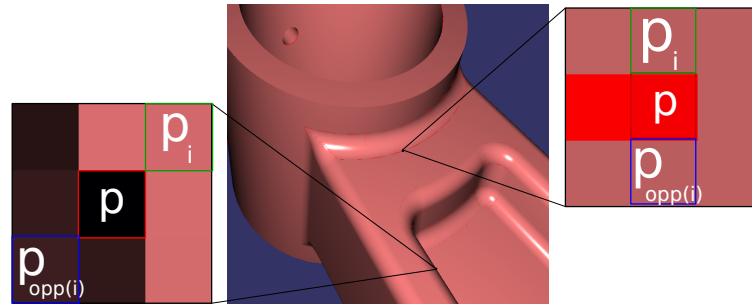


FIGURE 2.6 – Les quatre paires de pixels (p_i, p_{opp}) autour de p sont examinées. Encart en haut à droite : p est identifié comme crack, puisque les distances en espace objet entre les fragments associés à p et p_i d’une part, et p et p_{opp} de l’autre, sont toutes deux supérieures à d_{min} , que p_i et p_{opp} sont plus proches de la caméra que p , et que p n’a pas le même identifiant de face B-Rep que p_i et opp . Cette dernière condition n’est pas vérifiée dans l’encart se trouvant en bas à gauche, et p n’est alors pas détecté comme étant un crack.

2.4 Remplissage en espace object

Dans cette section, nous détaillons un algorithme qui remplit les cracks d’une manière robuste et fiable. Il fonctionne en détectant précisément quelle face B-Rep du modèle est visible à l’emplacement des cracks.

2.4.1 Etape 3 : construction du masque de profondeur

Dans le but de minimiser le nombre de fragments à traiter dans l’étape ultérieure de remplissage, nous construisons dans l’étape 3 un masque de profondeur (ensemble de valeurs de profondeur, ces valeurs étant individuellement réglables pour chaque fragment) permettant de limiter la profondeur des fragments qui seront traités dans l’étape 4. L’objectif est d’ignorer rapidement par *culling précoce* tous les fragments ne pouvant avoir un impact sur le remplissage des cracks, à savoir ceux ne tombant pas dans le premier anneau de voisinage des cracks et ayant

certaines propriétés de profondeur, détaillées ci-dessous.

L'étape 3 opère en espace image. Elle génère une valeur de profondeur pour chaque pixel, définie comme étant la valeur maximale de la profondeur du pixel générée dans l'étape 1 et celles des pixels sur le premier anneau de voisinage. Pour les pixels ne définissant pas des cracks et n'étant pas non plus dans le premier anneau de voisinage de ces derniers, cette valeur n'est autre que la profondeur issue de l'étape 1. Pour les pixels représentant des cracks où ceux se trouvant dans leur premier anneau de voisinage, cette valeur correspond à la profondeur à partir de laquelle les fragments de l'étape 4 peuvent être ignorés. Les fragments ayant une profondeur supérieure sont cachés par le crack et ne peuvent participer au remplissage.

L'étape 3 sert aussi à générer une valeur *ZRGB* pour chaque pixel du framebuffer. Cette valeur est une valeur entière non signée, obtenue en concaténant la profondeur et la couleur des pixels obtenues dans l'étape 1, et en représentant le résultat par une valeur pouvant être utilisée par le GPU de manière atomique. La nature atomique garantit que le GPU lit et écrit une valeur de manière exclusive d'un thread à l'autre, dans l'ordre chronologique d'accès. La profondeur est convertie en un entier non signé et est stockée dans les bits de poids fort, tandis que les composantes rouge, verte et bleue de la couleur sont stockées dans les bits de poids faible. Grâce à cette organisation, une opération mathématique *minimum* ou *maximum* effectuée avec deux valeurs *ZRGB* a pour effet de comparer et mettre à jour la profondeur dans les bits de poids fort et de mettre à jour les valeurs rouge, vert et bleu *conjointement* à la profondeur correspondante.

2.4.2 Etape 4 : remplissage en espace objet

La tessellation respectant une déviation maximale écran de k pixels effectuée dans la première étape et la rasterisation qui s'en est suivie a laissé une empreinte à l'écran ne faisant qu'approximer celle de la surface analytique. Les pixels se trouvant dans un anneau de voisinage de $k \leq 0.5$ pixels autour des triangles tessellés peuvent être couverts par l'empreinte de la surface analytique à l'écran, et sont donc potentiellement des cracks. Quelle que soit la valeur de k , nous considérons un anneau de voisinage d'1 pixel, puisque les pixels sont indivisibles (Figure 2.3).

L'étape 4 réalise le remplissage à proprement parler en effectuant le lancer de rayon sur les surfaces qui peuvent potentiellement remplir les cracks. Ces surfaces sont celles situées dans le premier anneau de voisinage de chaque crack et situées à une profondeur de vue inférieure. L'étape 4 passe par un rendu intégral du modèle, comme pour l'étape 1. Notons au passage que notre algorithme ne peut fonctionner qu'avec à sa seule disposition les identifiants de patch et de face B-Rep et la coordonnée u, v associés à chaque pixel lors de l'étape 1. Ces données, n'étant disponibles que pour les fragments rasterisés les plus proches du point de vue, ne permettraient de réaliser un lancer de rayon que sur la surface correspondante, ce qui serait insuffisant pour remplir de nombreux cracks. Pour chaque pixel dans le premier voisinage d'un crack et étant parcouru par une primitive, nous voulons générer un fragment. Pour ce faire, nous réalisons un double rendu, une première fois en mode plein et une seconde fois en mode fil de fer (wireframe). Le mode fil de fer permet de s'assurer que des fragments sont effectivement générés pour les triangles fins (Figures 2.7 et 2.8). Grâce au masque de profondeur préparé dans l'étape précédente et au mécanisme de culling précoce sur la profondeur offert par le GPU, l'étape 4 ne traite que les fragments qui sont dans le premier anneau de voisinage d'un crack (Figure 2.9), et de profondeur inférieure à ce dernier.

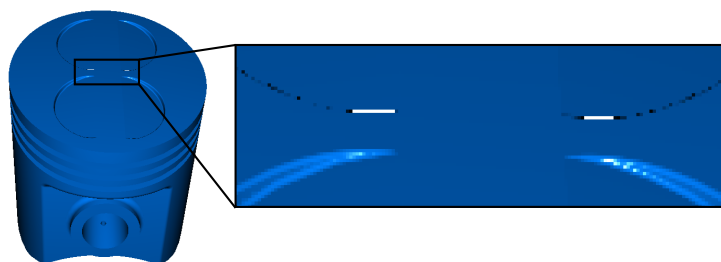


FIGURE 2.7 – Des cracks apparaissent si le rendu en mode fil de fer n'est pas réalisé conjointement à celui en mode plein. Les lignes horizontales blanches (encart) représentent ici des pixels de la couleur du fond d'écran, et qui n'ont pas été remplis sur cette capture où le rendu en mode fil de fer n'a pas été effectué.



FIGURE 2.8 – Empreinte d'une surface tessellée sous forme d'un maillage et rastérisé par le GPU. A gauche : mode plein. Les fragments affectés par la rastérisation laisse une empreinte discontinue à l'écran, puisque seuls 4 fragments ont leur centre tombant véritablement dans l'enceinte des triangles tessellés (Partie I, Figure 3.4). Le rendu obtenu est montré sur la Figure 2.7. Au centre : rendu en utilisant 4 échantillons par fragment. L'empreinte est toujours discontinue. A droite : le mode fil de fer laisse une empreinte de fragments continue, comme nous le voulons.

2.4.2.1 Lancer de rayon sur les surfaces à l'emplacement des cracks

Pour chaque fragment entrant passant le test de profondeur, nous effectuons un lancer de rayon de la surface support correspondante à l'emplacement de chaque pixel flaggé avoisinant. Le lancer de rayon n'est effectué que si le fragment en entrée a une profondeur inférieure à celle définie dans la valeur ZRGB associée au pixel identifié comme crack. En utilisant l'identifiant de face B-Rep et la surface support associée, ainsi que la coordonnée u, v du fragment entrant, nous initions une recherche de racine avec des itérations de Newton successives [FP79] pour déterminer si un rayon ayant pour origine la caméra et passant par l'emplacement du crack a une intersection avec la surface. Si une intersection est effectivement trouvée, nous effectuons la classification selon la méthode de Schollmeyer et Fröhlich [SF09] (Partie II, Section 3.2) pour déterminer si cette intersection se trouve dans la zone de découpe. Si l'itération de Newton ne converge pas, ou si la classification nous informe que l'intersection est hors de la zone de découpe, le crack en cours de traitement n'est pas rempli et le crack suivant est considéré. Autrement, la profondeur et la couleur du fragment issus du lancer de rayon sont utilisés pour mettre à jour la valeur ZRGB, à condition que la profondeur obtenue soit inférieure à la composante Z de cette dernière valeur. La mnémotechnique *atomicMin* est utilisée pour réaliser la mise à jour de manière atomique, de sorte que des mises à jour concurrentes initiées depuis plusieurs fragments (d'une même primitive dans la majorité des cas, mais aussi potentiellement de primitives différentes) soient proprement ordonnancées par le GPU, et que la valeur finale reflète la valeur de profondeur minimale, avec sa couleur associée.

La classification d'un fragment est réalisée par la classification de la coordonnée u, v associée au *centre* du fragment. Des fragments ont ainsi pu être classifiés comme étant hors de la zone

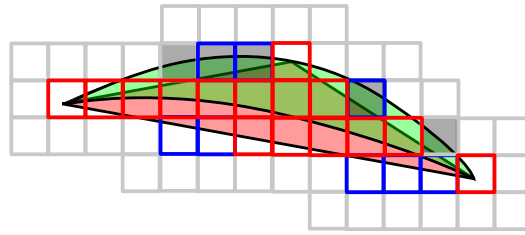


FIGURE 2.9 – En vert : empreinte d’une section de surface B-Rep telle qu’elle est analytiquement définie. Triangle rouge : empreinte d’une approximation à $k = 0.5$ pixel de la section de surface. Carrés rouges : fragments rasterisés en mode plein. Carrés bleus : fragments rasterisés en mode fil de fer. Carrés gris : pixels localisés sur le premier anneau de voisinage autour des fragments rouges et bleus, candidats pour le remplissage. Carrés avec un fond gris : pixels identifiés comme cracks pour lesquels le remplissage va effectivement avoir lieu.

de découpe, et ont par conséquent pu ne pas être rasterisés dans la première étape, bien que recouvrant parfois *en partie* une portion de la surface dans la zone de découpe. De la même manière, ils peuvent être ici ignorés dans l’étape 4 qui effectue un nouveau rendu intégral de la scène. Pour les triangles fins issus de la tessellation, une classification hors de la découpe peut mener à des opportunités manquées mais néanmoins décisives pour la qualité de rendu de remplir des cracks avoisinants. Désactiver la classification pour le rendu en fil de fer permet de forcer la rasterisation des arêtes de tous les triangles et de s’affranchir de ce problème (Figure 2.10).

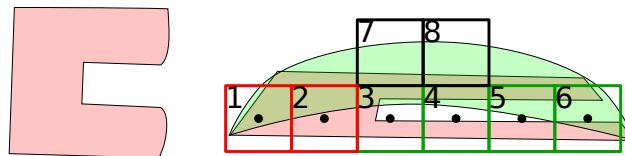


FIGURE 2.10 – A gauche : structure de découpe. A droite : empreinte de la surface associée à l’écran, telle qu’elle est approximée par un quad tessellé (1 quad = 2 triangles rouges). Lorsque la découpe est activée, la rasterisation des triangles rouges ne produit que deux fragments (1 et 2), puisque le résultat de la classification des 4 fragments 3, 4, 5 et 6, en vert, est hors de la face, leur centre étant hors de la face. Les fragments 7 et 8 sont identifiés comme étant des cracks. Alors que le fragment 7 est dans le voisinage du fragment 2 et peut potentiellement être rempli, le fragment 8 ne le peut pas, n’étant un voisin ni de 1 ni de 2.

Nous voulons générer des fragments pour les triangles **faisant dos à la caméra**, mais dont la représentation analytique de la surface associée fait partiellement face à la caméra (Figure 2.11). Ces triangles sont typiquement générés le long des silhouettes, et doivent donner lieu à une rasterisation. Nous nous assurons donc que l’élimination des triangles faisant dos à la caméra (backface culling) est **désactivée** lorsque nous effectuons le rendu en mode fil de fer.

A l’issue de l’étape 4, tous les pixels flaggés dans l’étape 2 ont leur valeur ZRGB mise à jour pour refléter la couleur et la profondeur correspondant à l’intersection entre le rayon passant par la caméra et l’emplacement écran associé au pixel flaggé et la face du modèle visible à cet emplacement. Tous les cracks qui ont été préalablement identifiés sont ainsi remplis.

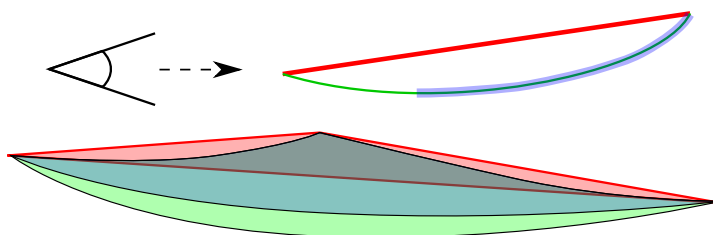


FIGURE 2.11 – Avec l'élimination des triangles faisant dos à la caméra (*back-face culling*) active, ce triangle ne sera pas rasterisé. La section de surface B-Rep qu'il approxime fait pourtant partiellement face à la caméra. C'est le cas pour la partie en vert clair en bas de cette figure. Cette situation se produit principalement avec les triangles très fins.

2.4.2.2 Mécanisme de mailboxing

Nous avons vu dans la Partie I que le rasteriseur du GPU a tendance à traiter les fragments d'une même primitive d'une manière concurrente, par batch. Pour améliorer les performances, nous proposons un mécanisme de *mailboxing* destiné à limiter le nombre de lancés de rayon initiés par des invocations de shader de fragment concurrentes, pour un même pixel à remplir, sur le même patch. Pour chaque pixel identifié comme crack, nous maintenons dans un buffer spécialisé l'identifiant du dernier patch de Bézier pour lequel le lancer de rayon s'est traduit par une intersection et la découpe s'est traduite par une classification *sur la face*, entraînant une tentative de remplissage. Avant chaque lancer de rayon à un emplacement de crack donné, nous lisons l'identifiant de patch dans ce buffer dédié et n'effectuons le lancé que si l'identifiant du patch initiant le lancé est différent. Puisque le traitement des fragments d'une même primitive affecte le remplissage de cracks voisins et potentiellement communs d'un fragment à l'autre, ce procédé permet de s'assurer que la première invocation du shader de fragment parvenant à combler un crack spécifique puisse « prévenir » les autres invocations concurrentes ou ultérieures que les opérations de lancer de rayon ne sont plus nécessaires, pour le patch associé tout du moins, à un emplacement donné.

Remarquons qu'en raison de la nature imprécise de la méthode itérative de Newton, une convergence peut se produire quand le lancer de rayon est initié depuis une coordonnée u, v d'un fragment donné, mais que cette convergence peut échouer lorsqu'il est effectué avec la coordonnée u, v d'un fragment voisin, pourtant proche de la précédente. De la même manière, lorsque la convergence se produit, la coordonnée u, v résultante des itérations peut se trouver *sur la face* alors que pour un fragment voisin, la classification donnerait un résultat *hors de la face*, si la coordonnée u, v en question se trouve à une forte proximité d'une courbe de découpe. Nous n'inscrivons donc l'identifiant de patch de Bézier dans le buffer dédié que si une convergence se produit effectivement et que le classement se fait bel et bien *sur la face*. Si ces conditions ne sont pas remplies, les autres fragments peuvent continuer à essayer de contribuer au remplissage d'un crack spécifique.

2.4.3 Etape 5 : sortie écran

L'étape 5 travaille en espace écran. Les pixels flaggés reçoivent la couleur et la profondeur de la valeur ZRGB correspondante, tandis que les autres pixels reçoivent la couleur et la profondeur obtenues à l'issue de l'étape 1. La valeur ZRGB ne pouvant être sur certaines plates-formes qu'une approximation et non une simple transformation de la profondeur et couleur calculées pendant le lancer de rayon, il est plus opportun d'utiliser les valeurs de couleur et de profondeur présentes

dans les buffers dédiés à la fin de l'étape 1 autant que possible. A l'heure où nous écrivons ces lignes, la plupart des GPU supportent des opérations atomiques sur 32 bits ; en revanche, seuls les GPU les plus récents supportent des opérations atomiques 64 bits.

Chapitre 3

Résultats

3.1 Qualité de rendu

Nous comparons la qualité de nos deux méthodes de remplissage de cracks dans cette section. Nous comparons aussi la qualité d'image entre une méthode de rendu basée sur un lancer de rayon intégral sur toute la scène, et un rendu basé sur une tessellation dynamique avec notre méthode de remplissage en espace objet. La méthode de rendu basée sur un lancer de rayon intégral est implémentée en utilisant l'enveloppe convexe des polygones de contrôle des patches de Bézier, utilisés pour disposer de volumes englobant à l'intérieur desquels le lancer de rayon opère. Les intersections rayon/surface sont calculées en s'appuyant sur la méthode itérative de Newton pour la recherche de racines. Cette méthode est utilisée dans la vidéo démonstrative de Schollmeyer et Fröhlich [SF09].

3.1.1 Comparaison des deux méthodes de remplissage de cracks

Nous comparons la qualité d'image obtenue avec les deux méthodes de remplissage en évaluant la fiabilité de la routine de détection des cracks, et en analysant comment les cracks sont remplis avec chacune des deux méthodes. Pour cela, nous comparons les pixels affectés par le remplissage et ceux correspondant dans une image de référence obtenue avec un lancer de rayon intégral.

La routine de détection de crack est affectée par la valeur du paramètre d_{min} . Une implémentation de notre algorithme peut soit estimer d_{min} de manière précise (Section 2.2), ou définir sa valeur à une estimation volontairement faible de sorte que tous les cracks soient détectés. Dans les deux cas, des pixels non identifiables comme des cracks sont flaggés de manière occasionnelle, notre routine de détection étant parfois légèrement conservatrice sous certains points de vue. Nous devons donc évaluer comment nos méthodes de remplissage se comportent en présence de cracks détectés de manière superflue. Pour notre méthode en espace objet, davantage de remplissage effectué à des emplacements non assimilables à des cracks est synonyme d'une chute des performances et un impact marginal sur la qualité d'image. Nous observons sur ce point clairement les limitations associées à notre routine de remplissage en espace écran, où les cracks représentant des faux-positifs sont remplis comme les autres en utilisant les informations de couleur voisines, ce qui se traduit par des artéfacts, comme ceux montrés sur la Figure 3.1. Augmenter la valeur de d_{min} au-delà de sa valeur idéale (Section 2.2) est tentant, mais entraînerait la non-détection d'un nombre croissant de cracks (Figure 3.2).

Par ailleurs, avec notre méthode en espace image, nous pouvons identifier des artéfacts qui peuvent être vus à la jonction entre les faces où de nombreux cracks sont alignés. Cette méthode de remplissage ne sait pas choisir de manière fiable la meilleure surface avec laquelle les cracks

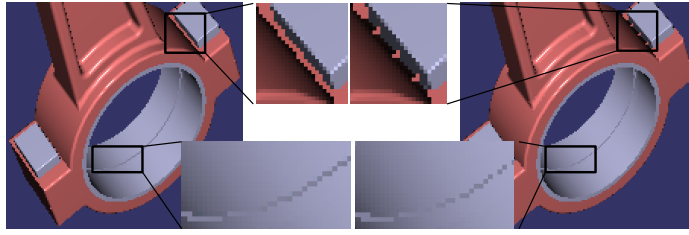


FIGURE 3.1 – Comparaison de la qualité d’image obtenue avec les deux méthodes de remplissage, et en utilisant une détection conservative ($d_{min} = 1.0$). Des cracks sont remplis en se basant sur les couleurs voisines lorsque notre méthode de remplissage en espace écran est utilisée, ce qui se traduit par des artefacts visuels (encart en haut à droite). Notre méthode de remplissage en espace objet ne produit pas d’artéfact car le remplissage, fusse-t-il conservatif, se fait avec un lancer de rayon sur les surfaces au voisinage des cracks (à gauche).

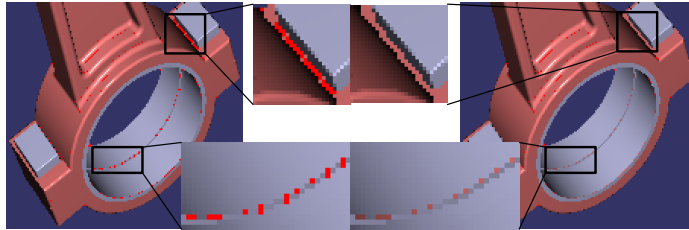


FIGURE 3.2 – Influence de d_{min} sur la détection de cracks. $d_{min} = 1.0$ (à gauche) et $d_{min} = 5.0$ (à droite). Avec $d_{min} = 5.0$, plusieurs cracks ne sont pas détectés et ne seront pas remplis. Avec $d_{min} = 1.0$, tous les cracks sont détectés, mais certains pixels sont détectés comme tels alors qu’ils ne le devraient pas.

doivent être remplis, étant donné que les deux surfaces adjacentes aux cracks sont présentes dans les deux cas sur 3 pixels voisins au crack (Figure 3.3). D’autres artefacts peuvent être observés dans des situations où il y a une fine mais brusque variation sur la profondeur de vue (Figure 3.4).

3.1.2 Comparaison avec un rendu effectué intégralement en lancer de rayon

La qualité visuelle de l’image obtenue avec notre pipeline de rendu est similaire à celle obtenue avec une méthode de rendu intégralement effectué avec des lancers de rayon, offrant des courbes lisses and une absence de cracks. Il existe évidemment un légère divergence entre le rendu des fragments rastérisés et ceux issus du lancer de rayon. Cependant, puisque la déviation entre la surface analytique et son approximation n’est qu’au maximum un demi-pixel, les normales interpolées par la tessellation sont similaires à celles calculées avec le lancer de rayon. Le même modèle d’éclairage est utilisé pour les fragments rastérisés et ceux issus du lancer de rayon.

Un lancer de rayon généralisé sur toute la scène laisse apparaître quelques artefacts en bordure de surfaces (Figure 3.5). Notons que, puisque nous avons décomposé les NURBS d’entrée en patches de Bézier et que nous affichons les patches de Bézier de manière individuelle, ces bordures peuvent être relativement nombreuses. Notre méthode a l’opportunité d’effectuer plusieurs lancers de rayons pour chaque pixel identifié comme crack (Section 2.4.2.2), ce qu’une méthode standard basée sur le lancer de rayon ne fait pas. Notre méthode de remplissage par lancer de rayon ne s’arrête pour un patch spécifique que lorsque la méthode itérative de Newton converge, et la classification de la coordonnée u, v résultante est *sur la face*. Nos tests montrent que permettre plusieurs tentatives de convergence pour le lancer de rayon a un impact positif sur la

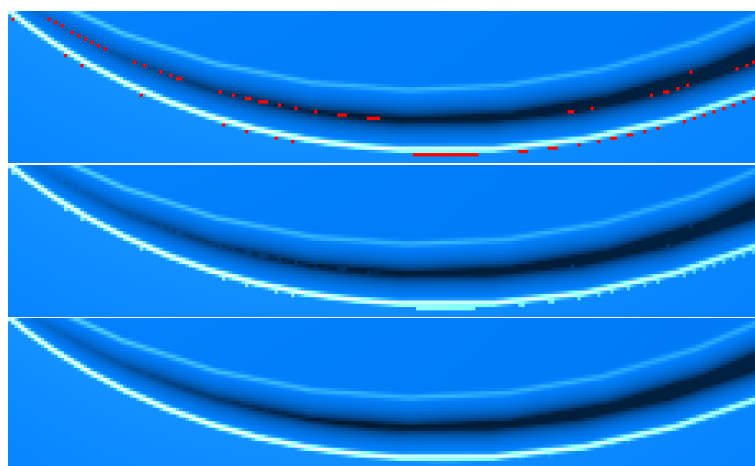


FIGURE 3.3 – *En haut : un grand nombre de cracks peut parfois se trouver aligné le long des jonctions (ligne de cracks située en bas) Au centre : lorsque ces cracks sont remplis, la mauvaise surface peut parfois servir de référence avec notre méthode de remplissage en espace écran, ce qui mène à l'apparition d'artéfacts visuels. En bas : méthode de remplissage en espace objet.*

qualité d'image. La Figure 3.5 compare la qualité d'affichage entre notre pipeline de rendu et une méthode intégralement basée sur le lancer de rayon.

3.2 Performance

Nous évaluons la performance de notre méthode de rendu sur une machine équipée d'un processeur Intel Core i7-860 à 2,8 Ghz, et d'une carte graphique GeForce GTX 780. La résolution utilisée est de 1655x988 pour l'ensemble des tests.

La performance est évaluée pour chaque étape et les résultats sont présentés dans la Table 3.1. Puisque les cracks sont détectés avec une approche fonctionnant en espace écran avec l'information de profondeur, le nombre de cracks détectés n'augmente pas de manière linéaire avec la taille du modèle. Pour les grands modèles, une forte concentration de fragments due à une rasterisation très dense ne laisse apparaître que peu de cracks, les autres étant cachés derrière les primitives se trouvant en avant plan. La performance de notre méthode est de ce fait en grande partie limitée par le nombre de cracks effectivement visibles à l'écran. Notre algorithme minimise de manière efficace le nombre de lancés de rayon à réaliser à travers le mécanisme de culling précoce sur Z dans l'étape 4. De même, la méthode itérative de Newton utilisée pour le calcul d'intersection est initiée avec une coordonnée u, v proche de la coordonnée u, v obtenue après convergence, ce qui permet de réduire le nombre d'itérations et donc d'augmenter les performances de l'étape 4.

D'après nos tests, augmenter les facteurs de tessellation de telle sorte qu'une précision paramétrique k de moins de 0,5 pixel soit respectée n'affecte que très peu les performances, bien que cela puisse avoir un impact légèrement négatif. k a un impact sur le rendu en fil de fer, et réduire sa valeur a pour effet d'augmenter la tessellation, et donc le nombre de fragments issus du rendu en fil de fer et pouvant se trouver dans le voisinage des cracks, ce qui peut nuire aux performances. Il est dommage que les GPU ne proposent pas un mode de rasterisation permettant d'éclairer tous les pixels recouverts par une primitive, y compris ceux recouverts marginalement, ce qui permettrait d'éviter le rendu en fil de fer.

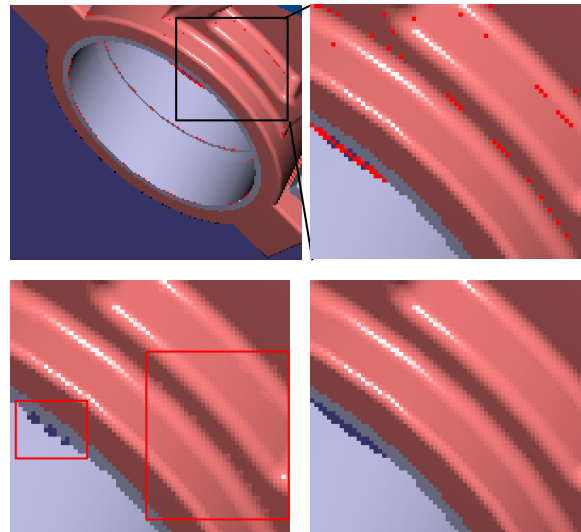


FIGURE 3.4 – *En haut* : cracks détectés, en rouge. *En bas à gauche* : le remplissage en espace écran essaie à tort de remplir l'espace entre deux arêtes de silhouette qui n'ont pourtant aucun lien entre elles (petit encart à gauche). Utiliser une moyenne de couleurs le long des jonctions entre faces laisse des artéfacts mineurs (grand encart à droite). *En bas à droite* : remplissage en espace objet offrant un rendu correct.

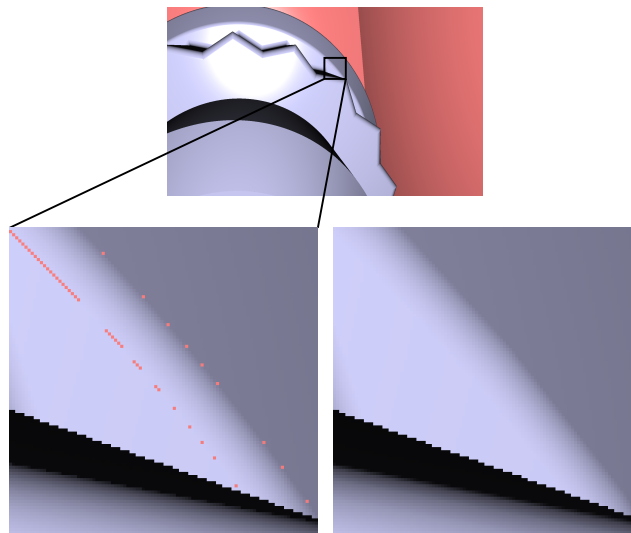


FIGURE 3.5 – *Comparaison de la qualité d'image entre un lancer de rayon réalisé sur toute la scène et un rendu basé sur une tessellation dynamique et utilisant la méthode de remplissage des cracks en espace objet.* *En bas à gauche* : un rendu de la scène entièrement en lancer de rayon laisse apparaître des artéfacts (en rouge). *En bas à droite* : tessellation dynamique et remplissage de crack en espace objet, sans artéfact. Dans les deux cas, la même tolérance est utilisée pour la méthode de Newton (correspondant ici à une distance de 0.001 millimètre en espace objet) et un nombre toujours suffisant d'itérations est autorisé.

	Figure 2.2	Fig. 3.5	Figure 3.6
Etape 1 rendu initial	2.78	3.2	4.38
Etape 2 détection des cracks	2.32	2.35	2.33
Remplissage en espace image			
Etape 3 remplissage	0.4	0.26	0.43
Total méthode espace image	5.5	5.81	7.14
Remplissage en espace objet			
Etape 3 construction du masque de profondeur	0.36	0.4	0.33
Etape 4 remplissage	7.52	2.18	6.82
Etape 5 sortie vers le buffer de rendu final	0.22	0.2	0.18
Total méthode espace objet	13.2	8.33	14.04
Lancer de rayon sur toute la scène	37.24	119	48

TABLE 3.1 – Temps de rendu en millisecondes, avec $k = 0.5$ pixel, $d_{min} = 1.0$ correspondant à 1 millimètre.

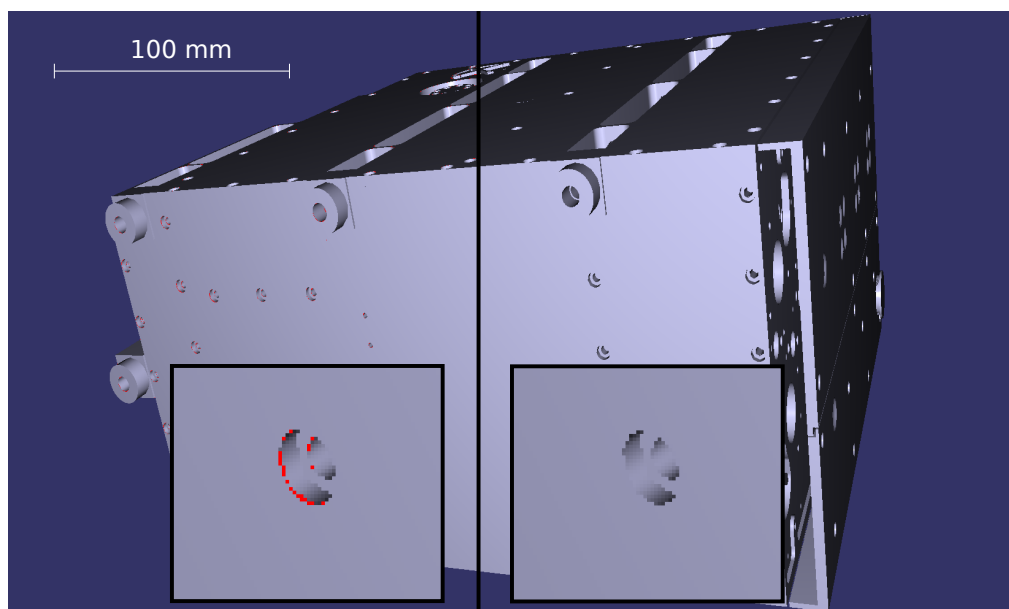


FIGURE 3.6 – Rendu d'une pièce mécanique complexe constituée de 13167 patches de Bézier. À gauche : cracks identifiés en rouges. À droite : rendu final obtenu après avoir rempli les cracks avec la méthode en espace objet.

Chapitre 4

Conclusion

Nous avons proposé une méthode de rendu permettant de bénéficier d'une qualité d'affichage comparable à celle du lancer de rayon, mais généralement deux à six fois plus performante. Notre méthode détecte les cracks créés par la tessellation et les remplit avec deux méthodes différentes. Notre méthode en espace écran a un faible impact sur les performances et une qualité d'image acceptable. Notre méthode en espace objet utilise la couleur et la profondeur obtenues avec un lancer de rayon à l'emplacement des cracks, et permet d'obtenir un rendu d'une qualité similaire à celle obtenue avec un lancer de rayon sur toute la scène, mais avec une vitesse considérablement accrue.

Nos deux méthodes de remplissage peuvent être interchangeables au moment du rendu. Pendant la navigation, la méthode en espace objet peut être utilisée pour améliorer les performances, au prix de quelques artefacts visuels.

Notre algorithme est implémenté sous forme d'étapes de rendu successives qui peuvent s'insérer sur tout système de rendu basé sur la tessellation dynamique, pourvu que celui-ci fournisse une précision de couverture et paramétrique (suivant la définition de Yeo et al. [YBP12]) inférieure ou égale à un demi pixel. Notre implémentation utilise des patchs de Bézier de degrés arbitraires et se base sur des SLEFEs [LP99] pour calculer les facteurs de tessellation. D'autres types de patchs paramétriques et d'autres méthodes de tessellation [ZS00] peuvent être utilisés. La méthode itérative de Newton utilisée pour le lancer de rayon offre un libre choix pour le type de surface et ne repose que sur l'évaluation de points et de dérivées partielles. La méthode de découpe est elle aussi libre, mais requiert une précision irréprochable, orientant notre choix vers la méthode de Schollmeyer et Fröhlich [SF09].

Notre méthode se détache de l'état de l'art en ce sens qu'elle ne requiert la génération d'aucune géométrie additionnelle venant explicitement combler les cracks entre les faces. Elle ne requiert aucun prétraitement spécifique et opère exclusivement pendant le rendu, et est intégralement prise en charge par le processeur graphique. Elle tire pleinement partie des unités de tessellation et de rasterisation des GPU pour lesquels les fabricants proposent des implémentations matérielles dédiées, soigneusement optimisées, au-delà des égards qu'ils témoignent aux unités de traitement parallèle et qui ont le vent en poupe depuis le milieu des années 2000.

Cinquième partie

Conclusion et perspectives

Chapitre 1

Bilan de notre recherche

Nous avons adressé dans cette thèse deux besoins rencontrés dans le domaine de la Conception Assistée par Ordinateur relatifs à la visualisation interactive d'objets plus ou moins complexes et volumineux.

Le premier volet de notre recherche s'est orienté vers le rendu de larges ensembles de données, résultats d'un assemblage final, composés de centaines de milliers, voire de millions de faces B-Rep. Nous avons collaboré avec un industriel pour qui l'utilisation de tels modèles est quotidienne et pour lui proposer des méthodes de rendu innovantes destinées à lever les limitations des techniques de rendu qu'il utilisait, notamment la tessellation statique.

Dans un second temps, nous avons considéré la problématique de rendu au sein des logiciels de modélisation. Nous avons identifié des barrières technologiques et les avons levées. En regard d'un état de l'art analysé dans son ensemble, nous avons pu proposer une approche pour le rendu offrant une qualité d'affichage convoitée, celle du lancer de rayon, mais permettant de mettre à profit les unités de tessellation et de rastérisation communément mises à disposition des programmeurs sur les processeurs graphiques, depuis longtemps.

Chapitre 2

Discussion et perspectives

2.1 Accélération de la découpe

Le travail présenté dans la Partie III [CVB*12] nous a permis de gagner en performances sur la découpe en tirant parti de la classification par triangle pour les surfaces planes. Le gain de performance est significatif pour les surfaces affichées en avant plan. La structure de découpe mise en œuvre permet de classifier des triangles lorsque ceux-ci sont alignés sur notre structure en quadtree. De fait, cette classification ne peut être utilisée qu’avec des niveaux de tessellation en puissance de 2, pour les deux directions paramétriques (u et v), ce qui est relativement contraignant.

Il est possible d’imaginer une structure permettant de classifier n’importe quel triangle dans la zone de découpe, lorsque ce dernier est effectivement sur une zone homogène de cette dernière, et quel que soit son emplacement ou sa taille. La problématique posée est la suivante : organiser la structure de découpe en différentes zones faisant office de structure d’accélération complique voire empêche la classification d’entités recouvrant plusieurs zones adjacentes, comme des triangles. Dans le cadre de l’accès multirésolution, et pour accélérer la classification de fragments distants qui sont nombreux dans une scène, nous avons été amenés à adopter une structure en quadtree. Un quadtree nous sert à extraire facilement un niveau de *mipmap*, permettant de limiter la profondeur de lecture des informations contenues à l’intérieur. Néanmoins, la classification de triangles ne peut être réalisée que si ces derniers sont contenus ou tout du moins alignés sur les cellules, et cela nous empêche d’effectuer la classification dès lors que le facteur de tessellation n’est pas une puissance de 2.

Une structure de découpe idéale pourrait à ce titre être une structure où peu de zones sont nécessaires pour organiser les différentes courbes de découpe, et ce, même si un parcours des différentes courbes (*marching*) est au final nécessaire pour faire la classification d’un point ou d’un triangle. L’absence de zones indépendantes favoriserait la classification de triangles, quels que soient les facteurs de tessellation utilisés indépendamment sur u et sur v . La structure présentée dans la Partie III et celle proposée par Schollmeyer et Fröhlich facilitent ainsi, ou au contraire *limitent* de façon significative, les opportunités de classifier des triangles (Figure 2.1). Une structure basée sur une combinaison logique de zones de découpe bimonotones nous paraît opportune (Figures 2.2 et 2.3). Les sections de courbes bimonotones peuvent décrire un sous-espace de la zone de découpe par extension des droites délimitant le rectangle englobant aux extrémités des courbes par exemple. Combinés de manière logique, ces sous-espaces peuvent définir en totalité la zone de découpe et, si ce n’est pas le cas et seulement alors, la zone de découpe peut être subdivisée en sous-espaces. Nos tests nous ont montré que la classification de

triangles dans de tels espaces bimonotones « décrits de manière logique » est relativement aisée, en recensant de quels cotés de chaque sous-espace se trouvent les sommets d'un triangle.

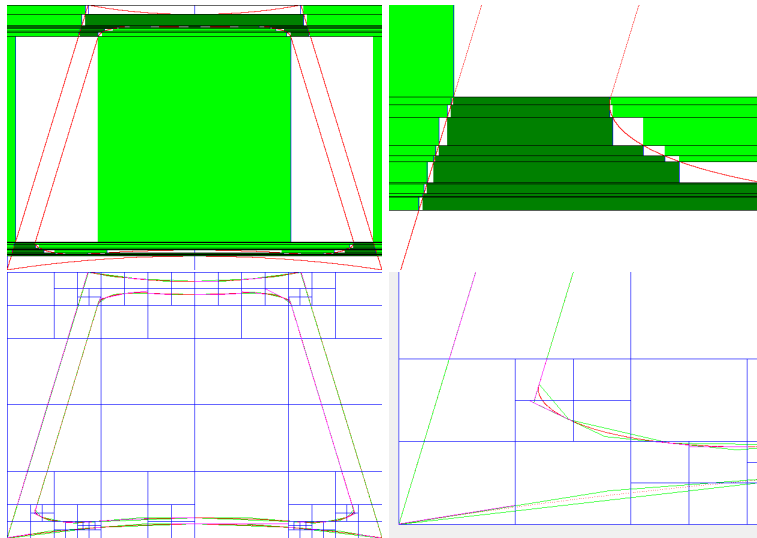


FIGURE 2.1 – En haut : la structure de Schollmeyer et Fröhlich divise l'espace de découpe ce qui peut favoriser (à gauche) ou non (à droite) la classification de triangle, un triangle classifiable ne pouvant recouvrir plusieurs zones de classification. En bas : la structure basée sur un quadtree proposée dans la Partie III peut favoriser davantage la classification de triangles, à condition de tesseller les surfaces avec des facteurs en puissance de 2.

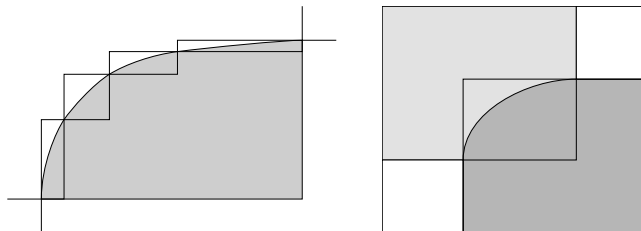


FIGURE 2.2 – Les découpes des faces comportent généralement de larges assemblages de courbes bimonotones (à gauche). Ces zones peuvent être classifiées en combinant logiquement des sous-espaces formés à partir des rectangles englobants les sections de courbes bimonotones (à droite). La classification de grandes zones peut ainsi être assurée sans avoir à subdiviser l'espace paramétrique en zones indépendantes, ce qui a tendance à compromettre la classification d'entités triangulaires ou quadrangulaires.

2.2 Suppression des jours géométriques

2.2.1 Au moment du rendu

Comme nous l'avons décrit auparavant, le rendu par tessellation produit des cracks qui peuvent être supprimés avec notre méthode. Reste à supprimer les jours géométriques (Partie I, Section 2.2).

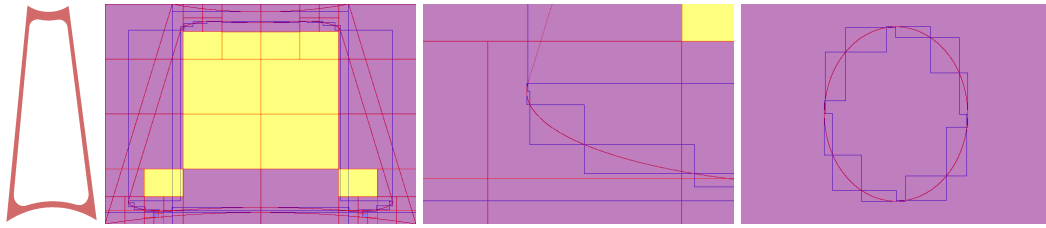


FIGURE 2.3 – *A gauche : découpe à représenter. Au centre gauche : vue d'ensemble d'une structure de découpe basée sur une combinaison logique de sous-espaces définis par des sections de courbe bimonotones. Au centre droit : agrandissement d'une cellule en bas à gauche. A droite : vue d'ensemble d'une structure représentant une découpe circulaire, facile à définir par une combinaison logique de sections bimonotones.*

L'algorithme exposé dans la Partie IV pourrait être amélioré pour permettre de colmater, au-delà des cracks causés par la tessellation, les jours géométriques entre les faces. Une manière de modifier notre algorithme pour éliminer ces jours pourrait être d'effectuer le lancer de rayon et de permettre la rasterisation de fragments hors des zones de découpe, ou même hors du domaine de définition des surfaces support. Faire le rendu des faces au-delà de leurs courbes de découpe ou prolonger les surfaces hors de leur domaine de définition (la plupart des surfaces paramétriques étant mathématiquement définies au-delà de leur domaine) sont des approches qui peuvent être considérées pour souder les faces adjacentes les unes entre elles et pour combler les éventuels jours géométriques présents dans le modèle d'origine. Il faut s'assurer que le prolongement des surfaces est minimisé, dans la mesure où plusieurs surfaces intersectant le rayon passant par un pixel correspondant à un jour géométrique peuvent candidater. Pour ce faire, il faut déterminer un moyen de calculer la distance d'une coordonnée (u, v) à la zone de découpe d'une surface — une opération fort complexe. Relativement à la détection des jours, une heuristique substantiellement plus étoffée que celle présentée dans l'étape 2 de l'algorithme de la Partie IV doit être imaginée.

2.2.2 Via transformation du modèle

Sederberg et al. [SFL*08] transforment un modèle défini par un ensemble de NURBS et découpes associées en modèle T-Spline, puis reconvertissent le modèle obtenu en ensemble de NURBS. Le traitement qu'ils proposent permet de supprimer purement et simplement la problématique de découpe et d'éliminer les éventuels jours géométriques entre les faces. En travaillant avec le modèle obtenu, l'algorithme de Yeo et al. [YBP12] devient suffisant pour faire du rendu précis, interactif, et sans cracks de tessellation ni jours géométriques. Reste à examiner de plus près les performances, notamment pour les modèles disposant de faces fortement découpées, où le nombre de NURBS générées peut devenir extrêmement grand.

Le temps de conversion du modèle à partir de la représentation B-Rep originelle peut également être prohibitif. Dans le travail présenté dans la Partie IV, le temps de conversion du modèle dans un format pouvant être pris en charge par notre moteur de rendu est relativement peu élevé. La conversion peut se faire en temps réel, après chaque opération d'édition au sein d'un logiciel de CAO. La construction de la structure de Schollmeyer et Fröhlich [SF09] n'est que relativement peu coûteuse et la transformation des surfaces support en patches de Bézier rationnels est assez simple. La transformation du modèle proposée par Sederberg et al. demande quant à elle un traitement plus lourd dont le coût exact doit être évalué avec précision. Ce coût peut être prohibitif pour une prise en charge au sein même des logiciels de CAO, qui est l'un des objectifs recherchés dans notre travail portant sur le rendu sans cracks. Les jours géométriques ont souvent

une épaisseur minime. Bien que dérangent pour la fabrication, ils sont généralement invisibles au procédé-même de rendu sur écran, y compris par lancer de rayon, où un rayon ne fait bien souvent qu'intersecter les faces du modèle adjacentes au jour sans jamais aller réellement au travers de celui-ci. Il existe ainsi de nombreux jours entre les faces des modèles illustrés à de nombreuses reprises tout au long de cette thèse (à commencer par le piston), de très faible épaisseur, et sans que le moindre jour soit mis en lumière par quelque procédé de rendu, excepté à des niveaux de zoom extrêmement élevés. Seuls les cracks sont réellement visibles et dérangelants. Une méthode de rendu permettant d'éliminer les jours ou un post-traitement les éliminant directement à partir du modèle B-Rep, en temps-réel, au sein même du logiciel de modélisation si possible, doit à ce titre idéalement être rapide à s'exécuter et facile à mettre en œuvre, pour pouvoir espérer être adoptée par des industriels.

Glossaire

- Accès multirésolution** : Méthodologie d'accès aux données permettant d'aller chercher la stricte quantité d'informations nécessaire, i.e. de résolution précise et bornée, pour réaliser une opération donnée
- Artéfact de rendu** : Imperfection visuelle due à l'utilisation d'une méthode de rendu ayant des limitations
- Bézier clipping** : Méthode de recherche d'intersection entre une courbe de Bézier et une droite, ou une surface de Bézier et un rayon
- Classification dans la découpe** : Permet de déterminer si une coordonnée paramétrique sur la surface se situe sur ou hors de la face, c'est-à-dire dans ou hors la zone de découpe
- Courbe de Bézier quadratique** : Courbe de Bézier non rationnelle de degré 2
- Courbe de Bézier rationnelle** : Courbe paramétrique pouvant être utilisée pour représenter un large éventail de formes géométriques en deux dimensions, y compris les formes coniques
- Courbe de découpe** : Courbe définie dans l'espace biparamétrique d'une surface permettant de définir la découpe
- Crack** : Jour visuel entre deux faces causé par la tessellation non suffisamment coordonnée des faces voisines.
- Culling Z précoce** : Mécanisme permettant de filtrer des fragments en cours de pipeline, avant même qu'ils ne passent entre les mains du programmeur. Généralement assuré par des unités dédiées sur le GPU
- Domaine de définition paramétrique** : Ensemble des valeurs de paramètre pour lesquels une définition est considérée comme valide
- Face B-Rep** : Partie de la surface B-Rep effectivement à l'intérieur de la zone de découpe
- Fragment** : Ensemble de données graphiques, générées et mises à disposition par un procédé de rendu telle que la rasterisation, représentant une primitive géométrique à un emplacement de pixel donnée (résolution inférieure au pixel également possible)
- Hors de la face** : Désigne un emplacement paramétrique de la surface situé hors de la zone de découpe de cette dernière
- Lancer de rayon** : Méthode de rendu reposant sur le calcul d'une intersection entre un rayon correspondant à un pixel de l'image finale à générer, et la scène
- Méthode de Horner** : Méthode d'évaluation de points sur des courbes ou surfaces de Bézier nécessitant un nombre de variables intermédiaires fixe, indépendant du degré
- Méthode de Newton** : Méthode de recherche de racine permettant d'itérer vers une solution avoisinante à partir d'une estimation de départ
- Micro-polygonisation** : Tessellation adaptative où les triangles finaux sont réduits à une taille inférieure à un pixel, voire un demi pixel

- NURBS** : Non Uniform Rational B-Spline. Courbe paramétrique ou surface biparamétrique définie avec des points de contrôle et des poids. Permet de soigner la continuité géométrique d'une courbe ou surface à l'autre et de fournir une relative flexibilité de modélisation. Peut être décomposé en ensemble de courbes de Bézier rationnelles suivant une procédure spécifique
- Processeur de flux** : Unité de traitement élémentaire d'un GPU
- Rastérisation** : Fonctionnalité de rendu pour fabriquer une image matricielle de pixels à partir de primitives géométriques simples
- Rastériseur** : Unité de rastérisation, implémentée de manière hautement optimisée sur un GPU
- Représentation B-Rep** : Représentation des solides à travers la définition géométrique de leur apparence extérieure
- Représentation implicite** : Représentation d'entités géométriques utilisant une formulation fonctionnelle, non paramétrique
- Représentation paramétrique** : Représentation d'entités géométriques en utilisant une formulation à base d'un (courbe) ou plusieurs (surface) paramètres
- Représentation surfacique** : Représentation de l'apparence extérieure des objets solides. Si des jours entre les faces existent, l'étanchéité n'est pas garantie
- Shader** : Sous-programme à implémenter par le développeur destiné à réaliser une tâche de base précise, telle que l'éclairage, la transformation de coordonnées, la tessellation, ou encore le filtrage ou l'amplification géométrique. Depuis 2012, désigne également un sous-programme destiné au traitement générique
- SIMT** : Single Instruction Multiple Threads. Mode opératoire exécutif d'un GPU, où une seule instruction est systématiquement exécutée par tous les processeurs de flux; ceux n'ayant pas vocation à exécuter du code, car étant en attente d'exécution d'une branche de code différente, exécutent le code mais de manière fictive (entrées/sortie vers la mémoire bloquées, le plus souvent)
- Structure de découpe** : Structure de données utilisée pour extraire la face à partir de la surface
- Sur la face** : Désigne un emplacement paramétrique de la surface situé effectivement dans la zone de découpe de cette dernière
- Surface B-Rep** : Entité géométrique biparamétrique telle qu'un plan, un cylindre ou une NURBS, servant de base à la définition des faces
- Surface de Bézier rationnelle** : Surface biparamétrique pouvant être utilisée pour représenter un large éventail de formes géométriques en trois dimensions
- Surface support** : Surface B-Rep
- Tessellation** : Pavage planaire (biparamétrique) utilisant une ou plusieurs formes géométriques, le plus souvent des triangles
- Tessellation adaptative** : Tessellation où les points de discrétisation sont répartis de manière non uniforme dans l'espace biparamétrique, et où les triangles créés approchent une section de la surface originelle suivant différents critères possibles
- Tessellation uniforme** : Tessellation où les points de discrétisation sont uniformément répartis dans l'espace biparamétrique
- Traitement générique parallèle** : Fonctionnalité proposée par les GPUs pour réaliser des traitements et calculs divers, graphiques ou non, et exploitant leur architecture parallèle

Index

- Approximation sous contrôle d'erreur, 27, 84
- Artéfact de rendu, 51, 69, 111, 123
- B-Rep, 9
- B-spline, 83
- Bézier clipping, 60
- Bézier rationnelle, courbe de, 25
- Bimonotone, section de courbe, 35, 49
- Classification, 45, 93
- Classification de triangles, 96
- Continuité géométrique, 11
- Courbe de découpe, 10
- Crack de tessellation, 40, 68, 114
- CSG, 9
- Culling de triangle, 96
- Culling Z précoce, 94, 111, 117
- Découpe, 10, 85
- Echantillonnage uniforme, 27
- Evaluation de dérivées partielles, 25
- Evaluation de points, 25
- Framebuffer, 13
- GPU, adaptation des algorithmes, 21
- GPU, exécution divergente, 20
- GPU, fonctionnement interne, 18
- GPU, rôle, 6
- GPU, ressources, 21
- Interpolation de points, 83
- Jour géométrique, 10, 136
- Lancer de rayon, 57, 118
- Ligne séparatrice, 86
- Méthode de Horner, 26
- Méthode de Newton, 63
- Mailboxing de patch, 121
- Micro-polygonisation, 67
- Modélisation, logiciel, 6
- Niveaux de détails, 42
- NURBS, 10, 25
- Occupation mémoire, 6, 102
- Patch, primitive, 15
- Pipeline, 13
- Processeur de flux, 16
- Quadratique, courbe, 85
- Quadtree, 50, 85
- Rastérisation, 15
- Recouvrement géométrique, 10
- Rendu basé découpe, 45, 74, 77, 111
- Représentation implicite, 58, 85
- Shader, 13
- Shader de fragment, 13, 93, 113, 116, 117
- Shader géométrique, 13, 96
- Shaders de tessellation, 13, 15, 56, 113
- SIMT, 16
- Slefe, 31
- Surface support, 10, 86, 91
- Surfacique, modèle, 11
- Tessellation adaptative, 65
- Tessellation statique, 35
- Tessellation uniforme, 56, 91
- Traitement générique parallèle, 16
- Volumique, modèle, 11

Bibliographie

- [AES91] ABBI-EZZI S. S., SHIRMAN L. A. : Tessellation of curved surfaces under highly varying transformations. *Computer Graphics Forum, Eurographics 1991 10* (1991), 385–397.
- [AGM06] ABERT O., GEIMER M., MULLER S. : Direct and fast ray tracing of nurbs surfaces. In *Interactive Ray Tracing 2006, IEEE Symposium on* (2006), pp. 161–168.
- [BGK04] BALÁZS A., GUTHE M., KLEIN R. : Fat borders : gap filling for efficient view-dependent LOD NURBS rendering. *Computers & Graphics 28*, 1 (2004), 79 – 85.
- [BS93] BARTH W., STURZLINGER W. : Efficient ray tracing for bezier and b-spline surfaces. *Computers & Graphics 17*, 4 (1993), 423 – 430.
- [Cat74] CATMULL E. E. : *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, 1974. AAI7504786.
- [CBV*14] CLAUX F., BARTHE L., VANDERHAEGHE D., JESSEL J.-P., PAULIN M. : Crack-free rendering of dynamically tessellated B-Rep models. *Computer Graphics Forum 33*, 2 (Proceedings of Eurographics 2014) (Apr. 2014).
- [CCC87] COOK R. L., CARPENTER L., CATMULL E. : The Reyes image rendering architecture. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 95–102.
- [CFLB06] CHRISTENSEN P., FONG J., LAUR D., BATALI D. : Ray tracing for the movie ‘Cars’. In *Interactive Ray Tracing 2006, IEEE Symposium on* (2006), pp. 1–6.
- [Cla79] CLARK J. H. : A fast scan-line algorithm for rendering parametric surfaces. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1979), SIGGRAPH '79, ACM, pp. 174–.
- [CLR80] COHEN E., LYCHE T., RIESENFELD R. : Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing 14*, 2 (1980), 87 – 111.
- [CR90] CLINE A. K., RENKA R. J. : A constrained two-dimensional triangulation and the solution of closest node problems in the presence of barriers. *SIAM Journal on Numerical Analysis 27*, 5 (Sept. 1990), 1305–1321.
- [CSS97] CAMPAGNA S., SLUSALLEK P., SEIDEL H.-P. : Ray tracing of spline surfaces : Bézier clipping, chebyshev boxing, and bounding volume hierarchy - a critical comparison with new results. *The Visual Computer 13*, 6 (1997), 265–282.
- [CVB*12] CLAUX F., VANDERHAEGHE D., BARTHE L., PAULIN M., JESSEL J.-P., CROENNE D. : An efficient trim structure for rendering large B-Rep models. In *Vision, Modeling and Visualization* (2012), pp. 31–38.

- [DFKP05] DE FLORIANI L., KOBELT L., PUPPO E. : A survey on data structures for level-of-detail models. In *Advances in Multiresolution for Geometric Modelling*, Dodgson N., Floater M., Sabin M., (Eds.), Mathematics and Visualization. Springer Berlin Heidelberg, 2005, pp. 49–74.
- [DL13] DENG C., LI Y. : A new bound on the magnitude of the derivative of rational Bézier curve. *Computer Aided Geometric Design* 30, 2 (Feb. 2013), 175–180.
- [Ebe06] EBERLY D. H. : *3D Game Engine Design, Second Edition : A Practical Approach to Real-Time Computer Graphics (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.
- [Efr04] EFREMOV A. : *Efficient Ray Tracing of Trimmed NURBS Surfaces*. PhD thesis, 2004.
- [EHS05] EFREMOV A., HAVRAN V., SEIDEL H.-P. : Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces. In *Proceedings of the 21st spring conference on Computer graphics* (New York, NY, USA, 2005), SCCG '05, ACM, pp. 127–135.
- [EML09] EISENACHER C., MEYER Q., LOOP C. : Real-time view-dependent rendering of parametric surfaces. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 137–143.
- [Far90] FARIN G. : *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.
- [FFB*09] FISHER M., FATAHALIAN K., BOULOS S., AKELEY K., MARK W. R., HANRAHAN P. : Diagsplit : parallel, crack-free, adaptive tessellation for micropolygon rendering. In *ACM SIGGRAPH Asia 2009 papers* (New York, NY, USA, 2009), SIGGRAPH Asia '09, ACM, pp. 150 :1–150 :10.
- [Flo92] FLOATER M. : Derivatives of rational Bézier curves. *Computer Aided Geometric Design* 9, 3 (1992), 161 – 174.
- [FMM87] FILIP D., MAGEDSON R., MARKOT R. : Surface algorithms using bounds on derivatives. *Computer Aided Geometric Design* 3, 4 (Jan. 1987), 295–311.
- [FP79] FAUX I. D., PRATT M. J. : *Computational Geometry for Design and Manufacture*. Halsted Press, New York, NY, USA, 1979.
- [GBK05] GUTHE M., BALÁZS A., KLEIN R. : GPU-based trimming and tessellation of NURBS and T-Spline surfaces. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 1016–1023.
- [Gig89] GIGER C. : Ray tracing polynomial tensor product surfaces. In *Eurographics 1989. Proceedings* (1989), pp. 125–136.
- [GJPT78] GAREY M. R., JOHNSON D. S., PREPARATA F. P., TARJAN R. E. : Triangulating a simple polygon. *Information Processing Letters* 7, 4 (1978), 175 – 179.
- [HH11] HANNIEL I., HALLER K. : Direct rendering of solid CAD models on the GPU. In *Proceedings of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics* (Washington, DC, USA, 2011), CADGRAPHICS '11, IEEE Computer Society, pp. 25–32.
- [Hop96] HOPPE H. : Progressive meshes. In *Proceedings of the 23rd annual conference on Computer Graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108.

-
- [HSH09] HU L., SANDER P. V., HOPPE H. : Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 169–176.
- [HSO08] HARRIS M., SENGUPTA S., OWENS J. : Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*, Nguyen H., (Ed.). Addison-Wesley, 2008, pp. 851–876.
- [Kaj82] KAJIYA J. T. : Ray tracing parametric patches. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1982), SIGGRAPH '82, ACM, pp. 245–254.
- [KKM07] KRISHNAMURTHY A., KHARDEKAR R., MCMAINS S. : Direct evaluation of NURBS curves and surfaces on the GPU. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2007), SPM '07, ACM, pp. 329–334.
- [KKMN95] KUMAR S., KRISHNAN S., MANOCHA D., NARKHEDE A. : *Representation and Fast Display of Complex CSG Models*. Tech. rep., Chapel Hill, NC, USA, 1995.
- [KM94] KUMAR S., MANOCHA D. : *Interactive Display of Large Scale Trimmed NURBS Models*. Tech. rep., 1994.
- [Kum96] KUMAR S. : *Interactive Rendering of Parametric Spline Surfaces*. PhD thesis, 1996.
- [LÖ6] LÖW J. : *Ray Tracing Bézier Surfaces on GPU*. PhD thesis, 2006.
- [LB05] LOOP C., BLINN J. : Resolution independent curve rendering using programmable graphics hardware. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 1000–1009.
- [LCWB80] LANE J. M., CARPENTER L. C., WHITTED T., BLINN J. F. : Scan line methods for displaying parametrically defined surfaces. *Communications of the ACM* 23, 1 (Jan. 1980), 23–34.
- [LE09] LOOP C., EISENACHER C. : *Real-Time Patch-Based Sort-Middle Rendering on Massively Parallel Hardware*. Tech. Rep. MSR-TR-2009-83, Microsoft Research, may 2009.
- [LH06] LEFEBVRE S., HOPPE H. : Perfect spatial hashing. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 579–588.
- [LK11] LAINE S., KARRAS T. : High-performance software rasterization on GPUs. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (New York, NY, USA, 2011), HPG '11, ACM, pp. 79–88.
- [LM87] LYCHE T., MORKEN K. : Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design* 4, 3 (Nov. 1987), 217–230.
- [LM88] LYCHE T., MORKEN K. : A data-reduction strategy for splines with applications to the approximation of functions and data. *IMA Journal of Numerical Analysis* 8, 2 (1988), 185–208.
- [LP99] LUTTERKORT D., PETERS J. : Linear envelopes for uniform B-spline curves. In *Curves and Surfaces* (1999), University Press, pp. 239–246.
- [MCFS00] MARTIN W., COHEN E., FISH R., SHIRLEY P. : Practical ray tracing of trimmed NURBS surfaces. *Journal of Graphics Tools* 5, 1 (Jan. 2000), 27–52.
- [NH07] NEHAB D., HOPPE H. : *Random-access rendering of general vector graphics*. Tech. Rep. MSR-TR-2007-95, Microsoft Research, july 2007.

- [NH08] NEHAB D., HOPPE H. : Random-access rendering of general vector graphics. In *ACM SIGGRAPH Asia 2008 papers* (New York, NY, USA, 2008), SIGGRAPH Asia '08, ACM, pp. 135 :1–135 :10.
- [NSK90] NISHITA T., SEDERBERG T. W., KAKIMOTO M. : Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 337–345.
- [NVI07] NVIDIA CORPORATION : *Compute Unified Device Architecture programming guide*. NVIDIA Corporation, 2007.
- [Pav82] PAVLIDIS T. : *Algorithms for Graphics and Image Processing*. Springer, 1982.
- [PM13] PAVANASKAR S., MCMAINS S. : Filling trim cracks on GPU-rendered solid models. *Computer-Aided Design* 45, 2 (Feb. 2013), 535–539.
- [PO08] PATNEY A., OWENS J. D. : Real-time Reyes-style adaptive surface subdivision. In *ACM SIGGRAPH Asia 2008 papers* (New York, NY, USA, 2008), SIGGRAPH Asia '08, ACM, pp. 143 :1–143 :8.
- [PR95] PIEGL L. A., RICHARD A. M. : Tessellating trimmed NURBS surfaces. *Computer-Aided Design* 27, 1 (1995), 16 – 26.
- [PSS*06] PABST H.-F., SPRINGER J., SCHOLLMAYER A., LENHARDT R., LESSIG C., FROEHLICH B. : Ray casting of trimmed NURBS surfaces on the GPU. In *Interactive Ray Tracing 2006, IEEE Symposium on* (2006), pp. 151–160.
- [RCL05] RAY N., CAVIN X., LÉVY B. : *Vector texture maps on the GPU*. Tech. rep., INRIA, 2005.
- [RHD89] ROCKWOOD A., HEATON K., DAVIS T. : Real-time rendering of trimmed surfaces. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1989), SIGGRAPH '89, ACM, pp. 107–116.
- [RNS93] REMBOLD U., NNAJI B., STORR A. : *Computer Integrated Manufacturing and Engineering*. Wokingham, England ; Reading, Mass. : Addison-Wesley Pub. Co, 1993. Later conferences entitled : International Conference on Automotive Electronics.
- [SAG84] SEDERBERG T., ANDERSON D., GOLDMAN R. : Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing* 28, 1 (1984), 72 – 84.
- [SC88] SHANTZ M., CHANG S.-L. : Rendering trimmed NURBS with adaptive forward differencing. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 189–198.
- [Sed83] SEDERBERG T. W. : *Implicit and parametric curves and surfaces for computer aided geometric design*. PhD thesis, West Lafayette, IN, USA, 1983. AAI8400421.
- [Sed95] SEDERBERG T. W. : Point and tangent computation of tensor product rational Bézier surfaces. *Computer Aided Geometric Design* 12, 1 (1995), 103 – 106.
- [Sei91] SEIDEL R. : A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry Theory and Applications* 1 (1991), 51–64.
- [Sel05] SELIMOVIC I. : New bounds on the magnitude of the derivative of rational Bézier curves and surfaces. *Computer Aided Geometric Design* 22, 4 (May 2005), 321–326.

-
- [SF09] SCHOLLMAYER A., FRÖHLICH B. : Direct trimming of NURBS surfaces on the GPU. In *ACM SIGGRAPH 2009 papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 47 :1–47 :9.
- [SFL*08] SEDERBERG T. W., FINNIGAN G. T., LI X., LIN H., IPSON H. : Watertight trimmed NURBS. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 79 :1–79 :8.
- [SGS10] STONE J., GOHARA D., SHI G. : OpenCL : A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering* 12, 3 (2010), 66–73.
- [SH92] SHENG X., HIRSCH B. : Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design* 24, 8 (1992), 437 – 444.
- [SHZO07] SENGUPTA S., HARRIS M., ZHANG Y., OWENS J. D. : Scan primitives for GPU computing. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2007), GH '07, Eurographics Association, pp. 97–106.
- [SS09] SCHWARZ M., STAMMINGER M. : Fast GPU-based adaptive tessellation with CUDA. *Computer Graphics Forum* 28, 2 (Proceedings of Eurographics 2009) (Mar. 2009), 365–374.
- [SSZ*04] SONG X., SEDERBERG T. W., ZHENG J., FAROUKI R. T., HASS J. : Linear perturbation methods for topologically consistent representations of free-form surface intersections. *Computer Aided Geometric Design* 21, 3 (Mar. 2004), 303–319.
- [TL08] TOLEDO R., LEVY B. : Visualization of industrial structures with implicit GPU primitives. In *Proceedings of the 4th International Symposium on Advances in Visual Computing* (Berlin, Heidelberg, 2008), ISVC '08, Springer-Verlag, pp. 139–150.
- [Tot85] TOTH D. L. : On ray tracing parametric surfaces. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), SIGGRAPH '85, ACM, pp. 171–179.
- [TvW88] TARJAN R. E., VAN WYK C. : An $o(n \log \log n)$ -time algorithm for triangulating a simple polygon. *SIAM Journal on Computing* 17, 1 (Feb. 1988), 143–178.
- [Vit01] VITTER J. S. : External memory algorithms and data structures : Dealing with massive data. *ACM Computing Surveys* 33, 2 (June 2001), 209–271.
- [Wal90] WALLIS B. : Graphics Gems. Academic Press Professional, Inc., San Diego, CA, USA, 1990, ch. Tutorial on forward differencing, pp. 594–603.
- [Woo89] WOODWARD C. : Theory and practice of geometric modeling. Springer-Verlag New York, Inc., New York, NY, USA, 1989, ch. Ray tracing parametric surfaces by subdivision in viewing plane, pp. 273–287.
- [WSC01] WANG S.-W., SHIH Z.-C., CHANG R.-C. : An efficient and stable ray tracing algorithm for parametric surfaces. *Journal of Information Science and Engineering* 18 (2001), 541–561.
- [YBP12] YEO Y. I., BIN L., PETERS J. : Efficient pixel-accurate rendering of curved surfaces. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 165–174.
- [YSSP91] YEN J., SPACH S., SMITH M., PULLEYBLANK R. : Parallel boxing in B-spline intersection. *Computer Graphics and Applications, IEEE* 11, 1 (1991), 72–79.
- [ZS00] ZHENG J., SEDERBERG T. W. : Estimating tessellation parameter intervals for rational curves and surfaces. *ACM Trans. Graph.* 19, 1 (Jan. 2000), 56–77.

Résumé

Les logiciels de modélisation géométrique utilisés dans la Conception Assistée par Ordinateur (CAO) servent aujourd'hui à concevoir des objets de toutes sortes, allant de simples lampes de bureau à des avions de lignes entiers. Les modèles de données B-Rep utilisés permettent de définir des formes de manière précise, et de créer des entités géométriques répondant à des critères divers et variés, qu'ils soient esthétiques, qu'ils reposent sur la résistance mécanique, ou encore sur des coûts de production.

Au fur et à mesure des opérations de modélisation dans le logiciel de conception (i.e. CATIA, par exemple), les modèles doivent être affichés de la manière la plus fidèle possible à la représentation analytique, telle qu'elle a été explicitement définie par l'opérateur CAO. Les logiciels de modélisation existant discrétisent les formes définies au rythme des opérations de modélisation, et affichent des polygones approximant face par face les objets 3D résultant de cette discrétisation. Des jours ou "cracks" dus à la tessellation apparaissent avec cette méthode de rendu et sont déplaisant voire gênant pour les opérateurs. Nous proposons dans nos travaux une manière de faire un rendu haute qualité et sans cracks des modèles, sans toucher à leur définition, et en conservant de bonnes performances à l'affichage. Notre méthode peut s'intégrer aux moteurs de rendu existant.

Les sessions de modélisation manipulent des objets de petite ou moyenne envergure. Une fois assemblés, les objets définissant de grosses structures comme par exemple des bateaux ou des avions doivent parfois être visualisés dans leur totalité, et toujours avec une grande précision visuelle. C'est le cas par exemple pour les applications de tests statiques où des techniciens déposent sur une maquette virtuelle des capteurs servant à évaluer le comportement mécanique de centaines voire de milliers de parties localisées sur la structure. Ces capteurs, de la taille d'un ongle, sont physiquement placés sur un produit entièrement usiné et assemblé, tel qu'une section entière de fuselage, dans un hangar. La maquette virtuelle doit permettre de visualiser la structure exactement telle qu'elle est, dans ses moindres détails. Pour réaliser le rendu d'une telle quantité de données de manière interactive et ce, sans sacrifier les performances ou la qualité de rendu à fort niveau de zoom, nous nous appuyons sur les méthodes de rendu dites basées découpe. Nous proposons une structure de découpe rapide, avec laquelle le rendu de très grands modèles peut être effectué. Cette structure est associée à des routines de tessellation dédiées pour chaque type de face B-Rep, ce qui nous permet d'afficher en temps-réel une section entière d'un avion de ligne gros porteur, constitué de centaines de milliers de surfaces B-Rep, tessellées puis découpées à la volée.

Mots-clés: B-Rep, CAO, rendu, modélisation, GPU.

Abstract

Modeling software applications dedicated to Computer Aided Design (CAD) are used to design objects of all sorts, ranging from small, simple desktop lamps to entire, complex aircrafts. The B-Rep models used during this design phase allow CAD operators to define shapes in a very precise manner, and create geometric entities meeting various design criteria, whether they are related to aesthetics, mechanical behavior, or production costs.

As modeling operations are performed in the CAD software application (eg. CATIA, for instance), models must be rendered with a fidelity and accuracy as high as possible with regards to their analytical definition as it has specifically been defined by the CAD operator. Existing modeling software applications discretize shapes as they are being edited and render polygons approximating 3D objects resulting from this discretization, usually done on a face by face basis. Gaps or "cracks" caused by this tessellation show up with this rendering method and are very annoying for the operators. We propose a method to perform crack-free, high fidelity rendering of B-Rep models, with no model preprocess and with good performance, allowing interaction even for large-scale objects. Our method can easily be integrated into existing engines based on dynamic tessellation.

Modeling sessions deal with small or medium scale object parts. Once assembled, these parts form large structures such as boats or aircrafts. These large assemblies may sometimes be interactively visualized as a whole, and once again high precision is strongly desired. Specifically, this is the case for static test applications where technicians place sensors on a virtual model called a Digital Mockup (or DMU) that is used to assess the mechanical strength of thousands of selected locations scattered throughout the model. These nail-sized sensors are then physically placed on a product entirely built and assembled in a hangar, such as an entire aircraft fuselage section. The DMU should represent as faithfully as possible the actual, physical model. For interactive rendering to take place with good frame rates and with high image quality, we rely on a trim-based rendering method with a custom, efficient, multi-resolution trim structure suitable for the rendering of large-scale models. This structure is used in conjunction with dedicated dynamic tessellation GPU routines for common types of B-Rep faces. This combination allows us to render in real-time and on current consumer hardware an entire wide-body aircraft fuselage section composed of hundreds of thousands of B-Rep faces that are dynamically tessellated and trimmed on the fly every time a rendering takes place.

Keywords: B-Rep, CAD, rendering, modeling, GPU.