REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – BISKRA
Faculté des sciences exactes et des sciences de la nature et de la vie
Département d'informatique

Mémoire en vue de l'obtention du diplôme de

Magister en Informatique

Option : Synthèse d'Image et Vie Artificielle

Intitulé

Texturation et déformation interactive de nuages de points

Par

Ali BEDDIAF

Soutenu le 09 juin 2011 devant le jury composé de :

Djedi Noureddine	**Professeur**	**Université de Biskra**	**Président**
Babahenini Med Chaouki	**Maître de Conférences**	**Université de Biskra**	**Rapporteur**
Moussaoui Abdelouahab	**Maitre de Conférences**	**Université de Sétif**	**Examinateur**
Cherif Foudil	**Maître de Conférences**	**Université de Biskra**	**Examinateur**

Résumé

Avec la constante augmentation de la complexité des scènes, et les progrès technologiques enregistrés, il s'avère que l'utilisation de maillages de primitives géométriques pour le rendu est devenue de moins en moins attractive comparée à la manipulation directe des points. Étant donné que, lors du rendu, les traitements traditionnellement faits au niveau de la géométrie de la scène, comme l'élimination des faces cachées, deviennent très coûteux quand la scène est très complexe.

Un nouveau paradigme pour la synthèse d'images 3D, appelé *rendu par points* (RPP) a émergé dans le domaine de l'informatique graphique. Le principe de ce paradigme de rendu est de modéliser les objets de la scène 3D comme un nuage de points sans connexité explicite et de générer une image par une simple projection de ce nuage de points, combinée à des techniques permettant de "remplir" les trous éventuels entre les points projetés afin de donner une impression de surface continue. L'un des avantages principaux du RPP est qu'il permet très simplement d'inclure une représentation multi résolution des objets (une représentation grossière lorsque l'objet est éloigné, et une autre plus détaillée lorsque l'objet se rapproche).

Néanmoins, le fait de ne pas avoir une gestion de la connexité implique qu'il est très difficile d'appliquer sur cette famille d'objets, certaines techniques classiques comme le placage de textures ou les outils de déformations géométriques.

L'objectif de ce travail de magister consiste, dans un premier temps, à formaliser mathématiquement ces opérations de texturation et de déformation sur des nuages, puis de développer différentes techniques algorithmiques permettant d'obtenir des outils interactifs.

Notre contribution consiste en la formalisation et l'implémentation d'un outil de texturation sur les nuages de points à base de splats, avec différents paramètres qualitatifs (primitives de rendu, filtrages, placages) et quantitatifs (motifs, modèles 3D). En plus, un deuxième outil de déformation de forme libre sur les objets 3D à base de points, et surtout, nous mettons la lumière sur le fait que cet outil est interactif grâce à une accélération GPU des cartes graphiques modernes.

Mots clés : *rendu à base de points, texturation, déformation interactive*.

Abstract

With the ever increasing of scenes complexity, and technological progress, it appears that the use of meshes for rendering geometric primitives has become less and less attractive compared to the direct manipulation of points. Given that, at rendering time, the treatment traditionally made on the scene geometry, such as hidden faces removal, become very expensive when the scene is very complex.

A new paradigm for 3D computer graphics, called point-based rendering (PBR) emerged in the field of computer graphics. The principle of this paradigm is to model 3D objects in the scene like a cloud of points without an explicit connectivity and generate an image by a simple projection of this points, combined with techniques to "fill" potential holes between the projected points to give an impression of continuous surface. One of the main advantages of the PBR is that easily allows to include a multi-resolution representation of objects (coarse representation when the object is away, and more detailed representation when the object is closer).

Nevertheless, the fact of not having a connectivity management implies that it is very difficult to apply to this family of objects, some conventional techniques such as the texture mapping or the tools of geometric deformations.

The goal of this magister thesis, initially, is to formalize mathematically these operations of texturing and deformation on the point clouds, then developing various algorithmic techniques to obtain interactive tools.

Our contribution consists of the formalization and the implementation of a tool for texturing point clouds based splats, with different qualitative parameters (rendering primitives, filtering, mapping) and quantitative parameters (textures, 3D models). In addition, a second free form deformation tool of point-based 3D objects, and most importantly, we put the light on the fact that this tool is interactive thanks to a GPU acceleration of modern graphics cards.

Keywords: *point-based rendering, texturing, interactive deformation*.

ملخص

مع الزيادة المستمرة لتعقيد المشاهد، والتقدم التكنولوجي، يبدو أن استخدام الشبكات لعرض الأوليات الهندسية أصبحت أقل جاذبية بالمقارنة مع المعالجة المباشرة للنقاط, بالنظر إلى ذلك، فان المعالجات التقليدية التي تجري على مستوى المشهد كإلغاء السطوح المخفية، أصبحت مكلفة عندما يكون المشهد كثير التعقيد.

لقد انبثق نموذج جديد لرسومات الحاسوب ثلاثية الأبعاد يسمى العرض بالنقاط, ويتكون هذا النموذج من حيث المبدأ على عرض نموذج الأجسام ثلاثية الأبعاد في المشهد وكأنه سحابة من النقاط دون اتصال بشكل حقيقي، وتوليد الصورة النهائية يتم عن طريق إسقاط بسيط لهذه النقاط ، جنبا إلى جنب مع تقنيات "ملء" الثقوب المتوقعة المحتملة بين النقاط لإعطاء انطباع عن سطح مستمر. إحدى المزايا الرئيسية لهذا النموذج هو أنه يتيح بسهولة التمثيل المتعدد الأبعاد للكائنات (تمثيل مبسط عند الابتعاد عن الكائن، وتمثيل أكثر تفصيلا عند الاقتراب منه).

ومع ذلك، فإن حقيقة عدم وجود اتصال بين النقاط يعني أنه من الصعب جدا أن ينطبق على هذه النوع من التمثيل بعض التقنيات التقليدية مثل التغليف بالصور أو أدوات التشويه الهندسية.

والهدف من عمل مذكرة الماجستير هذه، في البداية هو لإضفاء الطابع الرياضي على عمليات التغليف بالصور و ال تشويه الهندسي على سحابات النقاط ، وتطوير تقنيات تسمح بالحصول على أدوات تفاعلية.

مساهمتنا تتمثل في إضفاء الطابع الرياضي و التطوير لأداة تغليف على سحابات النقاط وذلك بإعدادات نوعية مختلفة (أولويات العرض ، التصفية و التغليف) وإعدادات كمية (الصور و النماذج ثلاثية الأبعاد) بالإضافة إلى ذلك ، تطوير أداة ثانية للتشويه الحر للكائنات ثلاثية الأبعاد المكونة أساسا من مجموعة من النقاط ، والأهم من ذلك، نشير الى حقيقة أن هذه الأداة هي أداة تفاعلية بفضل تسارع وحدة المعالجة الرسومية الموجودة في بطاقات الرسومات الحديثة.

الكلمات المفتاحية: العرض بالنقاط ;التشويه التفاعلي

Table des matières

RESUME I

ABSTRACT II

ملخص III

TABLE DES MATIÈRES IV

LISTE DES FIGURES VII

REMERCIEMENTS X

INTRODUCTION GENERALE 1

CHAPITRE I: LA REPRESENTATION A BASE DE POINTS 3

1.1. INTRODUCTION 3

1.2. HISTORIQUE 3

1.3. CLASSIFICATION DES GEOMETRIES BASEES POINTS : 4

1.3.1. NUAGES DESORGANISES 4

A. Points purs 4

B. Points orientés 4

C. Splats 5

D. Points différentiels (splats elliptiques) 5

E. Moving Least Squares Surfaces 6

1.3.2. NUAGES DE POINTS STRUCTURES 7

A. Images de profondeurs 7

B. Représentation volumique d'une surface 8

1.4. PROCESSUS DE RECONSTRUCTION DE LA SURFACE 9

1.4.1. ACQUISITION DES OBJETS 3D 9

1.4.2. RECONSTRUCTION DES SPLATS 9

1.4.3. TRANSFORMATION DES SPLATS CERCLES EN DES SPLATS ELLIPTIQUES 10

1.4.4. SELECTION ET OPTIMISATION 10

A. Algorithme de la sélection avide (Greedy Selection) 10

1.4.5. REMPLISSAGE DES TROUS 11

1.5. VISUALISATION DES NUAGES DE POINTS 12

1.5.1. APPROCHES BASEES LANCER DE RAYONS /BACKWARD WARPING 12

1.5.2. APPROCHES BASEES RASTERISATION (Z-BUFFER) / FORWARD WARPING 14

1.6. RASTERISATION ACCELEREE 14

1.6.1. TESTS DE VISIBILITE 14

A. View-frustum culling 14

B. Back-face culling 15

C. Occlusion culling 15

1.6.2. REPRESENTATIONS MULTIRESOLUTIONS PAR LOD (LEVEL OF DETAIL) 16

A. Qsplat 16

B. Surfels 20

1.7. CONCLUSION 24

CHAPITRE 2 : ÉVOLUTION DU RENDU A BASE DE POINTS 26
2.1. INTRODUCTION 26
2.2. POINTS 26
2.3. SPLATS 28
2.4. ACCELERATION DU RENDU 29
2.5. GPUS PROGRAMMABLES 29
2.6. ECLAIREMENT PLAT, GOURAUD ET PHONG DES NUAGES DE POINTS 31
2.7. CONCLUSION 32
CHAPITRE 3: LA TEXTURATION D'OBJETS 3D A BASE DE POINTS 33
3.1. INTRODUCTION 33
3.2. TEXTURATION 34
3.2.1. DEFINITION 34
3.2.2. FONCTIONS DE PLACAGE (UV MAPPING) 34
A. Placage plan 35
B. Placage sphérique 35
C. Placage cylindrique 35
D. Placage cubique 35
E. Autres placages 36
3.3. TEXTURATION DES NUAGES DE POINTS 36
A. Placage plan 37
B. Placage sphérique 37
C. Placage cylindrique 37
3.4. FILTRAGE DE TEXTURE 38
3.4.1. INTRODUCTION 38
3.4.2. METHODES DE FILTRAGE 38
3.4.3. PRE-FILTRAGE DE TEXTURE 39
3.5. MIPMAPPING 40
3.6. ALIASSAGE 41
3.6.1. INTRODUCTION 41
3.6.2. FILTRE DE RE-ECHANTILLONNAGE 42
3.7. RESULTATS EXPERIMENTAUX 43
3.7.1. COMPARATIF ENTRE LES DIFFERENTES PRIMITIVES DE RENDU 43
3.8. CONCLUSION 44
CHAPITRE 4: LA DEFORMATION INTERACTIVE DE NUAGE DE POINTS 45
4.1. INTRODUCTION 45
4.2. DEFORMATIONS LIBRES SIMPLES 45
4.3. DEFORMATIONS LIBRES DIRECTES 46
4.4. DEFORMATIONS LIBRES ETENDUES 47

4.5. DEFORMATIONS LIBRES AVEC UNE MATRICE DE TOPOLOGIE ARBITRAIRE 49

4.6. AUTRES TYPES DE DEFORMATIONS LIBRES 50

4.6.1. DEFORMATIONS LIBRES RATIONNELLES 50
4.6.2. DEFORMATIONS LIBRES BASEES SUR LES NURBS 50
4.6.3. DEFORMATIONS LIBRES ANIMEES 50
4.6.4. DEFORMATIONS LIBRES DE PLUSIEURS NIVEAUX 50

4.7. DEFORMATIONS SUR LES OBJETS A BASE DE POINTS 51

4.7.1. METHODE DE PAULY ET AL. 51
4.7.2. METHODE DE BOUBEKEUR ET AL. 52

4.8. MISE EN ŒUVRE D'UN DEFORMATEUR DE FORME LIBRE INTERACTIF SUR DES OBJETS A BASE DE SPLATS GRACE A L'ACCELERATION GPU 54

4.8.1. DEFORMATEUR 54
4.8.2. DEFORMATION DES NORMALES 55
4.8.3. DEFORMATION DE FORME LIBRE 56
4.8.4. PIPELINE GRAPHIQUE DE RENDU 58
4.8.5. RESULTATS EXPERIMENTAUX 59
4.8.6. CONTRIBUTION : ECRITURE DU VERTEX SHADER 63

4.9. CONCLUSION 65

CONCLUSION GENERALE 66

BIBLIOGRAPHIE 67

Liste des Figures

FIGURE 1. UNE SCENE REALISEE A L'AIDE DE 3DS MAX. 1
FIGURE 2. UNE SCENE D'UN PLAN D'UNE MAISON REALISEE A L'AIDE D'ARCHICAD. 1
FIGURE 3. LES COURBURES 5
FIGURE . 4. LA PARAMETRISATION DES SPLATS ELLIPTIQUES EN FONCTION DES COURBURES DE LA SURFACE SOUS-JACENTE. 6
FIGURE . 5. ETAPES DES MLS 7
FIGURE .6. ILLUSTRATION D'ECHANTILLONNAGE PAR LDI ET LDC 8
FIGURE.7 . APPROXIMATION LOCALE DE LA NORMALE D'UN ENSEMBLE DE POINTS, PAR LE PLAN DES MOINDRES CARRES (LIGNE DISCONTINUE), ET PAR LE PLAN TANGENT (LIGNE CONTINUE). 10
FIGURE .8. LE MECANISME DU DEVELOPPEMENT DU SPLAT CENTRE INITIALEMENT SUR LE POINT 'PI', AYANT COMME NORMALE 'NI', ET COMME HAUTEUR MAXIMALE 2ϵ*, ET COMME RAYON 'RI' QUI CORRESPOND A LA PLUS GRANDE DISTANCE DE PROJECTION 'DJ' DU POINT 'PJ' SUR LE SPLAT.* 10
FIGURE.9. LA TRANSFORMATION D'UN SPLAT CERCLE EN UN SPLAT ELLIPTIQUE SERA DANS LES DIRECTIONS PRINCIPALES (SECTION 3.1.4) DES COURBURES DE LA SURFACE SOUS-JACENTE AU POINT DONNE. ICI LE VECTEUR U COÏNCIDE AVEC LA DIRECTION DE LA FAIBLE COURBURE. 10
FIGURE .12. ABSENCE DES TROUS ENTRE LES SPLATS ENTRELACES 11
FIGURE .10 . LES SPLATS A,B,C ET D ONT ETE SELECTIONNES CONSECUTIVEMENT EN FONCTION DE LEURS CONTRIBUTIONS DANS LA SURFACE. LE SPLAT A DEVIENT REDONDANT DU FAIT QUE LES SPLATS B,C ET D COUVRENT BIEN LA SURFACE. 11
FIGURE .11. PRESENCE DES TROUS ENTRE LES SPLATS DUS AUX ESPACEMENTS ENTRE LES POINTS (CES ESPACEMENTS SONT INVERSEMENT PROPORTIONNELS DE LA DENSITE DES POINTS). 11
FIGURE .13. CALCUL ET APPROXIMATION DES INTERSECTIONS RAYONS/POINTS 13
FIGURE.14. REALISME APPORTE PAR LE LANCER DE RAYON SOUS FORME D'EFFETS D'OMBRE, DE REFLEXION ET DE REFRACTION LUMINEUSE (DES CAUSTIQUES) DU LAPIN EN VERRE SUR CELUI EN BOIS. 13
FIGURE .15. VIEW-FRUSTUM CULLING. 15
FIGURE .16. LE CONE DE NORMALE OU DE VISIBILITE. 15
FIGURE .17. EXEMPLE D'OCCLUSION. 16
FIGURE .18. CREATION DES SPHERES A PARTIR DU MAILLAGE TRIANGULE APPROXIMANT LE NUAGE DE POINTS. 17
FIGURE .19. ALGORITHME DE CONSTRUCTION DE LA STRUCTURE ARBORESCENTE (HIERARCHIE). 17
FIGURE .20. ALGORITHME DU PARCOURS DE L'HIERARCHIE. 18
FIGURE .21. HIERARCHIE DES SPHERES ENGLOBANTES. 18
FIGURE .22. STRUCTURE HIERARCHIQUE QSPLAT DU LAPIN DE STANFORD. 19
FIGURE .23. RENDU AVEC 3 FORMES DE SPLATS : CARRE (A GAUCHE), CERCLE (AU MILIEU), GAUSSIEN (A DROITE). 19
FIGURE .24. SURFEL BASIQUE. 20
FIGURE .25. DES TROUS ENTRE LES SURFELS BASIQUES. 20
FIGURE .26. SURFEL ETENDU. 21
FIGURE .27. A) PROCESSUS DE PRETRAITEMENT. B) PIPELINE DE RENDU DES SURFELS. 21
FIGURE .28. DEUX LDIS ORTHOGONALES STOCKANT LES SURFELS. 21
FIGURE .29. PRE-FILTRAGE DE TEXTURE AVEC DES DISQUES TANGENTS. 22
FIGURE .30. DEUX NIVEAUX DIFFERENTS D'ARBRE LDC. 22
FIGURE .31. EXEMPLE DE REDUCTION 3 A 1. 23
FIGURE .32. A)SURFEL AVEC DES TROUS. B) LES TROUS DETECTES. C)IMAGE RECONSTRUITE PAR UN FILTRE GAUSSIEN. 24
FIGURE .33. AJUSTEMENT ADAPTATIF ET PROGRESSIF DU TAUX D'ECHANTILLONNAGE (DE GAUCHE A DROITE). 26
FIGURE .34. RENDU DU DAMIER SANS FILTRAGE (A GAUCHE), RENDU DU DAMIER AVEC LE FILTRAGE EWA (A DROITE). 29

FIGURE .35. DIMINUTION DE LA SURCHARGE DU CPU GRACE A LA LINEARISATION DE L'HIERARCHIE, ET AU CONTROLE DES NIVEAUX DE DETAILS PAR LE GPU. 30
FIGURE .36. Z-BUFFER ETENDU OU FLOU. 30
FIGURE .37. RESULTAT DE LA PASSE SPLATTING DE VISIBILITE. 31
FIGURE .38. RENDU AVEC (DE GAUCHE A DROITE) : SPLATS NAÏFS (PLAT), SPLAT AVEC UN FILTRE GAUSSIEN (GOURAUD), PHONG SPLATTING (PHONG). 32
FIGURE .39. LES TEXELS (TEXTURE ELEMENT). 34
FIGURE .40. TRANSFORMATION AFFINE DE LA TEXTURE DE L'ESPACE TEXTURE A L'ESPACE OBJET. 34
FIGURE.41. PLACAGES (DE GAUCHE A DROITE) : PLAN, SPHERIQUE, CYLINDRIQUE ET CUBIQUE. 36
FIGURE.42. PLACAGES (DE GAUCHE A DROITE) : DE RELIEF ET D'ENVIRONNEMENT. 36
FIGURE .43. LA PRIMITIVE SPLAT = CERCLE COLORE + FILTRE GAUSSIEN DE RECONSTRUCTION. 36
FIGURE .44. CALCUL DE LA COULEUR D'UN POINT DONNE EN FONCTION DES TEXELS VOISINS. 38
FIGURE .45. PYRAMIDE DES IMAGES DU MIPMAP. 41
FIGURE .46. ALIASSAGE DU A L'ECHANTILLONNAGE. 42
FIGURE .47. ANTI-ALIASSAGE IDEAL PAR UN PRE-FILTRAGE SUIVI PAR UN ECHANTILLONNAGE. 42
FIGURE .48. FILTRE DE RE-ECHANTILLONNAGE = FILTRE DE RECONSTRUCTION + FILTRE PASSE BAS. 43
FIGURE.49. LES 3 ETAPES DU FFD SUR UN CYLINDRE. 46
FIGURE.50. UN MODELE REALISE ENTIEREMENT AVEC LA METHODE DES DEFORMATIONS LIBRES DIRECTES. 47
FIGURE.51. LE VOLUME SUBDIVISE AVANT ET APRES LA DEFORMATION. 49
FIGURE .52. UNE ETAPE DE SUBDIVISION APPLIQUEE SUR 3 MATRICES DIFFERENTES. 49
FIGURE.53. L'APPLICATION DU CONTROLE DES BORDURES. 50
FIGURE .54. A GAUCHE : UNE FONCTION DE MELANGE, AU MILIEU : CARTE DE COULEUR DU PARAMETRE D'ECHELLE (LE BLEU POUR LA REGION-0 ET LE ROUGE POUR LA REGION-1), A DROITE : LE PLAN DEFORME. 52
FIGURE.55. A GAUCHE LE CYLINDRE ORIGINAL, AU MILIEU : CARTE DE COULEUR DU PARAMETRE D'ECHELLE, A DROITE : LE CYLINDRE DEFORME PAR UNE ROTATION SUR L'AXE Y. 52
FIGURE .56.UNE DEFORMATION PAR UNE COMBINAISON DE DEUX MOUVEMENTS : TRANSLATION ET ROTATION. 52
FIGURE .57. RESULTAT DE L'APPLICATION DES DEFORMATIONS LIBRES SUR LE MODELE DU RAPTOR (8 M TRIANGLES). 54
FIGURE.58. PIPELINE DE RENDU D'OPENGL SIMPLIFIE AVEC L'INTRODUCTION DU MODELE SHADER. 59
FIGURE.59. A GAUCHE: L'ETAT NORMAL D'UNE SPHERE TEXTUREE A BASE DE SPLATS, A DROITE : UNE DEFORMATION LOCALE PAR UN MOUVEMENT TRANSLATIONNEL (ETIREMENT) SUR LA SPHERE. 60
FIGURE.60. A GAUCHE: UNE DEFORMATION LOCALE PAR UN MOUVEMENT ROTATIONNEL (TORSION), A DROITE : UNE DEFORMATION GLOBALE PAR UN MOUVEMENT TRANSLATIONNEL (ETIREMENT) SUR LA SPHERE. 60
FIGURE.61. A GAUCHE : UNE DEFORMATION GLOBALE PAR UN MOUVEMENT ROTATIONNEL (TORSION) SUR LA SPHERE, A DROITE : L'ETAT NORMAL D'UN VISAGE HUMAIN A BASE DE SPLATS. 60
FIGURE.62. A GAUCHE : UNE DEFORMATION LOCALE PAR UN MOUVEMENT TRANSLATIONNEL (ETIREMENT), A DROITE : UNE DEFORMATION LOCALE PAR UN MOUVEMENT ROTATIONNEL (TORSION) SUR LE VISAGE. 60
FIGURE.63. A GAUCHE : UNE DEFORMATION GLOBALE PAR UN MOUVEMENT TRANSLATIONNEL (ETIREMENT), A DROITE : UNE DEFORMATION GLOBALE PAR UN MOUVEMENT ROTATIONNEL (TORSION). 61
FIGURE. 64. GRAPHIQUE DU FPS MESURE LORS DE L'APPLICATION DU DEFORMATEUR DANS SA VERSION CPU, DONT LA GRILLE DES POINTS DE CONTROLE COMPORTE 27 POINTS. 61
FIGURE. 65. GRAPHIQUE DU FPS MESURE LORS DE L'APPLICATION DU DEFORMATEUR DANS SA VERSION CPU, DONT LA GRILLE DES POINTS DE CONTROLE COMPORTE 125 POINTS. 62
FIGURE. 66.GRAPHIQUE DU FPS MESURE LORS DE L'APPLICATION DU DEFORMATEUR DANS SA VERSION CPU, DONT LA GRILLE DES POINTS DE CONTROLE COMPORTE 343 POINTS. 62
FIGURE. 67.GRAPHIQUE DU FPS MESURE LORS DE L'APPLICATION DU DEFORMATEUR DANS SA VERSION GPU, DONT LA GRILLE DES POINTS DE CONTROLE COMPORTE 27 POINTS. 64
FIGURE. 68.GRAPHIQUE DU FPS MESURE LORS DE L'APPLICATION DU DEFORMATEUR DANS SA VERSION GPU, DONT LA GRILLE DES POINTS DE CONTROLE COMPORTE 125 POINTS. 65

FIGURE. 69.GRAPHIQUE DU FPS MESURE LORS DE L'APPLICATION DU DEFORMATEUR DANS SA VERSION GPU, DONT LA GRILLE DES POINTS DE CONTROLE COMPORTE 343 POINTS.....65

Remerciements

Je remercie chaleureusement Monsieur le Professeur Noureddine Djedi pour m'avoir donné le grand honneur de présider ce jury.

J’exprime mes remerciements à Monsieur le Docteur Abdelouahab Moussaoui pour sa disponibilité à juger ce travail malgré ses occupations.

Je tiens à remercier Monsieur le Docteur Foudil Cherif pour son acceptation à examiner ce travail.

J'adresse tous mes remerciements à Monsieur le Docteur Mohamed Chaouki Babahenini pour m'avoir encadré, orienté dans ce travail de recherche, m'avoir accordé son soutien, et m’avoir guidé tout au long de ces années. Je lui exprime également toute ma reconnaissance pour sa serviabilité, sa compréhensibilité, sa disponibilité, sa bienveillance et la pertinence de ses interventions scientifiques pendant ces années.

Sans oublier à remercier mon aimable famille qui m’a offert l’atmosphère adéquate à l’achèvement de ce travail, mes amis : Abderrahim, Amine, Lemya, Mohamed et mes collègues de la direction régionale du commerce batna : Hadj, Rachid, Ferhat, Salem, Amel, Lazhar, Baki et tous ce qui ont participé de près ou de loin, à l’aboutissement de ce travail.

Je souhaite également saluer tous les membres du laboratoire LESIA de l'université de Biskra, et particulièrement les collègues Abdelmoumene Zerari et Nadir Henni de l’équipe RRTR (Rendu Réaliste Temps Réel) pour leur générosité scientifique et fraternelle.

Introduction générale

De nos jours, une large gamme de logiciels professionnels et puissants de modélisation, d'animation et de rendu d'objets 3D, est née, permettant aux infographistes et aux designers de créer des jeux, des films et surtout des effets spéciaux aussi réalistes qu'on n'arrive pas à les discerner de ce qui est réel (*Figure 1*). A titre d'exemple, on cite quelques logiciels qui ont connu une large vogue : 3DS Max, Maya.

Figure 1. *Une scène réalisée à l'aide de 3DS max.*

En plus des utilisations du genre divertissement, ces logiciels 3D servent dans l'industrie comme un outil de conception assistée par ordinateur (CAO) pour la rationalisation des processus d'ingénierie, permettant de produire des maquettes et des prototypes ainsi que les outils permettant de les réaliser. On cite à titre d'exemple pour le domaine de l'architecture, les logiciels suivants : ArchiCAD (voir *Figure 2*), AutoCAD.

Figure 2. *Une scène d'un plan d'une maison réalisée à l'aide d'ArchiCAD.*

Ces logiciels se basent essentiellement sur la manipulation des objets 3D qui sont constitués à la base d'un ensemble de primitives qui sont les polygones. Récemment d'autres primitives de modélisation 3D sont apparues, permettant d'avoir différentes représentations, ces dernières se basent sur des images (création de formes 3D à partir d'images) ou sur des nuages de points (absence de connectivité).

La représentation à base de point a connu des véritables avantages par rapport à la représentation classique à base de polygones, quoique la dérivation et le développement des outils permettant la manipulation efficace ce type d'objets restent un axe de recherche actif dans lequel peu de travaux ont été publiés.

Nous allons dans ce travail, étudier, proposer, mettre en œuvre et valider d'une part une technique pour la texturation des objets à base de points et d'autre part un outil de déformation.

Pour répondre à cette problématique, nous avons organisé notre mémoire en quatre chapitres, dans le premier chapitre, nous présentons un état de l'art sur la représentation à base de points, et nous décrirons l'évolution du rendu à base points dans le deuxième chapitre. Dans les deux chapitres suivants nous présentons l'essentiel de nos deux contributions, ainsi dans le troisième chapitre nous abordons la texturation d'objets 3D à base de points, et finalement nous terminons par la déformation interactive des nuages de points intégrant les GPU.

Chapitre 1: La représentation à base de points

1.1. Introduction

En synthèse d'images, la modélisation géométrique 3D des scènes, représente la première étape de tout projet en infographie. La modélisation polygonale ou par maillage de polygones a montré une qualité agréable avec une erreur d'approximation réduite et une bonne efficacité grâce au développement technologique accru des processeurs graphiques ou GPUs (Graphics Processing Unit) qui sont capables de traiter des centaines de millions de triangles par seconde.

Cependant vu d'une part l'augmentation massive du nombre de polygones représentant des objets nombreux et complexes, la gestion de la scène devient très difficile en matière de manipulation, de traitement, et surtout de visualisation temps réel, et d'autre part l'apparition des scanners photographiques 3D capables d'acquérir non seulement la géométrie, mais aussi l'apparence (orientation) des objets très complexes, et des objets issus du monde réel, sous forme d'un nuage de points discrets, ont mené les chercheurs à penser à une autre primitive géométrique de modélisation autre que le polygone, c'est le point.

1.2. Historique

L'utilisation du point comme primitive de modélisation, n'était pas comme même assez récente. Dans les années 70, les premiers jeux vidéo représentaient déjà des vaisseaux spatiaux explosant grâce à de petits points lumineux remplissant l'écran. En fait, les points ont dans un premier temps principalement été utilisés sous la forme de particules pour modéliser des objets dit "flous", c'est-à-dire dont la surface n'est pas vraiment définie et donc difficilement représentable par les modèles géométriques classiques. Une particule n'est rien d'autre qu'un point 3D associé à un certain nombre d'attributs tels que couleur, taille, densité, forme, accélération, etc. En 1979, Charles Csuri et al. utilisent des particules statiques pour modéliser la fumée sortant d'une cheminée [CHP+79]. En 1982, Jim Blinn anime les particules pour représenter des nuages [Bli82].

Dans le même temps, Reeves présente son fameux système de particules [Ree83], plus générique que le précédent, permettant de simuler feux, explosions et herbes. Ce système de particules a notamment été utilisé pour générer un mur

de feux dans le film Star Trek II : The Wrath of Khan [RLE+82]. Peu après, diverses améliorations ont été proposées permettant de représenter, par exemple, chutes d'eaux [Sim90] et autres fluides [MP89]. Cependant, toutes ces méthodes utilisent des particules dites isotropes, c'est-à-dire sans orientation, afin de représenter des objets volumiques.

En 1985, Levoy et Whitted furent les premiers à envisager l'utilisation du point comme primitive universelle de modélisation et de rendu de surface [LW85].

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

Ainsi la vraie naissance c'était en 1998 grâce à deux travaux simultanés : LDI (Layered Depth Image) de Shade et al. [SGwHS98] et aussi, le système de rendu interactif à base de points, du Grossman et Dally [GD98].

1.3. Classification des géométries basées points :

Deux grandes classes de géométries à base de points se présentent:

- Nuage désorganisé
- Nuage organisé sous forme d'images ou de grille 3D

1.3.1. Nuages désorganisés

La géométrie ici est une liste de points avec les coordonnées (x, y, z), qui peuvent avoir des attributs relatifs à la géométrie de la surface sous-jacent de l'objet 3D.

On cite ci-dessous, les différents modèles du plus simple au plus complexe.

A. Points purs

Ce sont des points qui ne comportent pas d'attributs additionnels autre ses coordonnées tridimensionnelles, c'est-à-dire, pas d'informations supplémentaires relatives à la surface sous-jacente.

B. Points orientés

En plus d'être des points purs, les points orientés ont un attribut qui définit la normale ou l'orientation de ses derniers suivant la surface sous-jacente de l'objet 3D.

Et si la densité des points est élevée, on peut déduire la normale d'un point donné à partir de celles des points voisins (par le calcul du plan des moindres carrés) qui correspond au vecteur propre du la petite valeur propre.

En fait, il reste toujours le problème de l'estimation des trous, c'est-à-dire les espacements entre les points voisins

Ce modèle a beaucoup de lacunes :

- manipulation géométrique locale difficile
- les courbures fortes nécessitent une grande densité de points, mais quand on augmente la résolution d'échantillonnage, un sur-échantillonnage inutile se voit en conséquence dans les régions moins courbées.

C. Splats

Un splat est un point orienté, plus un rayon, formant un cercle ou un disque tangent à la surface de l'objet. Le rayon doit être choisi d'une façon à remplir les trous entre le point et ses voisins.

Le splat joue le rôle de la densité locale ce qui donne une représentation optimale : peu de splats de grands rayons sur les surface plates, et beaucoup de splats de petits rayons sur les courbures, ceci tout en gardant la même erreur d'approximation (différence entre le splat et la surface sous-jacente).

D. Points différentiels (splats elliptiques)

C'est une adaptation optimale des splats pour les courbures locales de la surface. Les splats elliptiques sont définis par : un point pi, une normale, deux axes tangentiels u et v et leur rayons respectifs.

D'abord une courbure d'une courbe à un point P est le ratio $\Delta\varphi/\Delta s$, tel que, Q est un deuxième point proche du point P. Sachons que $\Delta\varphi$ est l'angle entre les deux axes tangents de la courbe sur P et Q, et Δs est la longueur du segment PQ de la courbe (*Figure 3*).

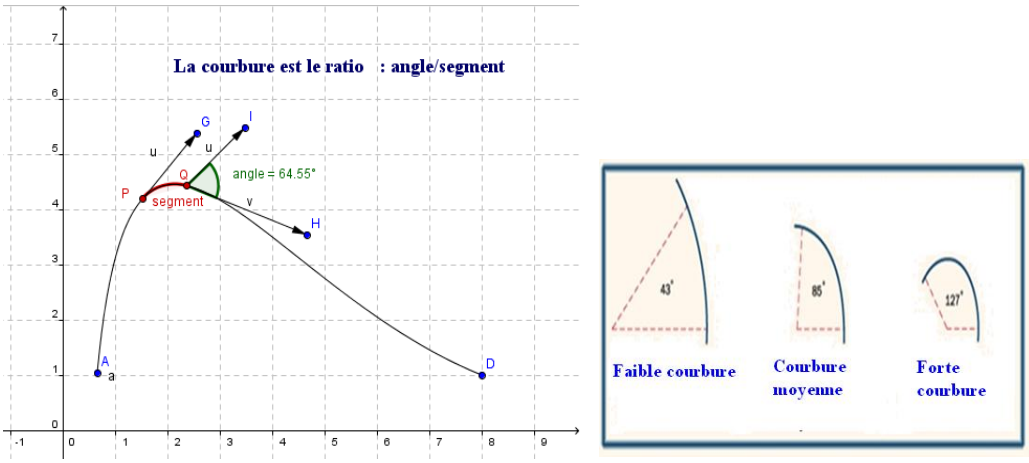


***Figure 3**. Les courbures*

Il sera possible de reconstruire exactement la surface si on a les courbures principales en chaque point de la surface [LR89]. Donc suivant les courbures de notre surface 3D, on paramètre et on ajuste les splats selon le processus suivant :

On trouve d'abord, pour chaque point de la surface 3D : les courbures minimale et maximale ainsi que leurs directions respectives, ces dernières s'appellent les directions principales. Ainsi le splat elliptique doit avoir comme

centre le point donné, et comme directions ou tangents les deux directions principales, et ses rayons sont inversement proportionnels aux courbures extrêmes, c'est-à-dire maximale et minimale (*Figure . 4*).

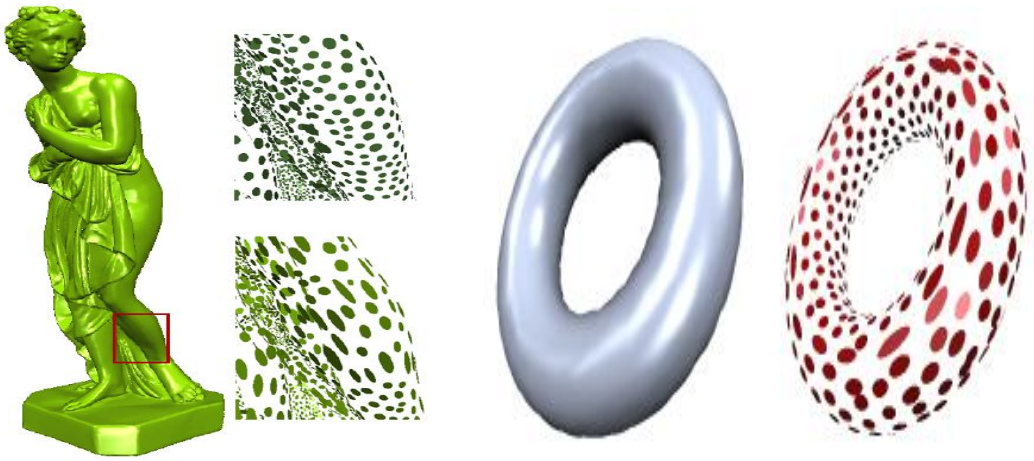


Figure . 4. *la paramétrisation des splats elliptiques en fonction des courbures de la surface sous-jacente.*

Ils permettent ainsi de dériver une normale localement, donc avoir un éclairage par pixel sans faire recours à l'interpolation des normales entre les points.

E. Moving Least Squares Surfaces

La technique des moving least squares (MLS) est une autre méthode de reconstruction ou d'approximation de surface continue à partir d'un nuage de points non uniforme. La principale motivation des MLS est de définir une surface continue à partir d'une représentation purement discrète qui est le nuage de points.

Les MLS ont été introduits dans un premier temps pour la reconstruction de fonctions par Lancaster et Salkauskas [LS81] puis ils ont été étendus à la reconstruction de surface manifold par Levin [Lev01]. Les premières applications à l'informatique graphique ont été proposées par Alexa et al. [ABCO+01] dans le cadre de la reconstruction optimale de surfaces d'un ensemble de points désorganisés par une approximation polynomiale locale utilisant les MLS. Pour cela le nuage de point doit être en premier, aussi réduit que possible (élimination des points bruités et redondants), voir *Figure . 5*.

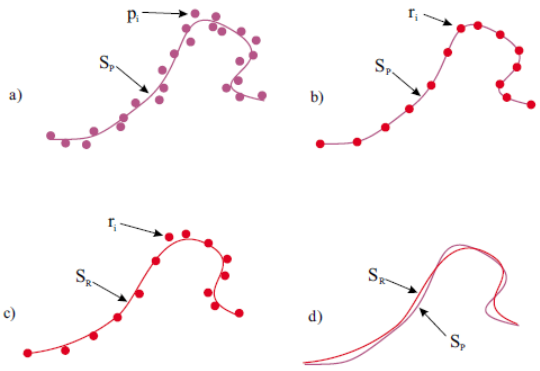


Figure . 5. *Etapes des MLS*

- ***a)*** *Approximation Sp du nuage original {pi}*
- ***b)*** *Déterminaison de l'ensemble réduit {ri}définissant Sp*
- ***c)*** *Approximation Sr du nuage {ri}*
- ***d)*** *Les approximations Sp et Sr se ressemblent.*

1.3.2. Nuages de points structurés

Parmi les avantages de ces types de représentations dites structurées, on peut citer la compacité, ce qui permet de réutiliser les algorithmes classiques de compression d'images pour compresser les données.

Ces types de représentations n'offrent pas non plus suffisamment de flexibilité, et sont principalement limités aux applications de rendu de scènes statiques. Il est en effet très difficile de réaliser une opération d'édition aussi simple que déplacer un point. Ces représentations sont donc très peu adaptées aux applications d'animation ou d'édition d'objets modélisés par points.

A. Images de profondeurs

Dans cette première catégorie, les points sont stockés dans des images 2D associées à un repère 3D, tel que les coordonnées x et y des points dans le repère de l'image deviennent implicites.

Donc ici on n'a qu'à stocker la 3ème coordonnée de chaque point de l'image 2D (ainsi qu'autres attributs : la couleur, la normale, etc)

Une seule image n'est évidemment pas suffisante pour représenter un objet tout entier. Grossman et Dally [Gro98] proposent d'utiliser de nombreuses images de profondeur (dans leur cas 32) acquises à partir de différents points de vue positionnés régulièrement autour de l'objet.

Afin de réduire la redondance induite par les différentes vues, les images sont découpées en petits blocs de 8x8 pixels. Un sous-ensemble de blocs non redondants est ensuite construit en sélectionnant itérativement les blocs parmi la liste des vues orthographiques. Un bloc est sélectionné si le morceau de surface qu'il représente n'est pas déjà correctement représenté par les blocs précédemment sélectionnés. Un premier inconvénient de cette approche est qu'il est possible que certaines parties de l'objet ne soient pas représentées car elles sont invisibles depuis le point de vue utilisé.

Une autre approche consiste à utiliser des images de profondeur en couches communément appelées LDI "Layered Depth Images". Initialement proposée par Shade et al. [SGwHS98] en 1998, une LDI est simplement une image où chaque pixel contient une liste de points se projetant sur le même pixel. En d'autres

mots, une image de profondeur classique stocke uniquement les premières intersections entre la surface de l'objet et les rayons issus des pixels de l'image alors qu'une LDI stocke toutes les intersections. Cependant, une seule LDI ne permet pas un échantillonnage correct de la surface dans toutes les directions puisque les zones tangentielles à la direction de la LDI seront sous-échantillonnées. Lischinski et Rappaport proposent d'utiliser trois LDI orthogonales entre elles et ils appellent cet arrangement un "Layered Depth Cube" (LDC) [LR98](voir *Figure .6*).

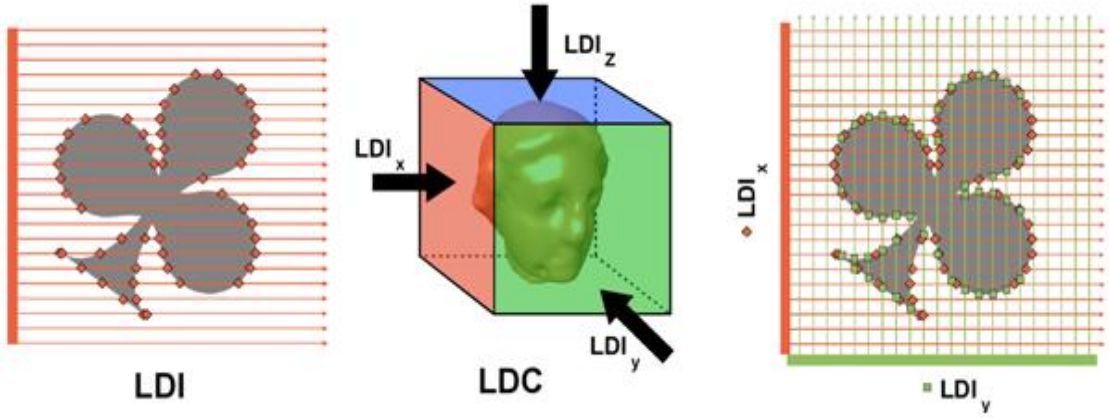


***Figure .6**. Illustration d'échantillonnage par LDI et LDC*

Finalement, pour éliminer la redondance induite par l'utilisation de trois directions orthogonales, Pfister et al. proposent une méthode de réduction appelée "3-to-1 reduction" [PZvG00].

B. Représentation volumique d'une surface

La représentation par voxel (élément de volume) permet en plus d'avoir implicitement les informations des coordonnées x et y des points (comme les LDIs), elle a une information implicite sur la coordonnée z, donc on n'a qu'à stocker les attributs liés aux points.

Du fait que beaucoup de voxels seront inutiles puisque le volume couvert par la surface de l'objet est petit par rapport le volume de la grille 3D, ce qui rend cette représenté peu efficace à cause de la régularité de la grille.

Botsch et al. [BWK02] proposent une représentation hiérarchique efficace d'un nuage de points transformant la grille régulière en octree. Après l'élimination des branches vides et un encodage entropique, le coût de cette représentation devient optimal puisque il sera de complexité de $O(n^2)$ (environ 1.5 bits par point !). Cependant, ce coût de stockage n'est que théorique et ne concerne que la position des points puisque en pratique il est nécessaire de stocker également les attributs des points avec au minimum une normale pour le calcul de l'éclairage.

Pour le rendu, le principal avantage est que ces représentations permettent d'accélérer le rendu via des algorithmes de projection incrémentaux et l'utilisation des calculs sur les entiers. Cependant, ces avantages ne sont valables que pour un rendu logiciel puisque les processeurs graphiques prennent en entrée uniquement des positions 3D explicites.

1.4. Processus de reconstruction de la surface

1.4.1. Acquisition des objets 3D

Les scanners 3D actuels sont capables d'acquérir des modèles très détaillés des surfaces d'un objet 3D sous forme d'échantillons des nuages de points. La reconstruction d'une représentation surfacique pour les surfaces sous-jacentes définies par le nuage de points est un fastidieux travail pour atteindre un rendu de qualité avec une meilleure efficacité sans être pénalisé par la perte des détails fins de la surface.

L'une des solutions est d'utiliser le maillage de triangles comme un modèle de reconstruction des nuages de points [HDD+92], mais cette dernière solution reste toujours inefficace du fait qu'elle résulte un énorme nombre de petits triangles dont la projection ne couvre généralement qu'un pixel.

Une autre solution consiste en l'utilisation des surfaces B-spline au lieu des maillages de triangles, ce qui donne une représentation plus compacte mais avec la perte des détails fins

Dans les dernières années, les techniques à base de points utilisent directement des surfaces pour représenter les surfaces, c'est la représentation par splats. Toutefois, elle exige une fréquence d'échantillonnage élevée (donc une densité des points élevée) pour avoir un rendu de qualité, Mais du fait que la fréquence spatiale d'un tel objet n'est pas régulière, donc on aura un fort échantillonnage même dans les régions de basse fréquence spatiale [KV03], pour cela elle nécessite des techniques de ré-échantillonnage aussi sophistiquées pour améliorer le niveau de détail.

1.4.2. Reconstruction des splats

Pour la génération des splats sous forme de cylindres (disque avec une hauteur), on démarre d'abord par trouver localement la normale du splat sur un point p à partir de ses k voisins par le calcul du plan des moindres carrés, malgré d'autre algorithme utilise plutôt le plan tangent (*Figure.7*), mais cette dernière fournit généralement une approximation meilleure [BK04].

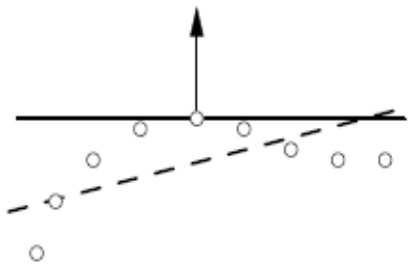


Figure.7 . *Approximation locale de la normale d'un ensemble de points, par le plan des moindres carrés (ligne discontinue), et par le plan tangent (ligne continue).*

Après, et pour une meilleure approximation, on agrandit le splat en longueur par l'ajout progressif des points voisins dont la distance de projection orthogonale sur le plan du splat ne dépasse pas une valeur de 2ϵ [WK04]. Voir *Figure .8*.

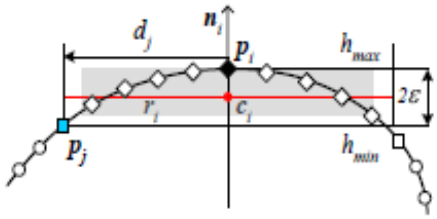


Figure .8. *le mécanisme du développement du splat centré initialement sur le point 'pi', ayant comme normale 'ni', et comme hauteur maximale* $\mathbf{2\epsilon}$ *, et comme rayon 'ri' qui correspond à la plus grande distance de projection 'dj' du point 'pj' sur le splat.*

1.4.3. Transformation des splats cercles en des splats elliptiques

Puis une étape optionnelle d'adaptation des splats en les transformant à des splats elliptiques dans les régions de fortes courbures tout en gardant la même erreur tolérée, comme illustré dans *Figure.9*

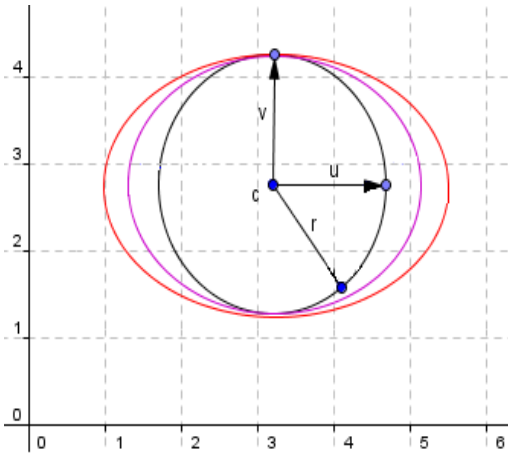


Figure.9. *la transformation d'un splat cercle en un splat elliptique sera dans les directions principales (section 3.1.4) des courbures de la surface sous-jacente au point donné. Ici le vecteur u coïncide avec la direction de la faible courbure.*

1.4.4. Sélection et optimisation

Après avoir une un ensemble de splats initiaux, on doit choisir parmi eux, le sous-ensemble minimum ayant une meilleure couverture du nuage de points avec l'absence des trous. Ceci n'est pas évident, Pour cela on fait recours aux algorithmes de sélections de d'optimisation où l'algorithme de la sélection avide donne généralement une solution convenable.

A. Algorithme de la sélection avide (Greedy Selection)

Il consiste, pour notre problème, en l'ensemble des étapes suivantes :

1. Initialiser la liste des splats sélectionnés SpS = ∅
2. Prendre un ensemble de splats candidats Sp={s1,…,sm} , et un ensemble de points P={p1,…,pn}
3. Sélectionner un splat {si} qui couvre un maximum de points {pj,… ,pk}de l'ensemble P.
4. Mettre à jour la liste des splats sélectionnés : SpS = SpS ∪ {si}
5. Mettre à jour les ensembles Sp et P : Sp=Sp\{si}, P=P\{pj,…,pk}
6. Répéter les étapes de 2 à 5 jusqu'à P = ∅

   En réalité, cet algorithme pourrait présenter des redondances de splats (*Figure .10*), donc une autre étape d'optimisation s'avère nécessaire.

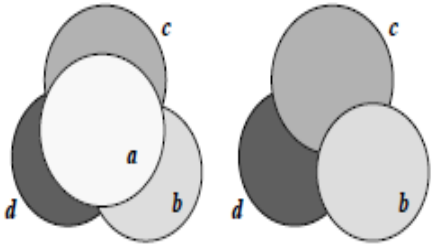


Figure .10 *. Les splats a,b,c et d ont été sélectionnés consécutivement en fonction de leurs contributions dans la surface. Le splat a devient redondant du fait que les splats b,c et d couvrent bien la surface.*

1.4.5. Remplissage des trous

Enfin on essaie de remplir les trous entre les splats. Une solution triviale, est de tester si tous les points sont couverts par au minimum un splat, malgré que ceci n'assure pas l'absence des trous du fait que des espacements entre les splats peuvent existent (*Figure .11*).

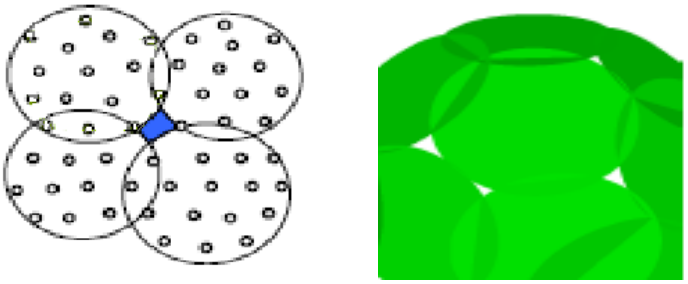


Figure .11. *Présence des trous entre les splats dus aux espacements entre les points (ces espacements sont inversement proportionnels de la densité des points).*

Ces trous se couvriraient parfaitement, si les splats se chevauchaient entre eux (*Figure .12*).

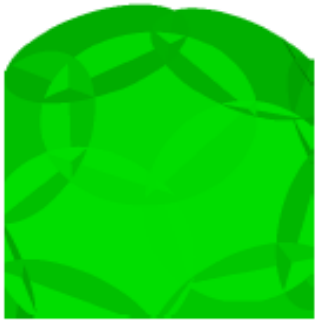


Figure .12. *Absence des trous entre les splats entrelacés*

1.5. Visualisation des nuages de points

On peut classer les approches de visualisation des nuages de points suivant qu'elles se basent sur le lancer de rayon ou la rastérisation.

1.5.1. Approches basées lancer de rayons /backward warping

Notons que dans cette classe, on ne reconstruit plus la surface à partir du nuage de points, comme c'était le cas du Hoppe et al.[HDD+92], cependant l'approche de Schaufler et Jensen[SJ00] permet d'avoir un modèle d'illumination plus complexe appliquée simplement sur une géométrie à base de points purs.

Ces approches résultent un rendu de haute qualité avec un grand réalisme grâce au modèle d'illumination globale du lancer de rayons sur un nuage de points non structuré, et cela permet d'éviter le fastidieux processus de reconstruction de la surface sous-jacente et ses informations topologiques.

Les autres approches ont une chose en commun: pas de modèles d'illumination globale pouvant être appliqués à la géométrie. Dans de nombreux cas la couleur est pré-calculée et stockée par échantillon de points, est copiée dans l'image finale directement. Le seul effet global retardé est les ombres fines des sources lumineuses ponctuelles en utilisant la technique des shadowmaps[OB98].

L'approche Schaufler et Jensen est basée sur une technique de calcul d'intersections qui utilise seulement un sous-ensemble local de points, cette technique comprend la détection d'intersections puis le calcul d'intensité.

La détection est effectuée en entourant le rayon par un cylindre de rayon r et en recherchant le point le plus proche de l'observateur à l'intérieur de ce cylindre. Cette recherche est accélérée en stockant les points dans un octree. Une fois qu'une intersection est détectée, le cylindre associé au rayon est coupé par deux plans orthogonaux, et positionnés ; l'un au niveau du point d'intersection et l'autre dans la direction de visée. L'ensemble des points à l'intérieur de ce petit cylindre est collecté et utilisé pour calculer plus précisément les attributs (position, couleur et normale) du point d'intersection final. Pour cela, ils proposent de mélanger les attributs par la moyenne pondérée suivante :

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

où di est la distance entre le point pi et l'intersection entre le rayon et le plan tangent de pi.

Notons que, comme pour les algorithmes de splatting, cette reconstruction est dépendante du point de vue. Côté performances, leur implantation non optimisée est trois à quatre fois plus lente qu'un programme d'intersection rayon/triangle optimisé. Pour fixer les ordres de grandeurs,

Schaufler et Jensen reportent un temps de rendu de 36 secondes pour un modèle de 500 000 points et une image de 5122 pixels.

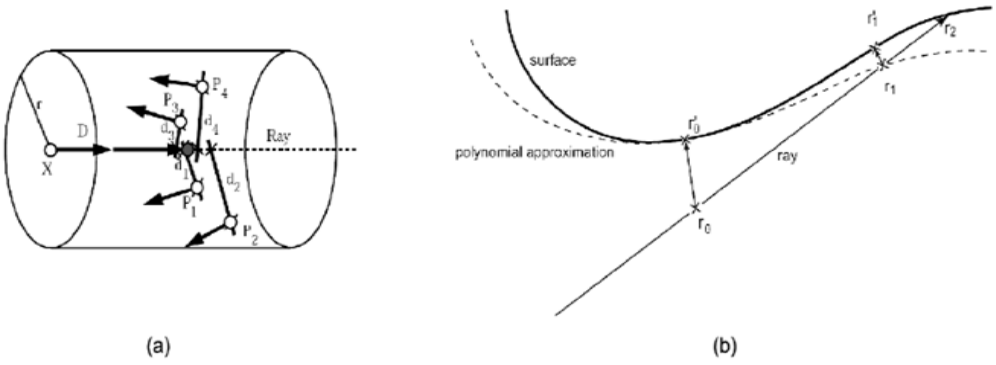


Figure .13. *Calcul et approximation des intersections rayons/points*

(a) Méthode Schaufler et Jensen : les attributs des points contenus dans un petit cylindre autour du rayon sont moyennés [SJ00] (*Figure .13*.a).

(b) Méthode par MLS : à partir d'une première approximation de l'intersection, une approximation polynomiale de la surface est calculée localement (*Figure .13*.b).

Une nouvelle approximation de l'intersection rayon/points est donnée par l'intersection du rayon avec l'approximation polynomiale [AA03b].

La génération des rayons récursivement à chaque intersection permet d'avoir des effets d'ombre, de réflectance (spécularité), de transmissibilité (transparence), voir *Figure.14*.

Malgré la haute qualité, et le réalisme apportés par les approches basées sur le lancer de rayons, ces dernières restent loin d'être des approches pouvant être utilisées en temps réel.

Figure.14. *Réalisme apporté par le lancer de rayon sous forme d'effets d'ombre, de réflexion et de réfraction lumineuse (des caustiques) du lapin en verre sur celui en bois.*

1.5.2. Approches basées rastérisation (z-buffer) / forward warping

Dans ce cas, après avoir reconstruit la surface d'un nuage de points, généralement en splats [ZPvBG01] (comme nous avons vu dans la section 1.4.), le rendu se fait en deux étapes principales : projection de la géométrie sur l'écran et sa discrétisation en pixels (rastérisation), et finalement une phase de Z-buffer pour établir la visibilité entre les pixels. Deux problèmes se posent suivant la densité du nuage de points :

- Aliassage : cas de réduction (zoom out) avec une haute densité, donc plusieurs points vont correspondre à un même pixel, alors que le plus proche seulement sera pris et le manque des autres engendre une perte de géométrie et de texture.
- Trous : cas d'agrandissement (zoom in) avec une densité insuffisante, en plus de l'apparition du deuxième plan (non censé être vu) à travers ces trous.

1.6. Rastérisation accélérée

Vu que la complexité des méthodes par rastérisation est linéaire, et qu'elle dépend proportionnellement du nombre de primitives de la scène, donc pour accélérer le calcul de la visibilité effectué par le z-buffer, il est judicieux de sélectionner parmi l'ensemble des primitives un sous-ensemble réduit qui est censé être visible. Il faut que cette sélection ne pas consomme beaucoup de temps de calcul.

Donc parmi les algorithmes de sélection, on distingue deux grandes familles:

- **Les algorithmes d'élimination des parties cachées (culling):** qui englobent un ensemble de tests de visibilité.
- **Les algorithmes de sélection des niveaux de détails (LOD):** qui adaptent les primitives de la scène suivant la résolution de l'image à calculer, de manière à simplifier la géométrie de l'objet complexe en enlevant ses détails si cet objet est éloigné.

1.6.1. Tests de visibilité

Ces tests servent à éliminer les parties (dans notre cas un sous-ensemble de points) non visibles d'un objet afin d'accélérer le rendu. Généralement, pour simplifier les calculs, des volumes englobants sont utilisés (boîtes ou sphères). On distingue trois types de tests de visibilité : le view-frustum culling, le back-face culling, et l'occlusion culling.

A. View-frustum culling

Ce test vérifie l'appartenance ou non des volumes englobants les points, à la pyramide de vision. Cette dernière est définie par six plans, quatre plans passant

par le centre de projection et les côtés du plan image, et deux autres définissant les distances minimale et maximale (*Figure .15*).

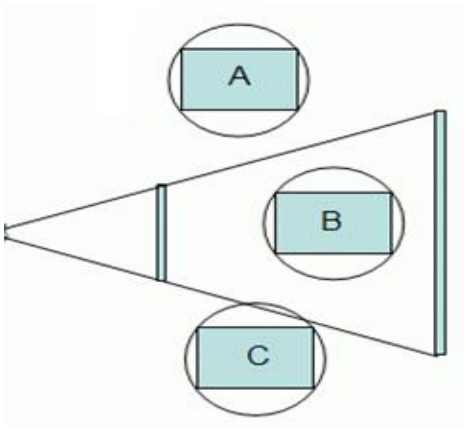


Figure .15. *View-frustum culling.*

B. Back-face culling

Ce test vérifie si l'angle entre la normale d'un point et direction de visée est inférieur à 90° ou non. Lorsqu'on considère tout un groupe de points, on aura besoin d'un moyen permettant de tester rapidement l'ensemble des orientations des points par rapport à la position de l'observateur. Une solution à cette exigence est fournie par la technique des cônes de normales (*Figure .16*).

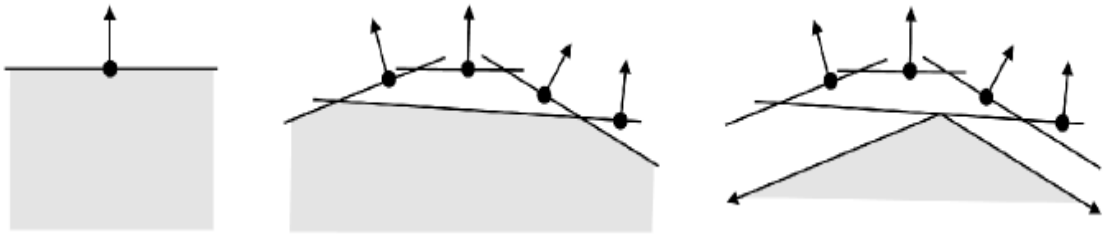


Figure .16. *Le cône de normale ou de visibilité.*

Un cône de normale (ou cône de visibilité) représente un espace depuis lequel aucun point du bloc n'est visible. Le cône de visibilité d'un groupe est construit à partir des normales et des positions des points du groupe. Basiquement, chaque point-normale définit un demi-espace à partir duquel le point ne peut être visible. Le cône de visibilité est alors choisi de manière à approcher au mieux l'intersection de ces demi-espaces.

Au moment du rendu, il suffit de tester si l'observateur est ou non à l'intérieur du cône.

C. Occlusion culling

Ce test vérifie si un point est caché par d'autres points par rapport à la direction de visée (*Figure .17*).

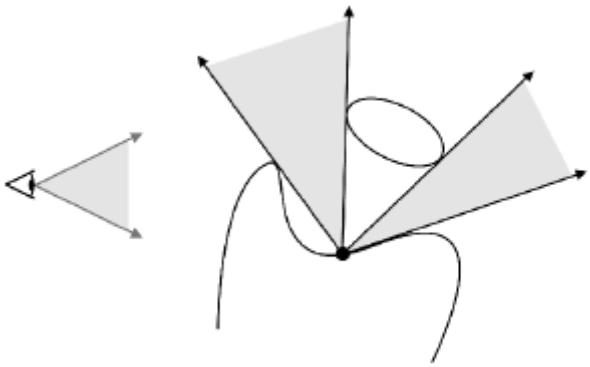


Figure .17*. Exemple d'occlusion.*

Il s'agit bien sûr du problème le plus difficile à résoudre efficacement, et peu de méthodes spécifiques aux nuages de points ont été proposées, et les occlusions restent un très vaste domaine de recherche.

1.6.2. Représentations multirésolutions par LOD (Level Of Detail)

D'une manière générale, il s'agit des structures de données arborescentes ou hiérarchiques dans lesquelles chaque nœud représente une portion de l'objet.

L'union des nœuds d'un niveau donné représente la totalité de l'objet, chaque niveau correspond à un degré de simplification différent. Le niveau 0, c'est-à-dire les feuilles de la hiérarchie, correspond à la meilleure résolution de la représentation. Le nœud d'un niveau quelconque représente une portion de l'objet correspondant à l'union de ses fils, mais avec une densité de points réduite.

Comme étude de cas, on va prendre les représentations : Qsplat et Surfel.

A. Qsplat

Introduction

Rusinkiewicz et Levoy [RL00] ont mis au point un système de rendu point appelé QSplat pour le projet *Digital Michelangelo*. Il s'agit d'une technique de rendu à base de points, permettant d'accélérer le rendu utilisant les tests de visibilité, le contrôle des niveaux de détails et la compression des larges modèles.

Qsplat se base sur le calcul d'une hiérarchie des sphères engloblantes par un regroupement hiérarchique des points (ou des sommets du maillage triangulé correspondant) en des sphères.

Phase de prétraitement

La phase de prétraitement, commence par la représentation du modèle sous forme d'un maillage triangulé. L'objectif étant de simplifier le calcul des normales à chaque nœud (malgré que le travail direct sur le nuage de points est aussi possible grâce au calcul du plan tangent des points voisins).

Après on assigne la taille ou le diamètre des sphères englobantes aux sommets de sorte qu'il n'y aura pas lieu à des trous entre les sphères, et ceci est assuré du fait que les sphères voisines se touchent entre elles (*Figure .18*). Rusinkiewicz et Levoy ont choisi d'assigner un diamètre constant pour tous les sommets qui égale au plus grand diamètre de tout le maillage, ceci rend les sphères un petit peu larges, mais assure la non existence des trous.

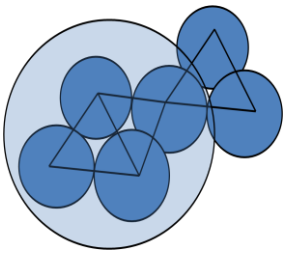


Figure .18. *Création des sphères à partir du maillage triangulé approximant le nuage de points.*

Ensuite, la construction hiérarchique des sphères englobantes se fait en partionnant les sommets selon l'axe le plus long de la boîte englobante selon l'algorithme suivant (*Figure .19*):

```
BuildTree(vertices[begin..end])
{
  if (begin == end)
    return Sphere(vertices[begin])
  else
    midpoint = PartitionAlongLongestAxis(vertices[begin..end])
    leftsubtree = BuildTree(vertices[begin..midpoint])
    rightsubtree = BuildTree(vertices[midpoint+1..end])
    return BoundingSphere(leftsubtree, rightsubtree)
}
```

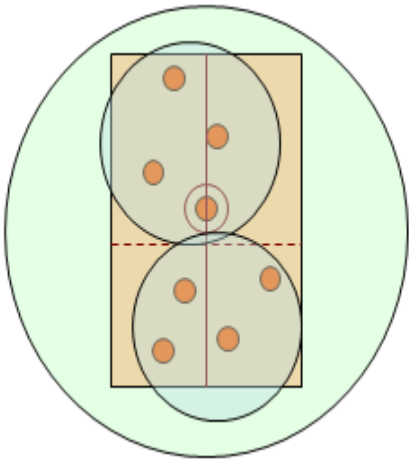


Figure .19*. Algorithme de construction de la structure arborescente (hiérarchie).*

Chaque nœud de l'arbre contient le centre de la sphère et son rayon, une normale, la largeur du cône normale (utilisée pour l'élimination des faces cachées) et éventuellement une couleur.

Phase de rendu

A la phase du rendu, on applique l'algorithme ci-dessous sur les nœuds de l'hiérarchie en partant de la racine (*Figure .20*).

```
TraverseHierarchy(node)
{
   if (node not visible)
      skip this branch of the tree
   else if (node is a leaf node)
      draw a splat
   else if (benefit of recursing further is too low)
      draw a splat
   else
      for each child in children(node)
         TraverseHierarchy(child)
}
```

***Figure .20**. Algorithme du parcours de l'hiérarchie.*

L'hiérarchie est parcourue de la façon suivante :

- Si le résultat du test de visibilité d'un nœud est négatif, on sautera toute la sous-branche de ce nœud.
- On ne rend un nœud sauf :
  - si un nœud feuille est atteint ou
  - la surface de la sphère englobante projetée sur le plan des visions, est inférieure à un seuil prédéfini.
- Autrement on itère les instructions ci-dessus sur chacun des nœuds fils du nœud courant (*Figure .21*).

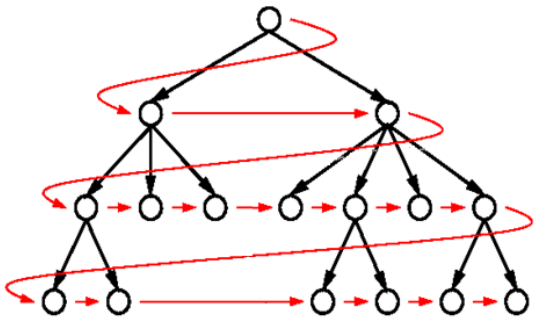


***Figure .21.** Hiérarchie des sphères englobantes.*

Le rendu d'un nœud se fait en affichant le splat associé, pour cela :

- La taille des splats est basée sur le diamètre de la sphère projetée en cours,
- La couleur est obtenue à partir d'un calcul d'éclairage en se basant sur des informations de la couleur et de la normale (*Figure .22*). Les splats sont rendus avec le Z-buffer ce qui comble la non prise en charge du test d'occlusion.

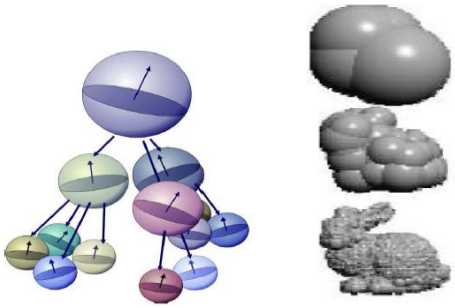


Figure .22. *Structure Hiérarchique Qsplat du Lapin de Stanford.*

Trois types de splats peuvent être utilisés:

- Les points d'OpenGL (primitive GL_POINT) avec une taille définie, ce qui le rend un carré.
- Un cercle opaque, qui est rendu par exemple sous forme de plusieurs petits polygones.
- Un cercle flou, qui est rendu avec un test alpha qui correspond à une approximation ou un filtre gaussien (*Figure .23*).

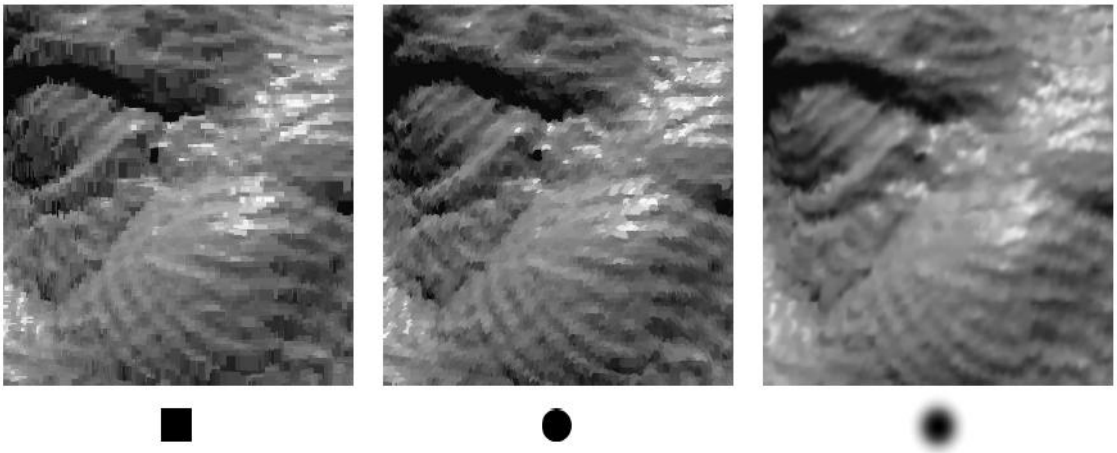


Figure .23. *Rendu avec 3 formes de splats : carré (à gauche), cercle (au milieu), gaussien (à droite).*

Bien sûr, les splats ici peuvent être ronds ou elliptiques. Cependant les splats elliptiques peuvent améliorer la qualité mais aussi peuvent causer des trous dans certains espaces.

QSplat peut rendre entre 1,5 et 2,5 millions de points par seconde :

- Un rendu accéléré grâce aux pré-calculs de l'illumination qui sont stockés au niveau de chaque élément.
- Possibilité entre trois primitives de rendu : carré, splat (cercle ou elliptique), sphère.
- Un affichage rapide apporté par l'utilisation des tests de visibilité (view-frustum et back-face culling).
- Un contrôle des niveaux de détails grâce à l'hiérarchie des sphères englobantes.

- Cependant cette représentation nécessite une étape de réorganisation du nuage de points en un maillage triangulé.
- Le diamètre de toute sphère englobant les sommets du maillage est constant et aussi large de sorte qu'il assure un remplissage des trous entre les sphères, mais ceci au détriment de la qualité du modèle approximant le nuage de points.
- Pas de vrai test d'occlusion culling.

B. Surfels

Introduction

En 2000, Pfister et al [PZvG00] ont introduit le concept de Surfel. Surfel est l'abréviation anglaise de Surface Element. C'est un nouveau paradigme qui utilise le point comme primitive géométrique avec des attributs, pour but d'avoir un rendu fluide et interactif des scènes complexes. L'idée principale est de discrétiser un objet en un ensemble de petites surfaces, où chacune décrit et reflète les attributs de la surface sous-jacente.

Initialement la forme basique des surfels est simplement constituée par la position tridimensionnelle, plus la couleur (*Figure .24*) :

Surfel_Basique {Position, Color }

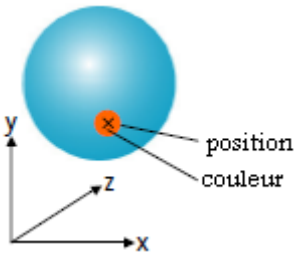


Figure .24*. Surfel basique.*

Cette forme résulte des trous entre les surfels basiques (*Figure .25*).

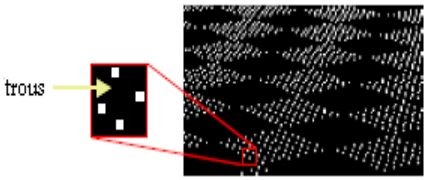


Figure .25. *Des trous entre les surfels basiques.*

Il est donc nécessaire d'interpoler la surface entre les points, ceci en étendant les surfels par une normale et d'autres attributs pour permettre des effets d'illumination avancés (*Figure .26*).

Surfel_Etendu{position, coleur, normale, rayon}

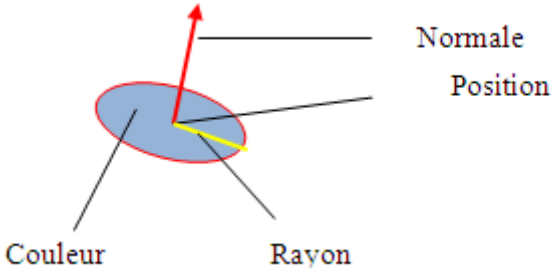


Figure .26*. Surfel étendu.*

Schéma global de surfel

La vue d'ensemble peut être divisée en deux phases (une phase de prétraitement et une autre de rendu) selon le schéma présenté dans *Figure .27* .

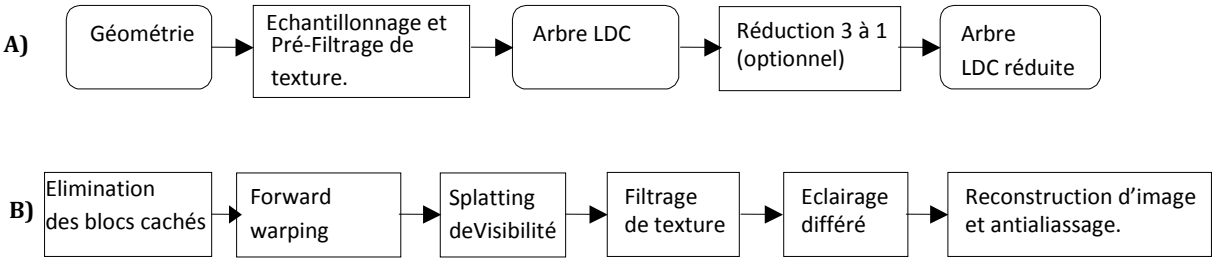


Figure .27. A) *Processus de prétraitement.* ***B)*** *Pipeline de rendu des surfels.*

Processus de prétraitement

Le processus convertit les objets géométriques et leurs textures en surfels. On emploie le lancer de rayon distribué pour créer trois LDI orthogonales. Les LDIs stockent de multiples surfels le long de chaque rayon, un pour chaque point d'intersection rayon-surface (*Figure .28*). Ces trois LDIs orthogonales forment un LDC (Layered Depth Cube) ou bloc.

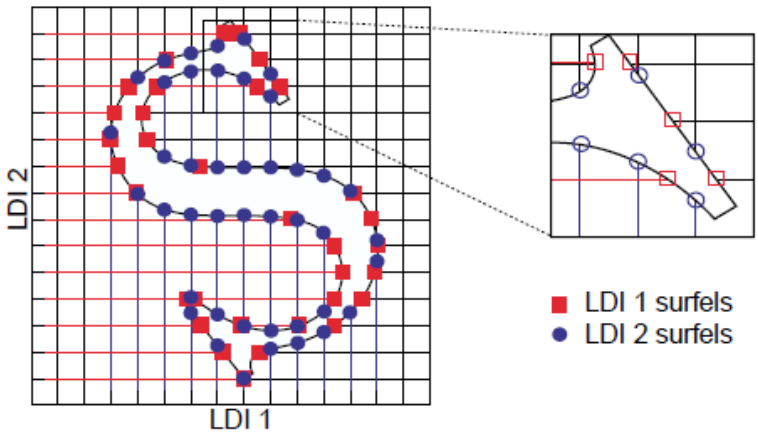


Figure .28. *Deux LDIs orthogonales stockant les surfels.*

Un nouvel aspect important de cette méthode, est la distinction entre l'échantillonnage de la forme (la géométrie) et celui de l'apparence (la couleur de texture). Un surfel stocke la forme (telle que la position et la normale) et l'apparence (tels que les multiples niveaux de couleurs de textures pré-filtrées).

La caractéristique clé du rendu des surfels, est que les textures sont pré-filtrés de l'espace objet vers l'espace texture durant cette étape, et pour éviter l'aliassage de texture, on applique un filtrage de texture par surfel durant l'étape de rendu (*Figure .29*).

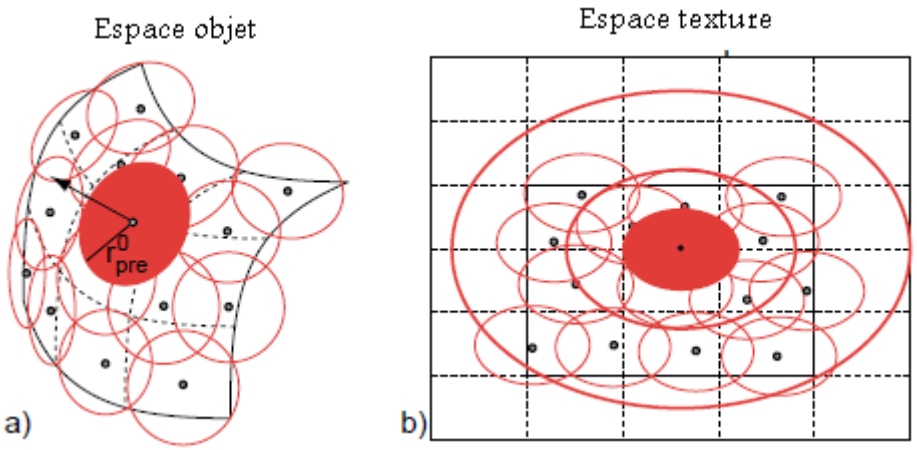


Figure .29. *Pré-filtrage de texture avec des disques tangents.*

En raison des similitudes avec les mipmaps traditionnels de texture, on appelle cette information hiérarchique de couleur : ***mipmap de surfel.***

A partir du LDC on crée une structure de données hiérarchique efficace pour le rendu. Pfister et al emploient une structure hiérarchique en divisant l'espace et en stockant un LDC à chaque nœud de l'octree. Chaque nœud de LDC dans l'octree est appelé un *bloc*. La structure de données résultante est appelée «***arbre LDC*** » (*Figure .30*).

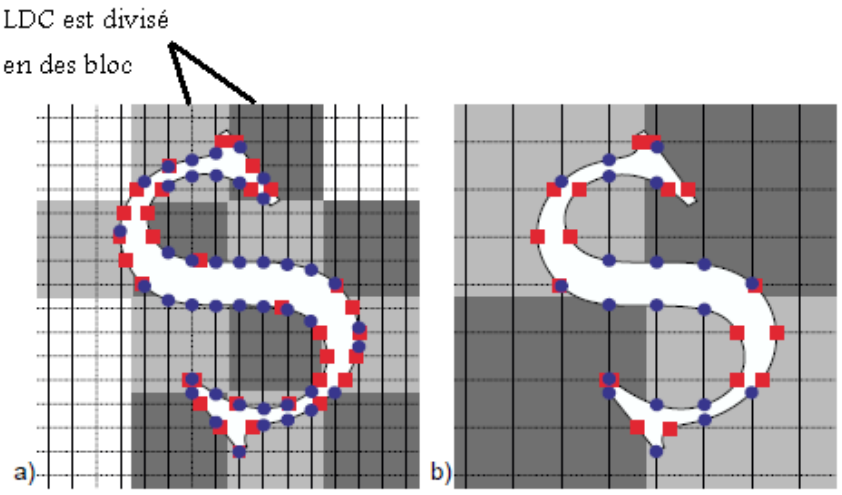


Figure .30. *Deux niveaux différents d'arbre LDC.*

Dans une étape facultative appelée réduction 3-à-1 on est amené à réduire le LDC en une seul LDI.

D'abord, les surfels vont subir une opération de ré-échantillonnage afin que les intersections se déplacent dans une grille rectiligne (*Figure .31*), ensuite on stocke les surfels résultants dans une seule LDI.

Cette réduction dégrade la qualité de la représentation en matière de géométrie et d'apparence, ce qui induit éventuellement des artefacts.

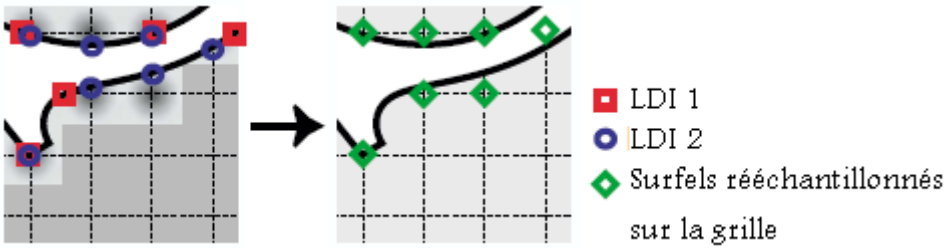


Figure .31. *Exemple de réduction 3 à 1.*

Pipeline de rendu des surfels

Cette étape consiste à projeter le niveau du bloc approprié à l'espace écran en utilisant la projection perspective :

- Élimination des blocs cachés: on parcourt l'arbre LDC, de l'haut vers le bas pour effectuer les tests de visibilité d'une façon hiérarchique efficace et on utilise d'abord le view-frustum culling sur les blocs (pour éliminer les blocs du LDC qui n'appartiennent pas à la pyramide de vue), puis on procède par le back-face culling grâce au cône de visibilité.
- Forward warping : il consiste à reprojeter les surfels de LDI à l'espace écran, en estimant la densité des surfels projetés dans l'image de sortie afin de contrôler la vitesse de rendu et la qualité de l'image reconstruite.
- Détecter les trous : on utilise la combinaison d'un z-buffer conventionnel avec une nouvelle méthode nommée «*splatting de visibilité* », permettant la survie uniquement d'un sous-ensemble réduit de surfels en éliminant le reste, en plus de la détection des trous.
- Filtrage de texture et d'ombrage : on filtre les couleurs de texture des surfels visibles en utilisant l'interpolation linéaire entre les niveaux appropriés du mipmap de surfel. Chaque surfel visible est éclairé, par le modèle d'illumination de Phong et la réflexion de mipmap.

- Remplissage des trous : cette étape finale a pour but de reconstruire l'image à partir des surfels visibles, en incluant le remplissage des trous et l'anti-aliassage. En général, la résolution de l'image de sortie et la résolution de z-buffer ne doivent pas être identiques.

Résultats

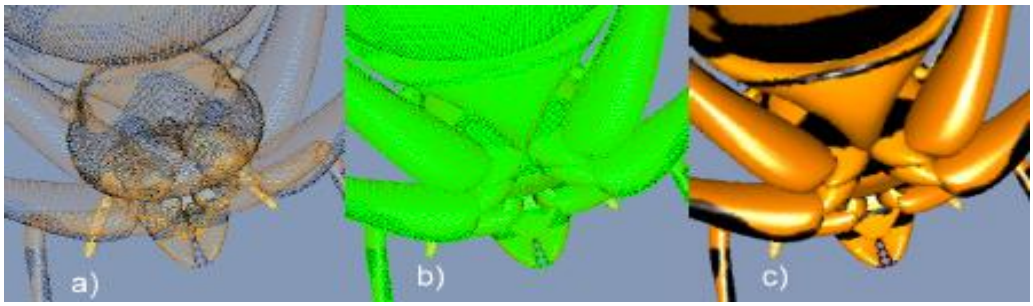


Figure .32. a)*surfel avec des trous.* ***b)*** *les trous détectés.* ***c)****image reconstruite par un filtre gaussien.*

- Les surfels sont idéals pour les modèles très complexes que ce soit sur le plan géométrique ou d'apparence.
- Diminution du coût de rendu grâce à l'anticipation de la texturation et la rastérisation à l'étape de prétraitement.
- La performance du rendu dépend du forward warping, l'illumination, et la reconstruction de l'image qui peuvent être accélérés par les architectures parallèles ou pipelinées.
- Le système surfel intègre naturellement l'anti-aliassage et le sur-échantillonnage.
- Le rendu par surfels est capable de produire des images de haute qualité à un taux interactif.
- Cependant, ce système ne fournit pas une définition mathématique de surface lisse contrairement à celle de Alexander et al. Dans « point set surface [ABCO+01] ».

1.7. Conclusion

Parmi toutes les méthodes de rendu des nuages de points que nous avons présenté, les méthodes par splats offrent sans aucun doute le meilleur compromis flexibilité/qualité/efficacité. Les composantes principales permettant d'atteindre ce compromis sont :

- pas de pré-calcul (flexibilité)
- anti-aliassage de la texture (qualité)
- éclairage par pixel (qualité)
- implantable sur les cartes graphiques (efficacité)

Bien que les cartes graphiques actuelles soient optimisées pour le rendu des triangles, Les GPU programmables permettent la mise en œuvre efficace de très haute qualité des techniques de rendu à base de points dont le rendement rapproche de celui des polygones.

Chapitre 2 : Évolution du rendu à base de points

2.1. Introduction

L'étape finale des applications interactives de traitement des points est le rendu efficace de ces derniers. En fait, le rendu de ces points dépend de la primitive géométrique choisie pour les représenter. Les sections qui suivent montrent les différentes techniques proposées dans la littérature pour produire à la fin du pipeline de rendu une image continue sans trou dans un temps fini avec les différents effets d'éclairement et un degré de réalisme convenable.

2.2. Points

Les points ont été proposés comme une primitive de rendu universelle par Levoy et Whitted [LW85]. Au lieu de dériver des algorithmes de rendu pour chaque primitive, ils proposent de subdiviser chacune en un ensemble de points suffisamment dense. Le rendu dans ce cas peut être alors fait par une technique adaptée.

Depuis, nous demandions de générer des images continues (sans trou) par le rendu d'un ensemble de points discret. Les méthodes utilisées pour remplir les trous entre les points doivent être trouvées. Ceci peut être fait par des techniques de reconstruction dans l'espace image (Grossman et Dally [GD98], Pfister et al [PZvG00]), ou par le ré-échantillonnage dans l'espace objet. Les techniques de cette dernière classe, consiste à ajuster dynamiquement le taux d'échantillonnage, afin que la densité des points projetés correspond à la résolution du pixel (*Figure .33*). Du fait que ceci dépend des paramètres du point de vue, l'échantillonnage doit être fait dynamiquement pour chaque image calculée.

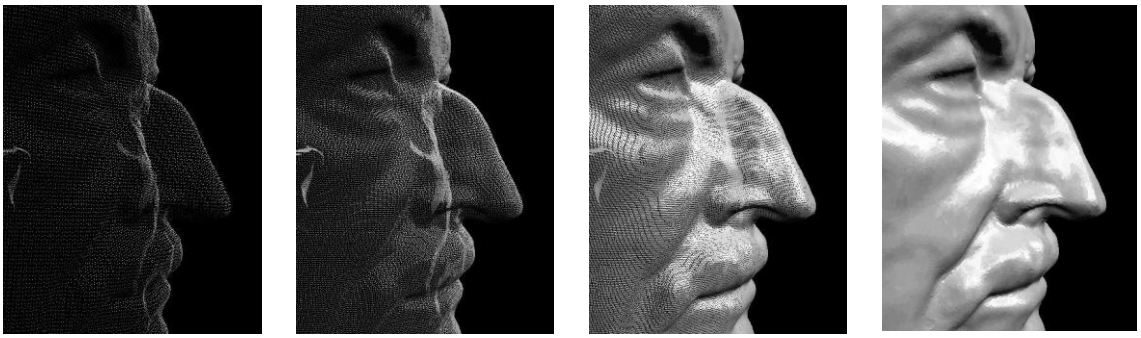


Figure .33. *Ajustement adaptatif et progressif du taux d'échantillonnage (de gauche à droite).*

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

En 2001, Stamminger et Drettakis [SD01] et Wand et al. [WFP+01] ont proposé en même temps deux techniques très similaires d'échantillonnage dynamique des maillages polygonaux.

La technique du randomized z-buffer [WFP+01] nécessite une phase de pré-calcul au cours de laquelle les facettes triangulaires seront triées spatialement dans un octree. Au moment du rendu, cet octree permet de former dynamiquement des groupes de faces ayant un facteur d'échelle proche.

Ensuite, une estimation du nombre total d'échantillons nécessaires à la visualisation est réalisée pour chaque groupe. Cette estimation est basée sur la distance entre l'observateur et le volume englobant du groupe et sur l'aire totale des triangles du groupe. Les échantillons sont ensuite choisis aléatoirement sur les triangles avec une fonction de densité de probabilité proportionnelle à la surface des triangles. Afin d'accélérer le rendu, les échantillons ponctuels sont stockés dans un cache.

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

A cause d'un échantillonnage aléatoire de la surface, ces méthodes sont plutôt adaptées à la visualisation d'objets complexes de type arbres ou autres végétaux. Dans le même article, Stamminger et Drettakis ont aussi proposé une méthode d'échantillonnage dynamique de terrain et des surfaces procédurales. La clé de cette technique est un schéma de raffinement adaptatif qui ne nécessite pas d'informations sur la connectivité entre les points. En revanche, les points initiaux doivent être uniformément répartis suivant une paramétrisation 2D de la surface. De plus, la représentation géométrique utilisée doit permettre la projection de n'importe quel point de l'espace sur la surface représentée.

Plus proche de la problématique de visualisation des nuages de points, Alexa et al. [ABCO+03] ont proposé d'associer à chaque point une approximation polynomiale de la surface définie dans le plan tangent. Cette approximation est calculée par la méthode des moving least squares. Au moment du rendu, ces approximations polynomiales sont dynamiquement échantillonnées de manière

à garantir une visualisation sans trou. Durant la phase de pré-calcul, en plus de l'évaluation des approximations polynomiales, il est nécessaire d'uniformiser la répartition des points ainsi que d'évaluer les domaines des polynômes afin de minimiser les risques de discontinuités et le sur-échantillonnage dus au chevauchement des approximations locales. En plus de la phase de pré-calcul et des éventuels problèmes de discontinuités, cette approche pose aussi les problèmes du coût de stockage et de l'interpolation des attributs de la surface comme la couleur dans le cadre d'objets texturés.

2.3. Splats

Zwicker et al. [ZPvBG01] ont présenté le rendu par splats (surface splatting) dans lequel ils associent une reconstruction utilisant la primitive disque ou ellipse dans l'espace objet, à un mélange lisse des splats dans l'espace image.

Nous pouvons ainsi qualifier ces méthodes comme étant intermédiaires entre les approches de reconstruction purement dans l'espace objet et purement dans l'espace écran. D'un côté, le chevauchement des splats dans l'espace objet est suffisant pour obtenir un rendu sans trou dans l'espace image. D'un autre côté, le rendu naïf de splats s'interpénétrant a pour conséquence des discontinuités de couleur dans l'image finale (*Figure .34*). Les splats doivent ainsi être mélangés entre eux lors du rendu. Alors, Zwicker et al. [ZPvBG01] ont proposé une méthode anti-aliassage anisotropique de haut qualité qui ressemble au filtrage anisotropique de texture EWA (Elliptical Weighted Average) par Heckbert [Hec89]. Chaque splat est associé à un noyau de filtre gaussien radialement symétrique, afin qu'une surface continue dans l'espace objet soit reconstruite par la moyenne pondérée des propriétés des splats superposés (couleur, normale, etc).

Les combinaisons de ces noyaux de reconstruction dans l'espace objet avec un filtre de bande limité dans l'espace image, donne lieu à un framework EWA splatting de haut qualité (*Figure .34*). Si les filtres dans l'espace image et l'espace objet sont gaussiens, et si la projection est localement (par splat) approximé par une transformation affine, alors ces deux filtres peuvent être combinés en un seul filtre gaussien, ce que permet une implémentation tout à fait efficace.

Néanmoins, l'implémentation purement logicielle du rendu des modèles complexes devient très couteuse en terme de temps de calcul, et atteint dans ce cas a un taux de 1M splats/seconde sur les machines actuelles.

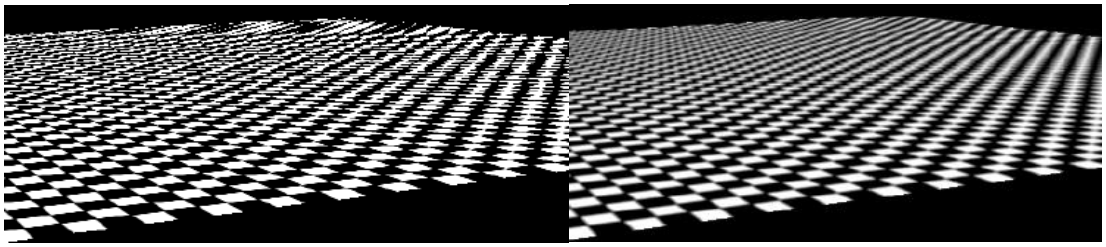


Figure .34. *rendu du damier sans filtrage (à gauche), rendu du damier avec le filtrage EWA (à droite).*

2.4. Accélération du rendu

Botsch et al. [BWK02] ont proposés l'utilisation d'une représentation hiérarchique pour un rendu plus efficace. A partir d'une grille régulière binaire représentant implicitement les positions des points de la surface, un octree est construit en regroupant récursivement les cellules huit par huit. Après l'élimination des branches vides et un encodage entropique, le coût de cette représentation devient optimal. Le rendu peut être finalement fait soit par point, ou par splat pour chaque cellule feuille non vide. Cette solution a permis d'avoir un taux de 5Msplat/seconde, cependant cette dernière est encore complètement logicielle, et elle suscite une surcharge dans le CPU, ce qui le bloque des autres tâches de traitement géométrique usuel.

2.5. GPUs programmables

Durant les dernières années, l'augmentation des performances et la programmabilité des cartes graphiques modernes (Lindholm et al [LKM01], Mark et al [MGAK03]), a déclenché le développement des méthodes de splatting basées matériel.

Une méthode de rendu hiérarchique basée sur une structure arborescente pré-calculée de sphères englobantes (QSplat) a été proposée par Rusinkiewicz et levoy [RL00].

Du fait que le parcours de cette arborescence n'était pas tellement efficace pour bien occuper les GPUs, Dachsbacher et al [DVS03], ont proposé une linéarisation de la structure qui correspond au parcours en largeur d'abord. Ceci a permis le transfert de la liste des points dans la mémoire des GPUs en une seule fois, en plus l'implémentation de la sélection LOD dans le vertex shaders des GPUs, résultent une diminution de la charge des CPUs et un taux impressionnant de plus de 50 Msplat/seconde a été atteint (*Figure .35*). L'inconvénient de cette approche est que la primitive de rendu utilisée pour chaque splat est seulement des carrés non-filtré dans l'espace image, par conséquent la qualité pourrait être améliorée considérablement par l'utilisation du filtrage gaussien et les splats elliptiques.

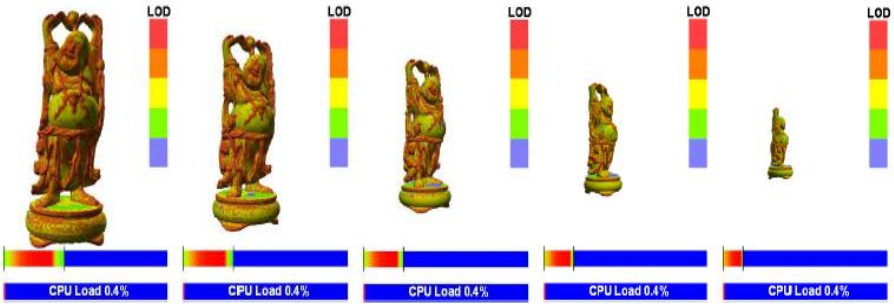


Figure .35. *Diminution de la surcharge du CPU grâce à la linéarisation de l'hiérarchie, et au contrôle des niveaux de détails par le GPU.*

Pour enlever les discontinuités d'éclairement et avoir un rendu lisse, un noyau de filtre gaussien devrait être associé à chaque splat aussi dans le cas de rendu accéléré par les GPUs programmable. La difficulté est que lorsque deux splats sont projetés à un même pixel, seulement les splats étroitement recouverts devraient être mélangés, cependant dans les autres cas, lorsque la distance Z (de profondeur) entre les splats est supérieure à un certain seuil, le splat le plus proche à la caméra doit remplacer ceux derrière. Une implémentation de cette fonctionnalité nécessite le test de profondeur flou (*Figure .36*) ou un test d'opacité plus général [Car84].

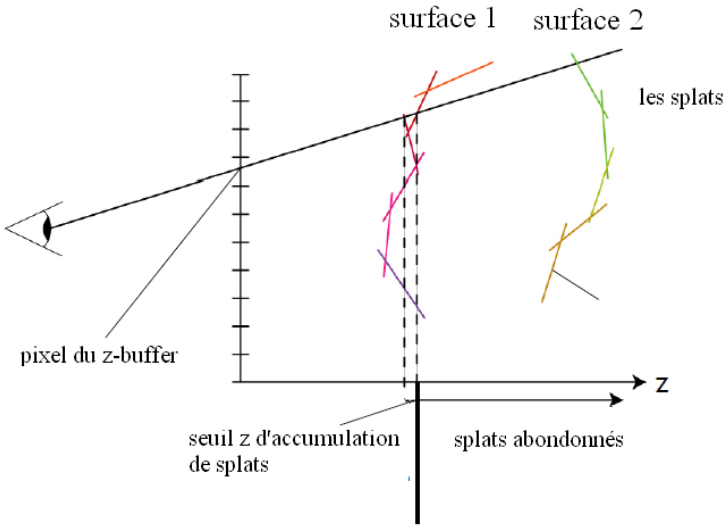


Figure .36. *Z-buffer étendu ou flou.*

Malheureusement, les deux techniques ne sont pas disponibles sur les GPUs actuels, et au lieu de ça, des multiples passes de rendu doivent être faites.

- La première passe est appelée **splatting de visibilité,** qui consiste en le remplissage du z-buffer par le rendu (sans la lumière) de tous les objets légèrement éloignés du point de vue par ϵ (*Figure .37*). Cette passe de rendu n'affecte que le tampon de profondeur et aucun calcul d'éclairage ou autres n'est effectué.

- La deuxième passe rend tous les splats avec le mélange gaussien, mais sans altérer le z-buffer, ce qui mélange uniquement ces splats qui se diffèrent par moins de ϵ dans la profondeur [RL00]. Ce processus accumule une sommation pondérée des couleurs $(\sum_i \alpha_i rgb_i, \sum_i \alpha_i)$ dans chaque pixel RGBA.

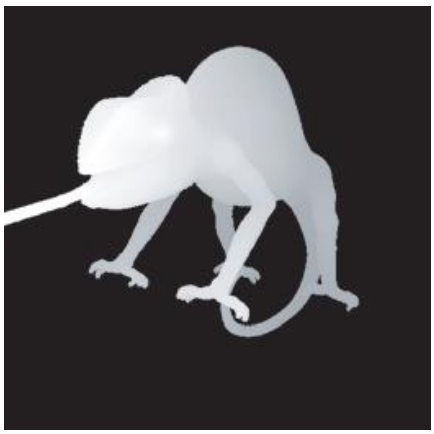


Figure .37. *Résultat de la passe splatting de visibilité.*

L'utilisation du pixel shaders des GPUs actuels va permettre la rastérisation des splats elliptiques en rendant uniquement un sommet par splat. Le Calcul de la taille projetée dans le vertex shader déclenche la rastérisation d'un carré dans l'espace image. Le fragment shader traite chacun des pixels et construit la forme elliptique par l'exclusion des pixels dehors l'ellipse. Une implémentation utilisant des splats circulaires dans l'espace objet et les deux passes du filtrage gaussien a été présenté par Botsch et Kobbelt [BK03], et ça a permis d'atteindre un taux de 10 Msplat/seconde.

2.6. Eclairement plat, Gouraud et Phong des nuages de points

La plupart des méthodes de rendu à base de points, utilisent un vecteur normal constant pour éclairer chaque splat individuellement, menant à un résultat très comparable à l'éclairement plat dans le cas des maillages triangulés. L'utilisation du mélange par le noyau gaussien de reconstruction, permet de lisser les artefacts d'éclairement, d'où le résultat devient comparable à l'éclairement de Gouraud. L'utilisation des champs de normale non constants et l'illumination par pixel, améliorent significativement la qualité visuelle ([ZPvBG01], [KV01] et [KV03]).

Dans le cas des maillages triangulés, l'éclairement de Phong atteint une qualité de rendu supérieure en calculant une illumination par pixel basée sur l'interpolation des vecteurs normaux des sommets d'un triangle dans l'espace objet. Ce concept n'est pas directement applicable pour le rendu à base de points, à cause de l'impossibilité d'accéder aux vecteurs normaux des splats voisins (pas d'information explicite de connectivité). En vue d'atteindre une illumination par-pixel tout en maintenant la simplicité de la représentation à base de points,

Botsch et al [BK04] associent un champ de normale linéaire à chaque splat individuellement. Ces champs de normale sont dérivés par les moindres carrés pour un ensemble dense donné de normales. L'illumination de Phong pour les splats (**Phong Splatting**) peut être simplement et efficacement implémentée sur les GPUs, ce qui résulte une haute qualité visuelle, comparable à l'illumination de Phong pour les maillages triangulés (*Figure .38*).

Figure .38. *Rendu avec (de gauche à droite) : splats naïfs (plat), splat avec un filtre gaussien (Gouraud), phong splatting (Phong).*

2.7. Conclusion

Les représentations à base de points se sont révélés être une alternative intéressante aux maillages polygonaux dans plusieurs applications. En particulier lorsque des modèles très complexes ou lourds doivent être traités où une topologie de surface adéquate n'est pas forcément nécessaire, en plus, la simplicité des représentations basées points conduit à des algorithmes plus simples et plus efficaces.

Malgré tout, et du fait que les cartes graphiques actuelles sont fortement optimisées et spécialisées pour le rendu des triangles, les points ou les splats ne peuvent toujours pas atteindre les triangles en terme de performance effective de rendu.

Dans l'avenir, de nouveaux types d'architectures matérielles graphiques pourraient exploiter les avantages particuliers des points 3D en tant que primitive de rendu générale en s'appuyant sur des techniques d'échantillonnage adaptatives dans l'espace objet afin de réduire la redondance de la représentation à base de points et le taux de réécriture des pixels sur l'écran.

Chapitre 3: La texturation d'objets 3D à base de points

3.1. Introduction

On aurait pu modéliser les propriétés micro-géométriques et celles du matériau pour avoir un vrai réalisme, mais ceci à un coût très élevé. L'approche la plus pratique est de remplacer la complexité géométrique et les matériaux par une texture qu'on plaque sur l'objet 3D.

Pour définir le mot texture, nous proposons les définitions suivantes:

- Le Petit Larousse : une représentation graphique d'une matière, d'une surface, dont le rendu en volume est effectué par placage sur un modèle en trois dimensions. (L'effet de matière [bois, pierre, etc.] est permis grâce à des procédés de numérisation d'images photographiées ou dessinées).
- Apparence visuelle et au toucher d'une surface.
- Une image utilisée pour définir les caractéristiques d'une surface.
- Une image multidimensionnelle qui est plaquée dans un espace multidimensionnel.
- Caractérisation de l'apparence fine des objets 3D à base d'images.

Une texture peut avoir de différentes dimensions :

- 1D : elle peut être vue comme une fonction d'un seul paramètre, elle est utile pour des gradients de couleur.
- 2D : elle peut être vue comme une fonction de deux paramètres, elle englobe les images ou les cartes 2D en général (carte de couleur, de réflexion spéculaire, de perturbation de normales, de transparence, d'ombre, de reliefs ou autre).
- 3D : elle peut être vue comme une fonction de trois paramètres, elle pourrait être un volume ou une distribution spatiale en général.

L'origine d'une texture peut être, un artiste, un générateur (logiciel), ou la combinaison des deux.

La texturation peut être considérée comme le processus de répéter un motif, une image dans toutes les directions de la surface d'un objet 3D.

3.2. Texturation

3.2.1. Définition

Il s'agit de trouver la correspondance entre une image 2D un modèle 3D, et de copier les pixels de la texture (appelés texels : TEXture ELement) aux pixels de l'objet selon cette correspondance (*Figure .39*).

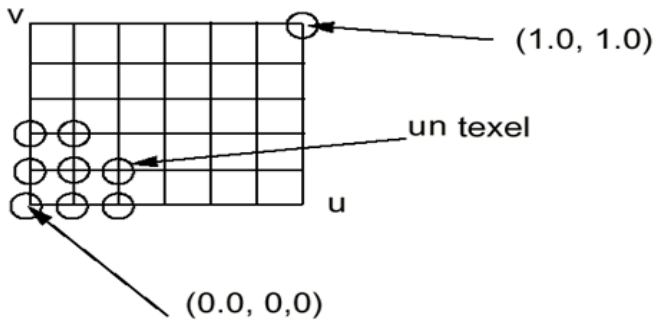


Figure .39. *Les texels (TEXture ELement).*

Le processus consiste mathématiquement à trouver les coordonnées bidimensionnelles (U et V), appelées les coordonnées de texture, en fonction des coordonnées cartésiennes de l'objet tridimensionnel (X , Y et Z).

OpenGL dispose d'une API permettant de couvrir tout le processus de texturation (y compris la génération automatique des coordonnées de texture « glTexGen»), mais uniquement pour les modèles polygonaux.

3.2.2. Fonctions de placage (UV Mapping)

Consiste à faire correspondre une image 2D à un objet 3D quelconque, nécessite généralement une transformation non-linéaire et ceci est un vrai problème. Dans le cas du plan 3D, le problème est résolu par une transformation affine simple (translation, rotation, changement d'échelle ou la combinaison des trois – voir *Figure .40*).

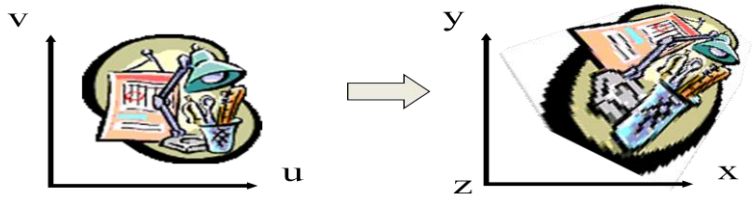


Figure .40. *Transformation affine de la texture de l'espace texture à l'espace objet.*

En plus ces transformations génèrent des aliassages qui proviennent de la correspondance texel/pixel.

Les fonctions de placage primordiales sont :

- ❖ placage affine : ceci comprend les transformations de : translation, rotation, mise à l'échelle ou cisaillement.
- ❖ placage bilinéaire : il préserve les lignes horizontales et verticales, au contraire, les lignes diagonales seront déformées.
- ❖ placage perspective : il préserve les lignes de toute direction, mais au détriment des équidistances.

Préservation / placage	**Lignes**	**Parallèles**	**Équidistances**	**Inversibilité**
Affine	Oui	Oui	Oui	Oui
Bilinéaire	Sauf diagonales	Non	Sauf diagonales	Non
perspective	Oui	Non	Non	Oui

Une approche de placage à deux passes, a été introduite par Bier et Sloan [BS86]. Elle consiste à plaquer la texture sur un objet intermédiaire englobant l'objet cible, dans la première passe, puis plaquer cet objet intermédiaire sur l'objet cible dans la deuxième passe. L'objet intermédiaire peut être soit : un plan, une sphère, un cylindre ou un cube.

Ces quatre fonctions de placages de base peuvent servir chacune, une partie d'un objet 3D plus complexe.

A. Placage plan

Dans ce cas le plan UV peut correspondre simplement aux plans XY, YZ, XZ ou un plan personnalisé (*Figure.41*).

B. Placage sphérique

Dans ce cas, le calcul des coordonnées UV dépend des coordonnées sphériques correspondant aux coordonnées cartésiennes (*Figure.41*).

C. Placage cylindrique

Dans ce cas, le calcul des coordonnées UV dépend des coordonnées cylindriques correspondant aux coordonnées cartésiennes (*Figure.41*).

D. Placage cubique

Dans ce cas, on englobe l'objet d'un cube, donc 6 plans (*Figure.41*).

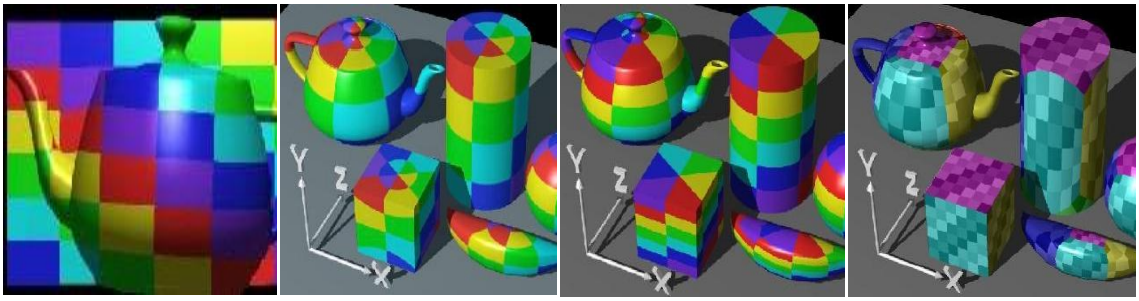


Figure.41. *Placages (de gauche à droite) : plan, sphérique, cylindrique et cubique.*

E. Autres placages

- Placage de relief (bump mapping) : dans ce cas, on ne plaque plus des couleurs sur la surface, mais plutôt des normales, ce qui permet de définir les endroits lisses et aigus de la surface sous-jacente (*Figure.42*).
- Placage d'environnement (environment mapping) : il est utilisé pour approximer les effets de réflexions globales comme la réflexion de la scène sur un objet complètement réfléchi (effet de miroir), voir *Figure.42*.

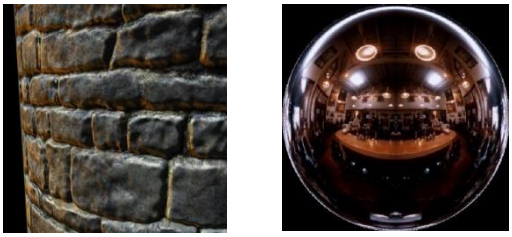


Figure.42. *Placages (de gauche à droite) : de relief et d'environnement.*

3.3. Texturation des nuages de points

Elle consiste au calcul des coordonnées de texture, et ceci pour l'ensemble des splats. Rappelons que la primitive splat consiste en un cercle ou ellipse coloré, plus un filtre gaussien appelé le filtre de reconstruction, comme illustré dans *Figure .43*.

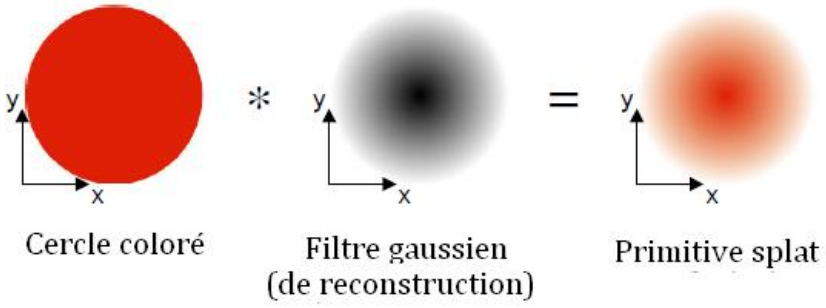


Figure .43. *La primitive splat = cercle coloré + filtre gaussien de reconstruction.*

Les équations de calcul des fonctions de placage de textures dans le cas plan, sphérique et cylindrique sont comme suit :

A. Placage plan

$$U = \frac{X_i - X_{MIN}}{X_{ECART}}, \qquad V = \frac{Y_i - Y_{MIN}}{Y_{ECART}}, \qquad X_{ECART} = X_{MAX} - X_{MIN},$$
$$Y_{ECART} = Y_{MAX} - Y_{MIN}$$

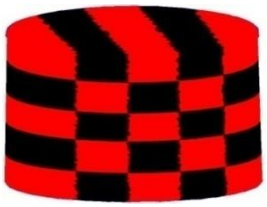
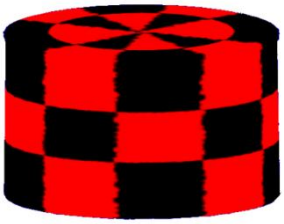
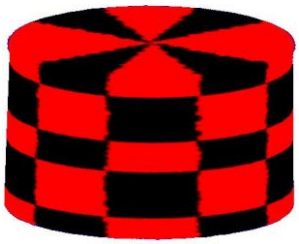
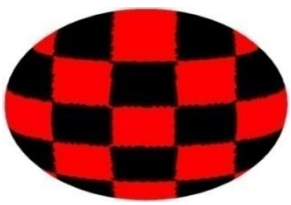
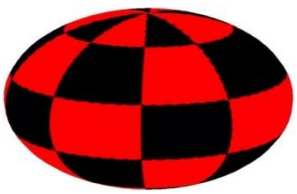
B. Placage sphérique

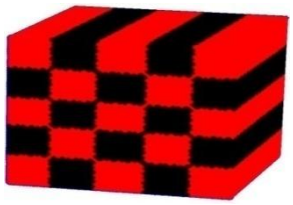
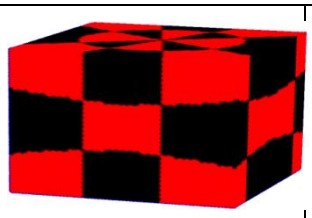
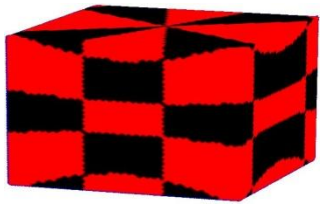
$$U = \frac{\text{acos}\left(\frac{X_i}{\sqrt{X_i^2 + Z_i^2}}\right)}{2\pi}, \qquad V = \frac{\text{acos}\left(\frac{Y_i}{\sqrt{X_i^2 + Y_i^2 + Z_i^2}}\right)}{2\pi}$$

C. Placage cylindrique

$$U = \frac{\text{acos}\left(\frac{X_i}{\sqrt{X_i^2 + Z_i^2}}\right)}{2\pi}, \qquad V = \frac{Y_i - Y_{MIN}}{Y_{ECART}}, \quad Y_{ECART} = Y_{MAX} - Y_{MIN}$$

Le résultat de l'application des trois types de plage de texture sur trois modèles géométriques basiques (le cube, la sphère et le cylindre) utilisant la primitive splat, sont comme montre le tableau suivant :

Placage / Objet	Plan	Sphérique	cylindrique
Cylindre			
Sphère			

Cube			

3.4. Filtrage de texture

3.4.1. Introduction

C'est la méthode employée pour déterminer la couleur de texture à appliquer à un pixel de la surface, en combinant les couleurs des tous près texels (*Figure .44*) pour le but d'avoir un placage de texture le plus continu possible (anti-aliassage). En bref, on mélange les pixels de texture ensemble en se basant sur le voisinage. Une autre forme pour le filtrage de texture s'appelle le lissage de texture. Il y a beaucoup de méthodes de filtrage de texture, qui se diffèrent entre eux en matière de qualité et de complexité de calcul. Puisque le filtrage de texture est une tentative de trouver une valeur à un point donné à partir d'un ensemble d'échantillons discrets aux points voisins, c'est une forme d'interpolation.

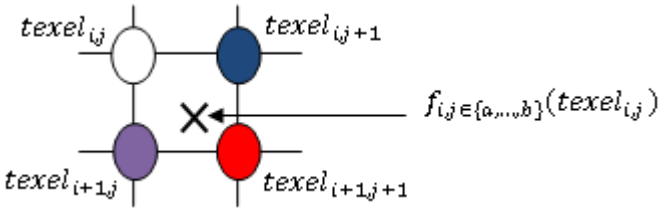


Figure .44. *Calcul de la couleur d'un point donné en fonction des texels voisins.*

3.4.2. Méthodes de filtrage

Ci-après, une liste des méthodes de filtrage de texture les plus usuelles, dans l'ordre croissant de la qualité d'image.

A. Filtrage avec interpolation du proche- voisin

L'interpolation du plus proche-voisin est la méthode la plus rapide qui consiste simplement à utiliser et à recopier la couleur du texel le plus proche dans le pixel en question. Bien qu'elle est rapide, elle présente plusieurs artefacts tels que, le blocage de texture durant d'agrandissement et l'aliassage pendant la diminution.

B. Filtrage avec interpolation des proches- voisins avec Mipmapping

Cette méthode aussi utilise une interpolation du plus proche voisin, mais elle utilise le mipmapping. D'abord le niveau du mipmap le plus proche est choisi en fonction de la distance, puis le centre du texel le plus proche est échantillonné

pour obtenir la couleur du pixel. Cela réduit l'aliassage de manière significative, mais cela n'ôte pas le blocage de texture.

C. Filtrage bilinéaire

Dans cette méthode, les quatre plus proches texels du centre des pixels sont échantillonnés (au niveau du plus proche mipmap), et leurs couleurs sont combinées par la moyenne pondérée selon la distance. Cela supprime le le blocage de texture observé au cours de l'agrandissement, car il y a maintenant un gradient de changement de couleur d'un texel à l'autre, au lieu d'un saut brutal. Le filtrage bilinéaire est presque toujours utilisé avec le mipmapping. La formule de calcul est la suivante :

$$couleur = texel_{i,\ j} + \left(texel_{i,\ j+1} - texel_{i,\ j}\right) * U + \left(texel_{i+1,\ j} - texel_{i,\ j}\right) * V + \left(texel_{i+1,\ j+1} - texel_{i+1,\ j} - texel_{i,\ j+1} + texel_{i,\ j}\right) * U * V$$

$$U = u - u_{i,\ j}$$

$$V = v - v_{i,\ j}$$

D. Filtrage trilinéaire

Le filtrage trilinéaire est un remède à un artefact commun vu dans le mipmap bilinéaire des images filtrées qui est le changement brusque de la qualité aux frontières, lors de la bascule d'un niveau de mipmap à l'autre. Le filtrage trilinéaire résout ce problème en faisant une recherche de texture plus un filtrage bilinéaire entre les deux niveaux de mipmap les plus proches (un de haute résolution, et l'autre de basse résolution), puis on filtre le résultat par une interpolation linéaire. Il en résulte une dégradation de la qualité de texture en matière de lissage et ceci quand la distance depuis le point de vue augmente. Bien sûr, plus proche que le niveau 0, il existe qu'un seul niveau mipmap disponible, et l'algorithme revient à un filtrage bilinéaire.

E. Filtrage anisotropique

Le filtrage anisotropique est le filtrage de la plus haute qualité disponible dans les cartes graphiques actuelles. Il a évolué parce que les deux filtrages bilinéaire et trilinéaire échantillonnent un carré de la texture, ce qui n'est correct que si la caméra se penche sur la texture de front. Il en résulte l'effet de flou lorsque la surface texturée est à un angle oblique. Le filtrage anisotropique corrige ce dernier par l'échantillonnage dans la forme correcte de trapèze selon un angle de vue. Les échantillons obtenus sont ensuite filtrés par un filtre trilinéaire pour générer la couleur finale.

3.4.3. Pré-filtrage de texture

Il se trouve, en général, que les opérations induites par le calcul d'interpolation du filtrage de texture, sont très couteux en matière de temps de

calcul, par conséquent, il s'avère judicieux de les faire en hors ligne (off-line), c'est-à-dire en temps différé, afin de permettre un rendu en temps réel.

Le pré-filtrage est utilisé pour limiter la bande de la texture au taux d'échantillonnage disponible. Le filtrage bilinéaire et trilinéaire utilisant les pyramides de mipmap qui sont souvent les plus utilisés actuellement.

3.5. Mipmapping

Le mipmaping est une approche utilisée dans le contexte du placage de texture, où la texture originale de haute résolution est mise à l'échelle et pré-filtrée dans plusieurs résolutions en hors ligne (off-line) avant d'être appliquer sur la surface en temps-réel (*Figure .45*). Le but du mipmapping est d'enlever l'aliassage lorsqu'on s'éloigne d'une surface texturée. Le niveau de détail des textures est adapté à la distance entre l'objet et la caméra. Ainsi, un objet proche affichera des textures en haute résolution tandis qu'un objet lointain se verra attribuer une texture de faible taille. Différents niveaux de détails ou niveaux de mipmap, peuvent être choisis.

Le mipmapping consiste à envoyer au GPU des échantillons de texture de résolutions décroissantes qui seront utilisés à la place de la texture originale, en fonction de la distance du point de vue à l'objet texturé, et du niveau de détails nécessaire. Le GPU n'a alors plus qu'à appliquer les bonnes textures sur les bons objets suivant leur éloignement, en réadaptant la texture chaque fois que l'objet se rapproche. La texture utilisée lors du rendu sera alors celle dont la résolution est la plus proche de celle de l'objet sur l'image projetée.

Par exemple, à partir d'une image d'une taille de 256x256 pixels seront produits les mêmes images aux résolutions de 128x128 pixels, 64x64, 32x32, 16x16, 8x8, 4x4, 2x2 et 1x1. Si la taille de l'objet sur l'image projetée à l'écran est de 30x30 pixels, la texture utilisée sera alors celle de résolution 32x32 pixels. Le filtrage trilinéaire permet d'éviter de voir les « sauts » lors du passage d'une texture à l'autre, en faisant une transition progressive. Le mipmapping seul est bien adapté à une texture perpendiculaire à l'observateur. Mais pour un sol par exemple, un filtrage anisotropique est nécessaire. Bien que la création de mipmap demande plus de mémoire vidéo (environ 33% de plus), cette technique permet de réduire les artefacts dûs à des filtrages successifs appliqués sur la texture lorsque l'objet est éloigné. La diminution du nombre de texels à traiter et des opérations de filtrages en temps réel de la texture, permet également un rendu plus rapide de l'image. L'inconvénient de cette technique est qu'elle nécessite plus d'espace mémoire de stockage (un tier additionnel de la texture originale).

Figure .45. *Pyramide des images du mipmap.*

3.6. Aliassage

3.6.1. Introduction

L'aliassage (en anglais aliasing), est l'apparition sur une image des marches d'escalier, ou de moirés, ou aussi c'est la disparition de petits objets. Le système visuel humain est particulièrement sensible aux transitions brusques sombre/lumineux, donc il s'attache à décortiquer les marches d'escaliers et moirés (alignement des marches d'escalier dans les régions rayées) qui n'ont pourtant aucune signification. L'œil voit des motifs qui n'existent pas, d'où la terminologie (alias qui signifie autre en latin).

Lors du placage de texture, on aura des artefacts si la texture contient des hautes fréquences, ces artefacts sont appelés l'aliassage de texture. L'aliassage a été étudié en détail dans le domaine du traitement de signal, où la théorie de base a été bien expliquée.

Ces théories ont été reprises par les infographistes pour résoudre le problème d'anti-aliassage, et des bons résultats ont été trouvés.

L'aliassage provient de la nature numérique de nos ordinateurs, cependant l'image et les modèles géométriques dans la nature ont un domaine spatial continu. Pour but de simplifier l'étude, nous commençons d'abord par un signal unidimensionnel. Pour avoir un signal numérique (discret), on doit d'abord échantillonner le signal analogique (continu), puis le quantifier. Et lors de l'affichage on reconstruit le signal continu à partir de son équivalent discret. Maintenant si on ne remplit pas certaines conditions dans ce processus on aura un signal reconstruit avec des artefacts, autrement dits, l'aliassage (*Figure .46*).

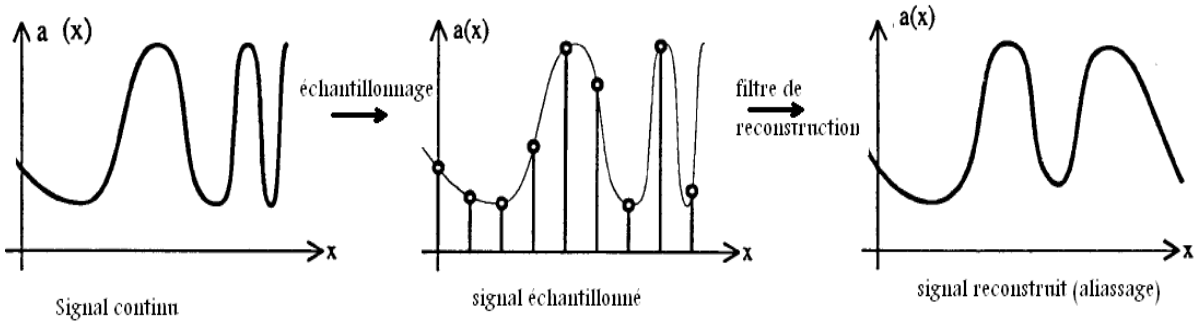


Figure .46. *Aliassage dû à l'échantillonnage.*

Selon le théorème de Shanon et Nyquist, pour pouvoir reconstruire un signal à partir de ses échantillons, on doit satisfaire deux conditions:

1. le signal doit être à bande limitée (le signal est nul au delà d'une fréquence donnée), ce qui supprime les hautes fréquences (*Figure .47*).
2. le signal doit être suffisamment échantillonné (fréquence d'échantillonnage supérieure ou égale deux fois la fréquence maximale), voir *Figure .47*.

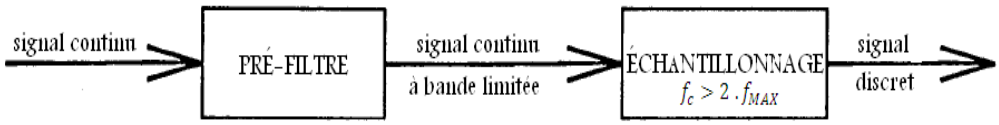


Figure .47. *Anti-aliassage idéal par un pré-filtrage suivi par un échantillonnage.*

Pour le pré-filtrage, on utilise un filtre passe bas, qui peut être, entre autre, un filtre B-spline, un filtre gaussien ou un filtre sinc.

3.6.2. Filtre de ré-échantillonnage

Dans le contexte du placage de texture, les grilles d'échantillonnage de la texture originale, et celle plaquée ne sont pas les mêmes, et du coup, l'image doit être filtrée par un filtre (appelé le filtre d'échantillonnage) qui dépend du type du placage appliqué. Si le placage est une transformation affine, l'échantillonnage idéal pourrait être calculé utilisant la convolution, sinon un post-filtrage est préféré (un échantillonnage suivi d'un filtrage).

Pour le cas du rendu par splat, le filtre de reconstruction gaussien dans l'espace objet ne suffit pas probablement pour remplir un pixel dans l'espace écran, ce qui résulte un aliassage. Pour pallier à ce problème Zwicker et al. [ZPvBG01] ont rajouté un deuxième filtre (passe bas) à celui ce reconstruction afin que les splat entrelacés se mélangent d'une façon plus équitable ce qui enlève les discontinuités et l'aliassage dans l'espace écran (*Figure .48*).

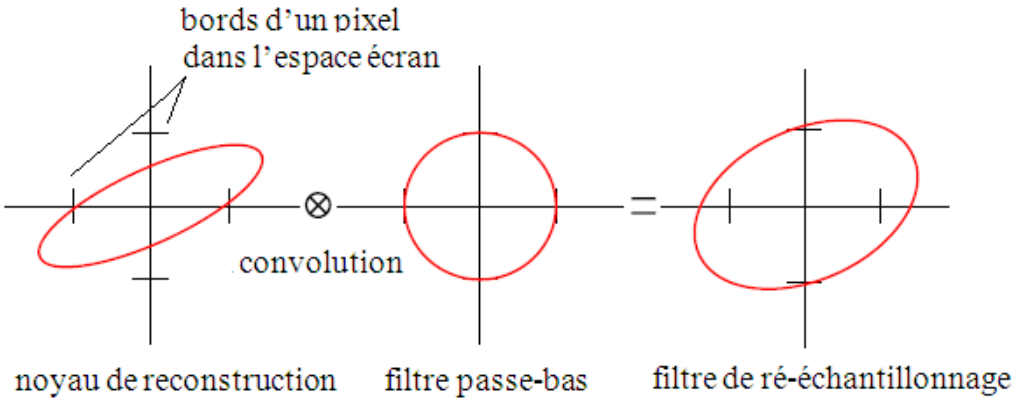
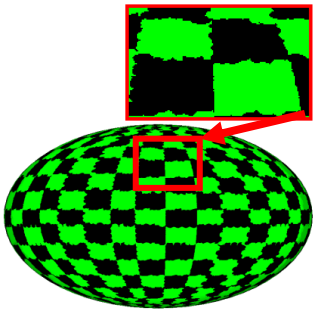
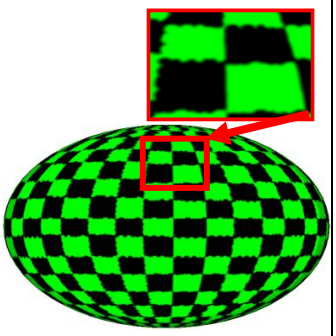
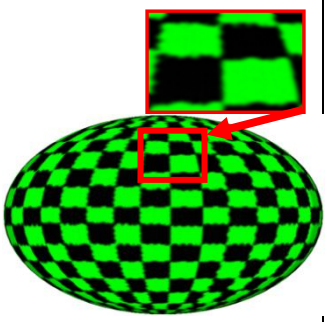
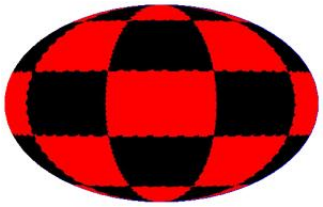
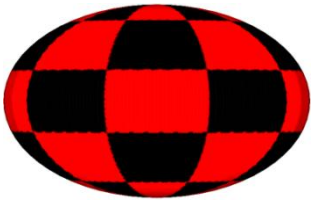
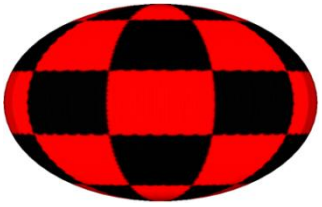


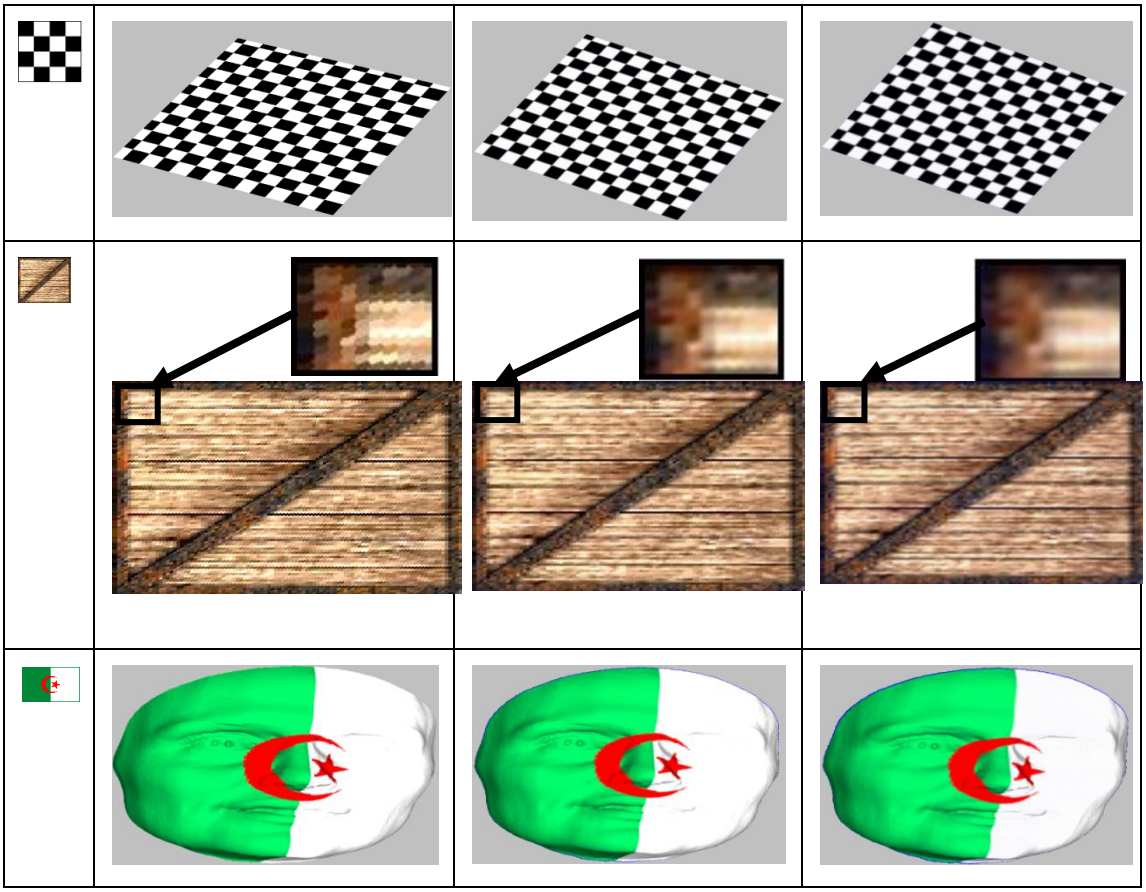
Figure .48. *Filtre de ré-échantillonnage = filtre de reconstruction + filtre passe bas.*

3.7. Résultats expérimentaux

3.7.1. Comparatif entre les différentes primitives de rendu

Le tableau ci-dessous compare la texturation d'un nuage de points utilisant les primitives géométriques : splat opaque, splat avec un filtre de reconstruction gaussien et finalement splat avec un filtre de ré-échantillonnage.

Motif	Splat opaque	Splat + filtre gaussien	Splat+ filtre de ré-échantillonnage
			
			
			



3.8. Conclusion

Les résultats présentés ci-dessus de la texturation d'un nuage de points montrent bien que le rendu des points par des splats associés à un filtre gaussien de reconstruction a atteint une haute qualité en matière d'absence de trous mais le plus important qui est la continuité de la texture plaquée et par conséquent ceci a permis d'enlever l'aliassage clairement apparu dans le cas des splats opaques.

Cependant les objets texturés utilisant uniquement la primitive splat associé à un filtre de reconstruction présentent des hautes fréquences où l'ajout d'un filtre passe bas à cette primitive (qui résulte un filtre de ré-échantillonnage) a permis d'enlever ces hautes fréquences en rendant la texture plaquée plus continue et un peu floue.

Dans le dernier chapitre, nous abordons la deuxième problématique liée aux nuages de points qui est la déformation interactive de ces derniers.

Chapitre 4: la déformation interactive de nuage de points

4.1. Introduction

Les applications interactives de modélisation et de conception nécessitent la capacité de manipuler les objets par des déformations lisses, celles-ci incluent, la flexion, la torsion, l'étirement et la compression de la surface de l'objet. Afin que les artistes et les concepteurs de modèles 3D puissent créer des modèles complexes et détaillés, l'utilisation des outils de déformation interactifs adaptés est primordiale à cet effet.

Bien sûr nous mettons bien l'accent sur l'interactivité de ces outils, parce qu'avoir un temps de réponse réduit est crucial et déterminant pour juger l'utilité de tels outils, qui seront finalement utilisés par les usagers.

La méthode proposée par Sedeberg et Parry [SP86] intitulée FFD : Free Form Deformations (les déformations libres) est considérée comme la méthode référence dans le contexte de déformation des formes.

Dans ce chapitre, nous proposons une contribution qui consiste en le développement des déformations libres sur les objets à base de points avec une accélération GPU, ce qui a permis d'avoir une nette amélioration en termes de temps de calcul.

4.2. Déformations libres simples

Les déformations libres sont des déformations d'espace. En effet, l'idée de base derrière les déformations libres est très simple à comprendre, elle consiste à envelopper l'objet à déformer dans un espace et à déformer l'espace. Les déformations faites sur l'espace sont ensuite appliquées à l'objet enveloppé ce qui le déforment. Il faut aussi noter que cette classe de déformations libres est une déformation dite indirecte : on ne déforme pas directement l'objet, mais on se sert plutôt d'un outil de déformation intermédiaire (les points de contrôle) afin de parvenir nos fins.

Inspirés en partie par les déformations non-linéaires de Barr [Bar84], Sederberg et Parry [SP86] proposent une façon simple et conviviale de déformation des modèles, appelée : les déformations libres. Le procédé se résume en trois étapes principales, comme suit:

1. Créer un volume ayant la forme d'un parallélépipède autour de l'objet et imposer des coordonnées locales à chaque point de l'objet à déformer.

2. Imposer une grille de points de contrôle sur le parallélépipède.

3. Déformer l'objet en bougeant les points de contrôle (voir *Figure.49*).

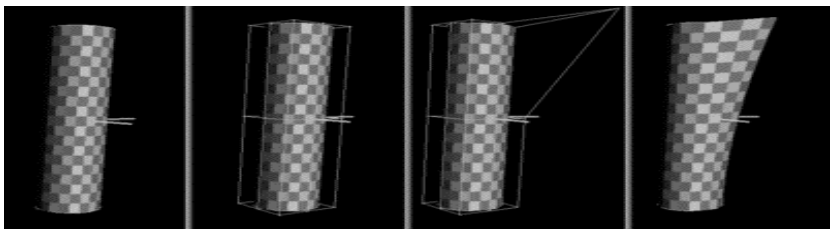


Figure.49. *Les 3 étapes du FFD sur un cylindre.*

Sachons que les coordonnées des points de l'objet 3D seront paramétrisées utilisant une fonction de mélange basée sur les polynômes de Bernstein trivariés. Comme dans le cas des surfaces paramétriques, les transformations libres ont les mêmes avantages et inconvénients que les courbes de Bézier, à savoir : les déformations sont globales, les points déformés se trouvent toujours à l'intérieur de l'espace défini par les points de contrôle, et enfin la déformation est indirecte (on ne peut manipuler directement les points de l'objet).

Il est possible de choisir une autre base de fonction de mélange et ainsi bénéficier des avantages et inconvénients de la base choisie.

Notons qu'il est possible avec cette méthode de faire des transformations locales sur un objet en ne paramétrisant qu'une partie de ce dernier. Cependant, il faut porter très attention à la continuité de la surface voulue, car les parties non-paramétrisées de l'objet ne bougeant pas lors de la déformation, il est alors très facile, si on ne fait pas attention, d'introduire des discontinuités de surface sur notre objet.

4.3. Déformations libres directes

Les déformations libres directes ont été introduites par Hsu et al. [HHK92] et elles permettent de modifier directement un ou plusieurs points de l'objet à la fois. Ainsi, les déformations sont plus précises et l'interface est plus simple à manipuler pour l'usager.

Dans leur modèle mathématique, Hsu et al. utilisent les B-splines comme fonction de mélange plutôt que le polynôme de Bernstein trivarié. Ce choix (grâce à la continuité C^2 entre 2 B-splines adjacentes), va leur permettre de représenter le parallélépipède de contrôle comme plusieurs volumes de B-splines et ainsi bénéficier d'un contrôle local pour leurs déformations. Comme on le sait, les B-splines ont cependant le désavantage de ne pas remplir complètement le volume englobant déterminé par les points de contrôle mais les auteurs pallient à ce problème en donnant une multiplicité 3 aux points de contrôle qui sont sur la bordure du parallélépipède de contrôle.

Simplement, le problème des déformations libres directes revient à solutionner l'équation suivante afin de trouver ΔP.

$$\Delta Q = B\Delta P$$

Où ΔQ est la matrice de différence de position entre 2 points de l'objet (1 x 3). B est la matrice des 64 fonctions de mélange (1 x 64) et ΔP est la matrice de différence de position entre les points de contrôle associés aux points de l'objet (64 x 3).

Afin de résoudre ce système matriciel, il suffit d'utiliser la méthode de pseudo-inverse afin d'obtenir B+, une matrice inverse de B dans le sens des moindres carrés. Dans le cas du mouvement d'un seul point à la fois, la solution est simple. Cependant il faut faire attention lorsqu'on traite plusieurs points en même temps car un point de contrôle en particulier peut influencer plusieurs points de l'objet de façon simultanée.

La simple amélioration consiste à permettre à l'usager de modifier directement l'objet plutôt que les points de contrôle de la matrice ce qui donne une facilité d'utilisation surprenante des déformations libres et permet ainsi de faire des modèles très complexes de façon relativement simple comme montré dans *Figure.50*.

Figure.50. *Un modèle réalisé entièrement avec la méthode des déformations libres directes.*

4.4. Déformations libres étendues

Les déformations libres étendues ont été introduites par Coquillart [Coq90], dans le but d'avoir une matrice de points de contrôle de forme arbitraire. En effet, pour Coquillart, quatre problèmes ont à être pris en compte lors de l'élaboration d'une méthode de modélisation de solides :

1. La position de la région déformée sur la surface.
2. La grandeur de la région déformée.
3. La forme de la coquille externe de la région déformée.
4. La forme de la région déformée (à l'intérieur de la coquille de déformation).

Les déformations libres simples résolvent les deux premiers problèmes mais pas les deux derniers en raison de la forme restrictive de la matrice des points de contrôle (parallélépipède). Ainsi, les déformations libres étendues proposées par Coquillart vont essayer de régler les deux derniers problèmes. Bien évidement, le point principal des déformations libres étendues est l'introduction de matrices

de formes autres que celle d'un parallélépipède qui seront classées dans deux catégories : les matrices prismatiques et les matrices non-prismatiques. Les matrices prismatiques sont celles qui seront construites en modifiant des matrices de la forme d'un parallélépipède en bougeant ou en fusionnant des points de contrôle de cette dernière.

Les matrices cylindriques sont construites de cette façon. De plus, des matrices prismatiques composites peuvent être obtenues en fusionnant deux matrices prismatiques. Les matrices non-prismatiques quant à elles, peuvent être créées par l'extrusion ou la révolution d'une forme 2D quelconque.

Le processus de déformation dans le cas des déformations libres étendues, peut se séparer en deux étapes essentielles:

1. L'usager sculpte la matrice des points de contrôle qu'il veut et la gèle ensuite sur l'objet à déformer.
2. L'usager modifie les points de contrôle de la matrice et l'objet sera déformé en conséquence.

La première étape est simple à comprendre mais les opérations mathématiques impliquées sont non-triviales. En effet, lorsque l'usager gèle la matrice sur l'objet déformé, une paramétrisation locale doit être calculée pour chaque sous-région de la matrice (dans ce modèle le polynôme de Bernstein trivarié de degré 3 est utilisé et on se retrouve donc avec des sous-volumes de Bézier).

A cause de la forme non parallélépipédique de chaque sous-région de la matrice un algorithme numérique (Newton dans ce cas particulier) doit être utilisé pour approximer la paramétrisation locale.

Dans un deuxième temps, les points de contrôle sont modifiés et à partir de la paramétrisation de chaque sous région, les nouvelles positions de l'objet déformé seront calculées.

Notons cependant que l'usager ne peut bouger que les points de contrôle qui forment les coins des sous-régions pour des raisons de continuité de surface. En effet, étant donné que chaque sous-région de l'espace est représentée par un volume de Bézier, des problèmes majeurs de continuité se posent sur les frontières des sous-volumes. Afin de pallier à ce problème, seuls les coins des sous-régions peuvent être modifiés par l'usager et le système calcule lui même la position de tous les autres points de contrôle afin d'assurer une bonne continuité de surface.

4.5. Déformations libres avec une matrice de topologie arbitraire

Une autre façon pour réussir à avoir une matrice de topologie arbitraire est la façon proposée par MacCraken et Joy [MJ96]. Dans leur article, MacCraken et Joy proposent de subdiviser récursivement un volume quelconque en de petites cellules facilement paramétrisables à l'aide d'une approximation trilinéaire en utilisant une méthode numérique. Ensuite, chaque point de l'objet se voit associer une cellule de départ et va se déformer avec elle. La *Figure.51* donne un exemple d'une déformation avec une matrice de topologie arbitraire.

La subdivision se fera grâce à une extension aux volumes de la subdivision de surface de Catmull-Clark [CC78]. De façon similaire à la subdivision de surfaces, de nouveaux sommets sont créés pour chaque primitive géométrique. Un point de cellule est créé au centroïde de chaque cellule, un point de face pour chaque face, un point d'arête pour chaque arête et un point de sommet pour chaque sommet. Ces points seront ensuite reliés entre eux pour former les nouvelles cellules du volume.

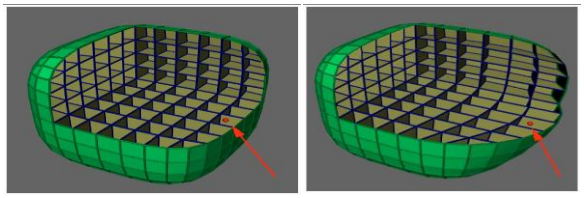


Figure.51. *Le volume subdivisé avant et après la déformation.*

La *Figure .52* donne un exemple d'une étape de subdivision appliquée sur trois matrices de formes différentes.

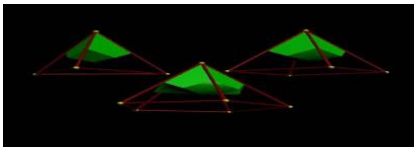


Figure .52. *Une étape de subdivision appliquée sur 3 matrices différentes.*

On note que, lors de l'application de la méthode de subdivision, l'espace déformable tend à s'éloigner passablement de la matrice originale. Afin de pallier a ce problème, des conditions de contrôle aux bordures devront être ajoutées au modèle. Ces conditions se résument comme suit :

1. Les sommets qui forment les coins de la matrice originale ne bougent pas.
2. Les bordures de la matrice originale sont identifiées et les points d'arête et de sommet associés à ces bordures sont calculé à partir de la subdivision des courbes B-splines plutôt qu'à partir de la subdivision de volume Catmull-Clark.
3. Tous les autres points sont calculés avec la subdivision de volume Catmull-Clark.

Voici un exemple montré dans la *Figure.53* de ce que donne l'application du contrôle des bordures.

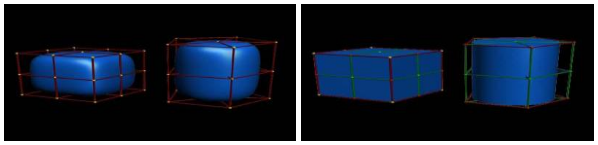


Figure.53. *L'application du contrôle des bordures.*

4.6. Autres types de déformations libres

De nombreux autres types de déformations libres ont été proposés dans la littérature, nous en donnons ici un bref aperçu.

4.6.1. Déformations libres rationnelles

Les déformations libres rationnelles ont été développées par Kalra et al. [KMMTT92] afin de donner un degré de liberté additionnel aux déformations libres simples. Dans le modèle des déformations libres rationnelles, chaque point de contrôle se voit assigner un poids qui définit l'attraction du point sur la surface (exactement comme dans le cas des B-splines rationnelles). Tout le reste se fait de la même façon que dans le cas des déformations libres simples à la différence qu'au lieu d'utiliser le polynôme de Bernstein trivarié, les B-splines rationnelles sont utilisées. Il s'en suit que le contrôle de la déformation n'est aucunement plus simple que dans le cas des déformations simples, il est juste un peu plus flexible.

4.6.2. Déformations libres basées sur les NURBS

Proposées par Lamousin et Waggenspack [LW94], les déformations libres basées sur les NURBS ont été créées pour étudier et améliorer différents aspects des déformations libres comme : les matrices de points de contrôle non-uniformes, la continuité entre les matrices de contrôle ainsi que le contrôle de la déformation. Utilisant les fonctions de mélange des B-splines rationnelles, qui permettent l'ajout de poids a chacun des points de contrôle et des points de contrôle non-uniformément répartis dans la matrice, Lamousin et Waggenspack ont réussi à introduire un modèle de déformation libre basé sur les NURBS.

4.6.3. Déformations libres animées

Coquillart [CJ91] propose ici un modèle de déformation libre qui va être utilisé pour animer un objet dans le temps. De la même façon que dans les déformations libres simples, un objet se voit assigner une matrice de points de contrôle initiale et déformée mais on ajoute ici un indice temporel à chacune des deux matrices. L'objet se déformera donc entre les 2 deux matrices selon l'intervalle de temps donné.

4.6.4. Déformations libres de plusieurs niveaux

Lee et al. [LCSW95] proposent des déformations libres de plusieurs niveaux afin d'introduire des contraintes de position au modèle simple dans le but de

faire du morphing. La déformation se fait ici en 2D mais pourrait être facilement étendue en 3D. L'idée de base est la même que dans le cas de déformations libres directes, mais ici des contraintes sont introduites quant à la position voulue des points déformés et le système doit minimiser l'erreur entre la déformation voulue et celle calculée.

4.7. Déformations sur les objets à base de points

4.7.1. Méthode de Pauly et al.

Pauly et al. [PKKG03] ont introduit un outil de déformation libre basé point qui permet à l'usager de déformer interactivement la surface par la spécification d'un champ de déformation lisse.

L'utilisateur d'abord définit une région déformable $X_d \subset S$ sur la surface du modèle, et il marque des parties de cette région comme une poignée de contrôle. La surface peut être modifiée après par une poussée, une traction ou une torsion sur cette poignée. Ces opérations interactives de la part de l'utilisateur seront traduites à un champ de tenseur continu, où pour chaque point dans la région déformable, on définit un mouvement de translation et de rotation sur lequel la surface va se déformer.

Le champ tenseur est basé sur un paramètre d'échelle continuellement variable $\in [0, 1]$, qui mesure la distance relative d'un point depuis la poignée. Plus un point est proche à la poignée, plus la déformation pour ce point est forte. Précisément, soit la poignée $X_1 \subset X_d$, autrement dit la région-1, et $X_0 = S - X_d$ la région-0, qui représente tous les points qui ne font pas partie de la région déformable. Pour ces deux régions, on définit des mesures de distance d_0 et d_1 respectivement pour j=0 et j=1, comme suit:

$$d_j(p) = \begin{cases} 0 & p \in X_j \\ min_{q \in X_j}(||q - p||) & p \notin X_j \end{cases}$$

Pour ces mesures de distance on calcul le paramètre d'échelle t comme suit:

$$t = \beta\left(\frac{d_0(p)}{d_0(p) + d_1(p)}\right)$$

où β: [0,1] →[0,1] est une fonction de mélange continue avec $\beta(0) = 0$ et $\beta(1) = 1$. Donc t=0 pour tous les point où $p \in X_0$, et t=1 pour tous les point où $p \in X_1$. Utilisant le paramètre d'échelle, on peut déterminer la position d'un point $p \in X_d$ après la déformation par la formule : p'= F(p, t). Où F est une fonction de déformation composée de deux parties, translation et rotation. On peut écrire :

- $F(p, t) = F_T(p, t)+F_R(p, t)$, tel que : $F_T(p, t) = p + t.v$, avec v est un vecteur de translation,

- $F_R(p, t) = R(a, t.\alpha).p$, avec $R(a, \alpha)$ est la matrix qui spécifie l'axe de rotation 'a' avec un angle α.

La *Figure .54* montre la déformation par translation d'un plan où le vecteur de translation v est égal au vecteur normale de ce plan. Selon une fonction de mélange β.

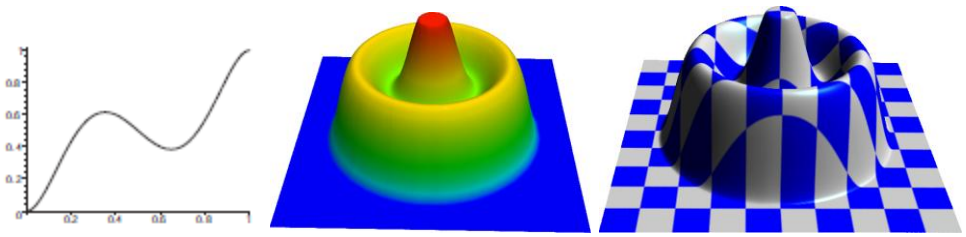


***Figure .54. A gauche** : une fonction de mélange, **au milieu :** carte de couleur du paramètre d'échelle (le bleu pour la région-0 et le rouge pour la région-1), **à droite** : le plan déformé.*

La *Figure.55* présente deux déformations par rotation d'un cylindre.

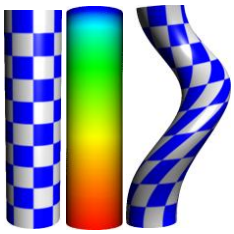


***Figure.55. A gauche** le cylindre original, **au milieu :** carte de couleur du paramètre d'échelle, **à droite** : le cylindre déformé par une rotation sur l'axe Y.*

Tandis que la combinaison des deux déformations (par translation et rotation) est illustrée dans *Figure .56*.

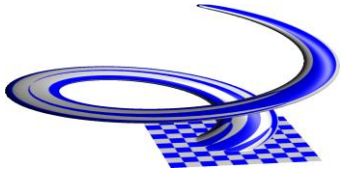


***Figure .56**.Une déformation par une combinaison de deux mouvements : translation et rotation.*

Pour effectuer une déformation de forme libre, l'utilisateur doit seulement sélectionnez la région-0 et la région-1 et choisir une fonction de mélange appropriée. Il peut alors interactivement déformer la surface en déplaçant la poignée avec une souris, ce qui donne une grande flexibilité.

4.7.2. Méthode de Boubekeur et al.

Du fait que l'interactivité est le principal obstacle pour l'utilisation des outils de déformation, en plus qu'elle ne peut pas être maintenue lorsque la complexité du modèle 3D est grande ou quand la déformation appliquée dépasse un certain seuil lié à la puissance de la machine. Boubekeur et al. [BSS07] proposent à résoudre ce problème d'évolutivité en introduisant un système de streaming basé sur une approche d'échantillonnage/reconstruction. D'abord, un algorithme

rapide de simplification adaptative out-of-core est effectué dans une étape de prétraitement, pour la construction d'une version simplifiée du modèle. Le modèle obtenu peut ensuite être soumis à des outils FFD, comme sa taille est réduite ce qui permet d'avoir une réponse interactive. Deuxièmement, une étape de post-traitement effectue une reconstruction de déformation préservative des caractéristiques en appliquant la déformation au modèle original subi en sa version simplifiée. Les deux étapes partagent une base de streaming basé point, ce qui les rend entièrement scalable et compatible à la fois aux nuages de points et aux maillages non-manifolds. Ce système offre également une couche générique out-of-core multi-échelle pour les outils FFD, puisque les deux étapes restent disponibles pour un sur-échantillonnage partiel au cours de la session interactive. Arbitrairement les grands modèles 3D peuvent donc être déformés de manière interactive avec la plupart des outils FFD, permettant l'utilisation des métaphores de déformation avancées pour des modèles allant de millions de milliards de points. Ce système permet aussi de travailler sur des modèles qui dépassent les capacités d'un outil FFD donné.

En tant que prétraitement, un sous-échantillonnage adaptatif efficace out-of-core est effectué sur l'objet d'origine de grande taille P_L durant le processus de streaming, pour obtenir une version simplifiée P_S qui permettra de préserver l'interactivité. Comme l'entrée peut être arbitrairement grande, l'algorithme utilise une finalisation spatiale pour maintenir une faible empreinte mémoire lors de l'échantillonnage. Cela conduit à la construction d'une grille grossière, où les cellules sont valables (tous les éléments sont présents dans la mémoire) que pour un temps partiel au cours de cette diffusion. Cela permet d'utiliser un arbre volume-surface pour l'échantillonnage adaptatif efficace de la géométrie fermée. Le partitionnement généré par l'arbre de volume-surface crée un point de prélèvement de la région, optionnellement équipé de triangles réindexés si il est fourni dans le flux d'entrée. Le modèle simplifié P_S, obtenu avec une empreinte mémoire, est ensuite soumis à un outil interactif FFD arbitraire, considéré comme une boîte noire qui fournit une déformation P_S* de P_S.

L'objectif du post-traitement de streaming out-of-core est de transférer efficacement et avec précision la déformation définie sur P_S pendant l'interaction FFD du gigantesque objet original P_L, pour obtenir l'objet final déformé P_L*. L'algorithme est purement local (chaque échantillon est traité indépendamment) ce qui le rend compatible avec le streaming géométrique, qui est crucial dans le contexte des objets de grande taille. Une extraction de chaque point p de P_L, un déplacement local selon P_S: p = p '+d.n, où p' est la projection de p sur un plan moyen H, avec une normale n définie par le filtrage local de P_S avec un noyau hermitien, et d est la distance de p à H. Notons que l'utilisation d'un déplacement scalaire dans la direction normale, le rend invariant aux rotations. En fait, P_S* a capturé déjà la déformation de la partie de basse fréquence du P_L, de sorte qu'on déforme cette partie de basse fréquence dans l'esprit de la déformation

Laplacienne à l'aide des coordonnées intrinsèques de p'. Au contraire des approches antérieures, ici le codage est basé sur le troisième voisinage. D'abord on sélectionne 3 échantillons de P_S dans le voisinage de p en utilisant un kd-tree, et en formant le plus petit triangle équilatéral possible T (p) = {qi, qj, qk} pour la symétrie et l'exactitude. Cette sélection permet l'utilisation des coordonnées barycentriques projetées B(p) = {bi, bj, bk} pour coder p' en fonction de ces échantillons, en évitant l'utilisation des moindres carrés. Enfin, la déformation est transférée par: p' *= T' * (p) B . '(p) ^{T}.

On extrait également un facteur d'échelle pour le calcul de d* en fonction de la modification de la surface locale sur P_S, comme suit: $\forall p \in P_L$ p*= D (p) = T' * (p). B '(p) ^{T}+d *. n*.

Cette déformation à base de points transférée du P_S * à P_L est rapide et lisse grâce au filtrage hermitien du plan de projection, qui a offert un bon compromis pour traiter rapidement le très grand nombre d'échantillons (jusqu'à un milliard) diffusés depuis P_L (*Figure .57*).

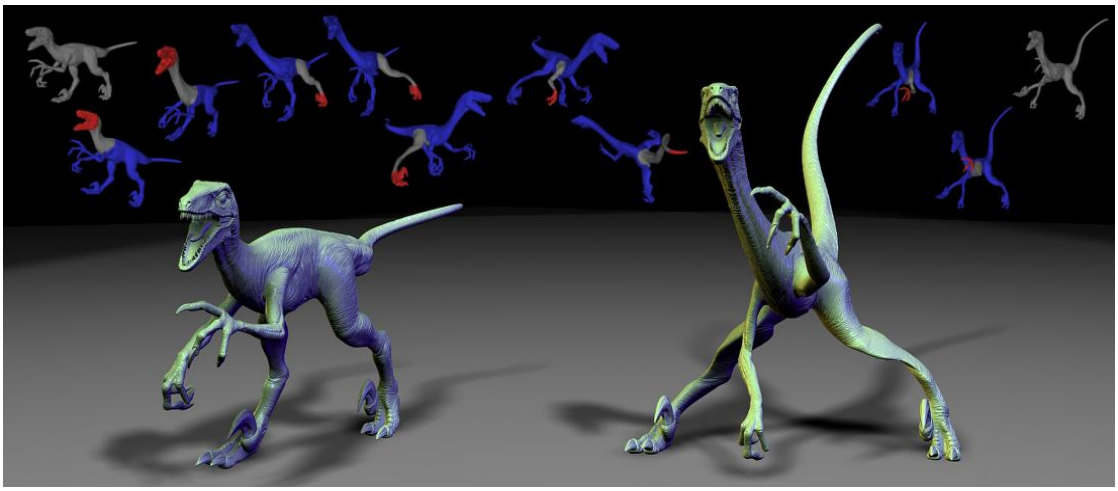


Figure .57. *Résultat de l'application des déformations libres sur le modèle du Raptor (8 M triangles).*

4.8. Mise en œuvre d'un déformateur de forme libre interactif sur des objets à base de splats grâce à l'accélération GPU

4.8.1. Déformateur

Un déformateur est une opération qui prend en entrée un ensemble de sommets et qui génère de nouvelles coordonnées pour les sommets (les coordonnées déformées). En outre, de nouvelles normales et tangentes peuvent également être générées. Cette tâche est parfaitement adaptée pour le vertex shader (voir la section 4.8.4), où des conditions doivent être remplies (pas de création ou de suppression des sommets ou des arêtes, sans influence d'un sommet donné déformé sur un autre, et le déterminisme de la déformation). La

plupart des déformateurs ont des paramètres de contrôle: les paramètres définis par l'utilisateur donnent au déformateur une certaine variation.

Dans le sens le plus général, nous pensons d'un déformateur comme une fonction vectorielle: $f(x, y, z)$.

Ou plus précisément, c'est l'ensemble de trois fonctions scalaires :

$$(f_x(x, y, z), f_y(x, y, z), f_z(x, y, z))$$

où f_x, f_y *et* f_x sont les fonctions composantes de *f*. Cette fonction *f* prend les coordonnées *(x, y, z)* d'un sommet et renvoie les coordonnées déformées de ce sommet. Pour déformer un maillage, on prend chaque sommet, on entre ses coordonnées dans *f*, et on stocke le résultat comme nouvelles coordonnées pour ce sommet. La fonction de déformateur *f* pourrait être définie pour tous les sommets, ici on parle d'une **déformation globale**, ou pour certains sous-ensembles, dans ce cas on parle d'une **déformation locale**.

Les transformations linéaires (translation, rotation et mise à l'échelle) ne sont pas des déformateurs particulièrement intéressant puisque nous savons tout simplement la façon de calculer les normales déformée des sommets (en les multipliant par la transposition inverse de la matrice de transformation). Mais en général, un déformateur donné ne pourra être écrit sous une forme matricielle s'il s'agit pas d'une transformation linéaire, et donc de nouvelles techniques permettant de transformer les normales doivent être trouvées.

4.8.2. Déformation des normales

La première solution triviale pour calculer les normales déformées est de procéder par une deuxième passe, après la passe de déformation des coordonnées des sommets. Dans cette deuxième passe, on recalcule les normales en se basant sur les sommets voisins dans le nouveau maillage déformé. Malheureusement nous n'avons pas cette option avec le vertex shader. Chaque sommet en cours de traitement dans le vertex shader n'a pas connaissance de ses voisins, et il n'y aura pas un autre étage où on peut chercher autour pour faire les calculs. La deuxième solution est une technique numérique approchée où la déformation des normales peut être obtenue en déformant trois points pour chaque sommet d'entrée. On trouve deux nouveaux points très proche du sommet d'entrée ; le premier dans la même direction de la tangente à ce sommet, et le deuxième dans la même direction de la binormale à ce sommet. Ces trois points définissent deux vecteurs dont le produit cartésien est la normale au somme d'entrée. On doit donc déformer les trois points par le déformateur, et ces trois points déformés vont définir deux nouveaux vecteurs dont le produit cartésien sera une approximation de la normale déformée. Plus Le rapprochement entre ces trois points est grand, plus l'approximation sera précise, toutefois des problèmes de précision numérique apparaîtra si les points sont trop rapprochés.

La troisième solution est basée sur la matrice jacobiennne. Si un déformateur $f(x, y, z) = (f_x, f_y, f_z)$ dont les dérivées partielles du premier ordre sont continues, alors on peut calculer pour chaque sommet une matrice qui s'appelle la matrice jacobienne **J**. comme on le sait, si une fonction *f (x, y, z)* est lisse, donc *f* peut être approchée par une transformation linéaire J, plus une translation *f (x) ≈ J (a) (xa) + f (a)*, pour un point a, et *x = (x, y, z).* Ceci selon le théorème de Taylor généralisée. La matrice jacobienne est définie comme suit lorsque toutes les dérivées partielles sont évaluées à *(x, y, z)*. (notons que la matrice jacobienne est souvent notée ∂ f (x)):

$$J(x, y, z) = \begin{bmatrix} \partial \mathrm{fx}/\partial \mathrm{x} & \partial \mathrm{fx}/\partial \mathrm{y} & \partial \mathrm{fx}/\partial \mathrm{z} \\ \partial \mathrm{fy}/\partial \mathrm{x} & \partial \mathrm{fy}/\partial \mathrm{y} & \partial \mathrm{fy}/\partial \mathrm{z} \\ \partial \mathrm{fz}/\partial \mathrm{x} & \partial \mathrm{fz}/\partial \mathrm{y} & \partial \mathrm{fz}/\partial \mathrm{z} \end{bmatrix}$$

Nous pouvons transformer les normales en utilisant la matrice jacobienne de deux façons équivalentes:

1. La transposée inverse de **J** (x, y, z) peut être calculée par sommet dans le vertex shader et elle peut être utilisée pour transformer la normale directement. Si **N** est la normale d'entrée, alors la normale déformée est **N'**:

$$\mathbf{N'} = (\mathrm{J}(\mathrm{x,y,z})^{-1})^{\mathrm{T}} * \mathbf{N}$$

2. Si le vecteur tangent U est donné par sommet, alors on peut générer le vecteur binormale **V**= **N^U**. On peut transformer la tangente et la binormale vers l'espace déformé, utilisant la matrice **J** (x, y, z). Alors, la nouvelle normale déformée sera:

$$\mathbf{N'} = (\mathbf{J}(\mathrm{x,y,z})*\mathbf{U}) \wedge (\mathbf{J}(\mathrm{x,y,z})*\mathbf{V})$$

Inverser les matrices 3x3 n'est pas une tâche particulièrement adaptée au vertex shader, par conséquent, la deuxième méthode est généralement plus facile à mettre en œuvre et plus rapide à calculer.

4.8.3. Déformation de forme libre

Rappelons le principe de la déformation de forme libre simple qui consiste à envelopper l'objet dans l'espace, puis déformer ce dernier. Ensuite les déformations faites sur l'espace seront appliquées à l'objet enveloppé.

Une fois le parallélépipède enveloppant l'objet est calculé, où son origine est X_0 et les trois côtés sont S, T et U, on calcule les coordonnées locales *(s, t, u)* des sommets de l'objet, selon la formule suivante:

$$X = X_0 + sS + tT + uU$$

Ensuite, on pourrait fixer la grille des points de contrôle qui contient *(l +1)* points sur l'axe S, *(m +1)* points sur l'axe T et *(n +1)* points sur l'axe U, *(l +1) * (m*

*+1) * (n +1)* points au total. Les coordonnées de chaque point de contrôle sont calculées comme suit:

$$P_{ijk} = X_0 + \frac{j}{l}S + \frac{i}{m}T + \frac{k}{n}U$$

Maintenant, la déformation peut être introduite en modifiant les coordonnées des points de contrôle de la grille, alors les sommets seront déformés en fonction de la déformation faite sur la grille, et ce grâce aux polynômes de Bernstein comme illustré dans le déformateur suivant:

$$X' = f(X) = f(x,y,z) = (f_x, f_y, f_z) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}\, B_i^l(s) B_j^m(t) B_k^n(u)$$

$f_x = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^x\, B_i^l(s) B_j^m(t) B_k^n(u)$

$f_y = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^y\, B_i^l(s) B_j^m(t) B_k^n(u)$

$f_z = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^z\, B_i^l(s) B_j^m(t) B_k^n(u)$

Où $B_i^l(s), B_j^m(t)\ et B_k^n(u)$ sont les polynômes de Bernstein définis comme suit :

$B_i^l(s) = \binom{l}{i}(1-s)^{l-i}s^i \quad , \quad B_j^m(t) = \binom{m}{j}(1-t)^{m-j}t^j \quad , \quad B_k^n(u) = \binom{n}{k}(1-u)^{n-k}u^k$

Jusqu'à présent, on n'a développé que les calculs des coordonnées déformées des sommets. Et afin de calculer les normales déformées associées à chaque sommet, on devrait calculer la matrice Jacobienne à chaque sommet, et ceci sera effectué par le calcul des dérivées partielles du premier ordre du déformateur *f(x,y,z)*, de la manière suivante:

$\frac{\partial f_x}{\partial x} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^x\, BP_i^l(s) B_j^m(t) B_k^n(u)$

$\frac{\partial f_x}{\partial y} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^x\, B_i^l(s) BP_j^m(t) B_k^n(u)$

$\frac{\partial f_x}{\partial z} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^x\, B_i^l(s) B_j^m(t) BP_k^n(u)$

$\frac{\partial f_y}{\partial x} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^y\, BP_i^l(s) B_j^m(t) B_k^n(u)$

$\frac{\partial f_y}{\partial y} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^y\, B_i^l(s) BP_j^m(t) B_k^n(u)$

$\frac{\partial f_y}{\partial z} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^y\, B_i^l(s) B_j^m(t) BP_k^n(u)$

$\frac{\partial f_z}{\partial x} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^z\, BP_i^l(s) B_j^m(t) B_k^n(u)$

$\frac{\partial f_z}{\partial y} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^z\, B_i^l(s) BP_j^m(t) B_k^n(u)$

$\frac{\partial f_z}{\partial z} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{ijk}^z\, B_i^l(s) B_j^m(t) BP_k^n(u)$

Où $BP_i^l(s), BP_j^m(t)\ et BP_k^n(u)$ sont les dérivées premières des polynômes de Bernstein calculées comme suit :

$$\frac{\partial B_i^l(s)}{\partial x} = BP_i^l(s) = \binom{l}{i}((l-i)(-\frac{1}{S})(1-s)^{l-i-1}s^i + i\left(\frac{1}{S}\right)(1-s)^{l-i}s^{i-1})$$

$$\frac{\partial B_j^m(t)}{\partial y} = BP_j^m(t) = \binom{m}{j}((m-j)(-\frac{1}{T})(1-t)^{m-j-1}t^j + j(\frac{1}{T})(1-t)^{m-j}t^{j-1})$$

$$\frac{\partial B_k^n(u)}{\partial z} = BP_k^n(u) = \binom{n}{k}((n-k)(-\frac{1}{U})(1-u)^{n-k-1}u^k + k(\frac{1}{U})(1-u)^{n-k}u^{k-1})$$

4.8.4. Pipeline graphique de rendu

En rendu 3D, le pipeline graphique ou le pipeline de rendu réfère aux étages nécessaires pour transformer des primitives 3D de l'espace objet dans une image 2D sur l'espace écran. Les étages se sont chargés du traitement de l'information d'entrée en tant que des propriétés des primitives (coordonnées, couleur, matériau, etc.) utilisées pour décrire ce qui doit être rendu. Les primitives typiques des graphismes 3D sont des sommets, des lignes et des facettes.

Les étages du pipeline de rendu d'OpenGL (comme une API standard ouvert multi-langages et multiplateformes pour le rendu 3D) sont illustrés dans *Figure.58*.

Initialement, l'ancien pipeline de rendu OpenGL a été de fonctionnalités fixes (ce qui signifie que le programmeur n'a pas un grand contrôle sur la manière par laquelle les données d'entrée seront traitées), mais avec l'introduction des GPU modernes (Graphics Processing Unit), plusieurs étages deviennent programmables à travers le modèle shader, en utilisant l'un des différents langages de programmation comme GLSL (OpenGL Shading Language), HLSL (High Level Shading Language) et Cg (C for graphics). Cela permet non seulement le contrôle de haut niveau des effets de rendu, mais le plus important est le déchargement des calculs étroitement liés aux sommets (vertex shader) et aux pixels (fragment shader) du CPU vers le GPU, afin d'avoir un rendu efficace en temps réel.

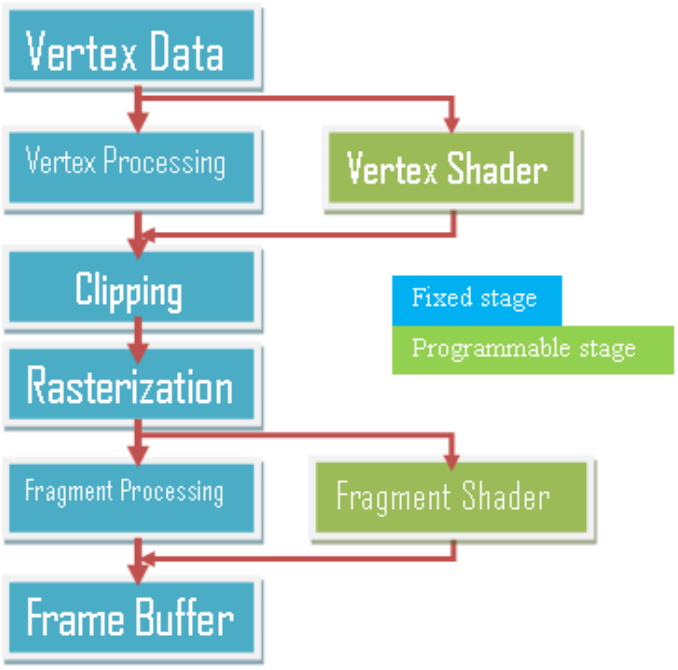


Figure.58. *Pipeline de rendu d'OpenGL simplifié avec l'introduction du modèle shader.*

4.8.5. Résultats expérimentaux

Nous avons appliqué le déformateur décrit en haut, sur plusieurs objets, nous montrons ici deux objets choisis, le premier est une sphère texturée à base de points qui contient 59650 splats, et le second est un visage humain à base de points qui contient 40880 splats. La grille du déformateur contient 10x10x10 cellules, donc 11x11x11 points de contrôle, c'est à dire 1331 points de contrôle au total.

Le déformateur a été mis en œuvre sur un ordinateur avec les caractéristiques suivantes: Mémoire vive: 1 Go, CPU: Intel Pentium Dual-Core 2,70 GHz, carte graphique: NVIDIA Geforce 9500 GT avec 1 Go d'espace mémoire.

Notez que la première implémentation est une implémentation basée CPU, c'est à dire sans l'exploitation des fonctionnalités programmables du GPU (seulement les fonctionnalités fixes).

Le déformateur a été appliquée avec différents paramètres de contrôle pour le but de générer des déformations globales et locales avec des mouvements de translation et de rotation. Pour la déformation globale, on applique le mouvement choisi sur tous les points de contrôle, de l'autre côté, pour la déformation locale, on applique le mouvement choisi sur un sous-ensemble des points de contrôle. Et les deux sortes du mouvement choisi sont:

1. L'étirement : est effectué par la translation des points de contrôles avec un vecteur 3D $V_d(x_d, y_d, z_d)$ selon la formule suivante:

$$\forall i \in [n_1 \;\; n_2], \qquad P'_i = P_i + V_d$$

2. La torsion : est effectuée par la rotation des points de contrôle autour d'un axe (A), avec un angle (α) selon la formule suivante:

$$\forall i \in [n_1 \ \ n_2], \qquad P'_i = Rot(P_i, A, \alpha)$$

Les figures suivantes (*Figure.59*, *Figure.60*, *Figure.61*, *Figure.62*, *Figure.63*) illustrent les résultats obtenus avec différentes configurations (déformation globale / locale, mouvement de translation / rotation).

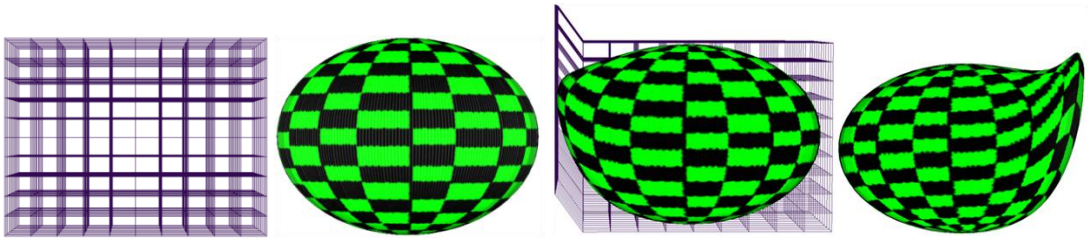


***Figure.59. A gauche**: l'état normal d'une sphère texturée à base de splats, **à droite** : une Déformation locale par un mouvement translationnel (étirement) sur la sphère.*

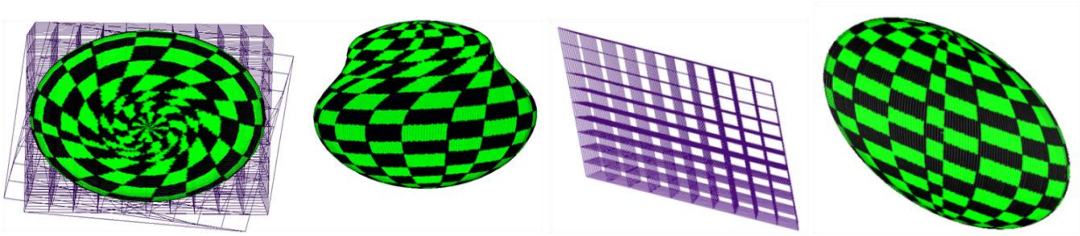


***Figure.60. A gauche**: une déformation locale par un mouvement rotationnel (torsion), **à droite** : une déformation globale par un mouvement translationnel (étirement) sur la sphère.*

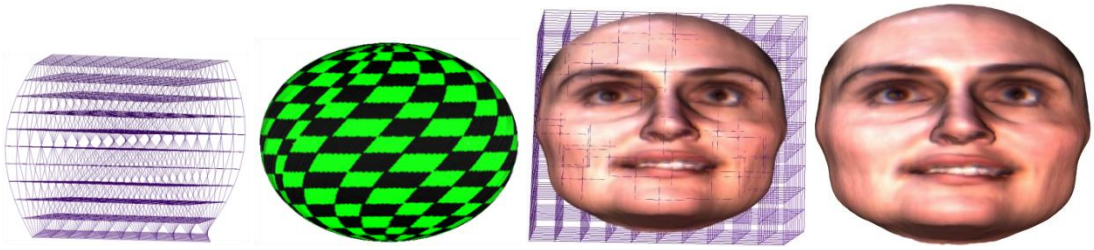


***Figure.61. A gauche** : une déformation globale par un mouvement rotationnel (torsion) sur la sphère, **à droite** : l'état normal d'un visage humain à base de splats.*

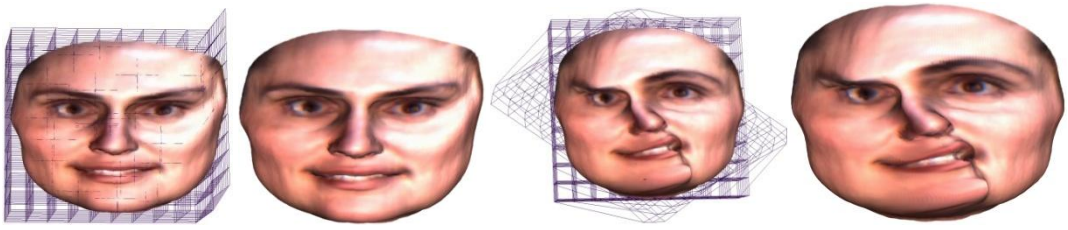


***Figure.62. A gauche** : une déformation locale par un mouvement translationnel (étirement), **à droite** : une déformation locale par un mouvement rotationnel (torsion) sur le visage.*

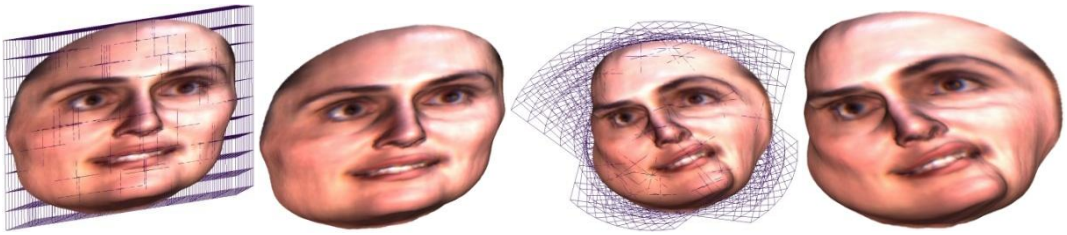


***Figure.63. A gauche** : une déformation globale par un mouvement translationnel (étirement), **à droite** : une déformation globale par un mouvement rotationnel (torsion).*

Afin d'élaborer notre déformateur, nous l'a soumis à un jeu de tests pour le mettre à l'échelle, ceci en variant la taille des objets à base de points de 10.000 splats à 150.000 splats, mais aussi la résolution de la grille des points de contrôle (27, 125 et 343 points) tout en mesurant le nombre des images produites par seconde : FPS (Frame Per Second). Notons qu'à cause de la limite maximale de la taille des variables 'uniform' qu'on peut transférer depuis le CPU vers le GPU, on s'est limité supérieurement à une grille de 343 points de contrôle.

Le déformateur peut être qualifié de l'une des mentions suivantes selon le FPS mesuré, comme suit :

- FPS >=25 : le déformateur est temps réel.
- FPS >=16 : le déformateur est fluide.
- FPS > 1 : le déformateur est interactif.
- FPS <=1 : le déformateur est non interactif.

Les résultats obtenus lors de l'application du déformateur dans sa version CPU avec les différentes configurations, sont comme montrent les figures suivantes.

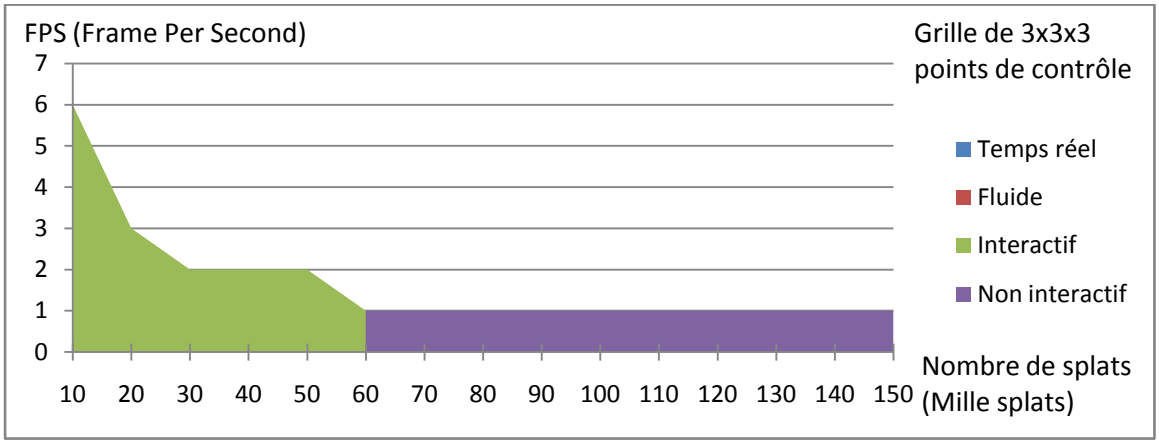


***Figure. 64.** Graphique du FPS mesuré lors de l'application du déformateur dans sa version CPU, dont la grille des points de contrôle comporte 27 points.*

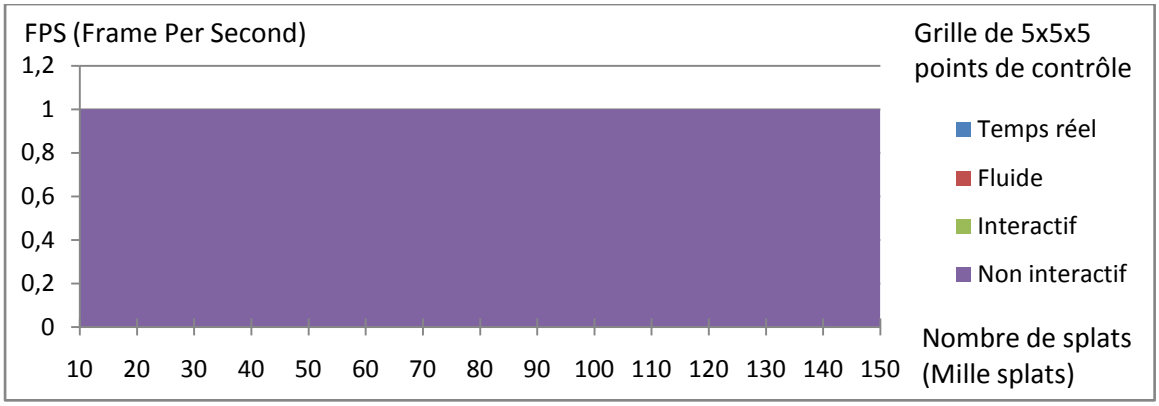


***Figure. 65**. Graphique du FPS mesuré lors de l'application du déformateur dans sa version CPU, dont la grille des points de contrôle comporte 125 points.*

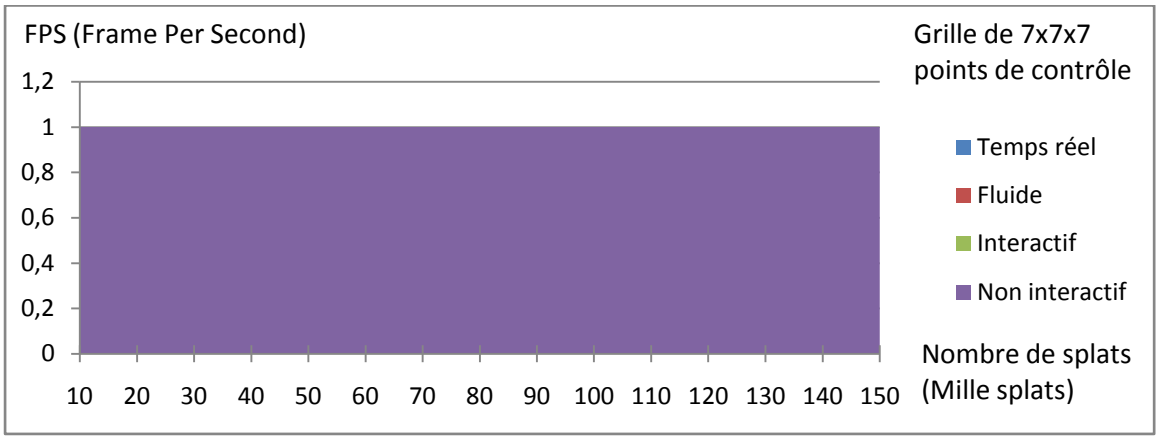


***Figure. 66.**Graphique du FPS mesuré lors de l'application du déformateur dans sa version CPU, dont la grille des points de contrôle comporte 343 points.*

Dans le graphique de la *Figure. 64*, on observe que le déformateur dans sa version CPU et dont la grille des points de contrôle contient uniquement 27 points, est interactif uniquement quand on l'applique sur des objets 3D comportant moins de 60 mille splats. Et au-delà de ce seuil, le déformateur devient non interactif.

Cependant, dès qu'on augmente la résolution de la grille des points de contrôle (à 125 et 343 points), le déformateur apparaît non interactif du tout quelque soit la taille de l'objet qu'on y soumette (*Figure. 65* et *Figure. 66*).

Il est lucide que cette première implémentation basée CPU du déformateur, est non interactive en général et ceci est dû à la lourdeur des calculs effectués au niveau du CPU.

4.8.6. Contribution : écriture du vertex shader

Notre contribution consiste à décharger les calculs de déformation depuis le CPU vers le GPU des cartes graphiques modernes, afin d'obtenir un temps de réponse interactif et acceptable et qui est plus approprié pour les usagers et les concepteurs.

Une fois que nous avons écrit le déformateur sous sa forme fonctionnelle, sa réécriture pour le vertex shader est assez facile. Les paramètres de contrôle sont passés comme des variables 'uniform' vers le vertex shader depuis l'application. Dans notre cas, ces paramètres sont : le type de la déformation, le type du mouvement, le vecteur de déplacement et l'axe / angle de rotation. Pour calculer les nouvelles coordonnées d'un sommet nous appliquons simplement le précédent déformateur *f (x, y, z)* sur les coordonnées du sommet en cours (gl_Vertex) puis on stocke le résultat dans la position du sommet (gl_Position) selon la syntaxe GLSL. Ainsi, le déformateur sera mis en œuvre au niveau du vertex shader. Toutefois, au niveau pixel shader, il n'y aura aucun traitement spécial. Ci-dessous des pseudo-codes du vertex shader et du pixel shader.

Vertex shader

```
uniform bool global_local;
uniform bool transl_rotation;
uniform vec3 disp_vec;
uniform float angle_rotation;
uniform vec axis_rotation;
vec4 Vertex;
vec3 Normal;
void main(){
Vertex=gl_Vertex;
Normal=gl_Normal;
deformation(Vertex,Normal,global_local,transl_rotation, disp_vec, angle_rotation, axis_rotation);
gl_FrontColor=gl_Color;
gl_Position=gl_ModelViewProjectionMatrix *Vertex;
}
```

Fragment shader

```
uniform sampler2D color_texture;
void main() {
```

```
gl_FragColor = gl_Color*texture2D(color_texture, gl_TexCoord[0].st);
}
```

Les résultats obtenus par l'implémentation basée GPU de l'ancien déformateur (*Figure. 67* et *Figure. 68* et *Figure. 69*), montrent bien l'amélioration en terme du temps de réponse par rapport les anciens résultats où toutes les déformations ont été interactives dans les différentes configurations (taille des objets et la résolution de la grille des points de contrôle), voire fluide et des fois temps réel !

Dans le graphique de la *Figure. 67*, on remarque que le déformateur dont la grille des points de contrôle comporte 27 points, est tout à fait interactif, et il atteint un taux fluide quand la taille des objets est moins de 130 mille splats. On constate aussi que le FPS touche des valeurs temps réel dès que la taille des objets soit moins de 80 mille splats.

Dans le graphique de la *Figure. 68*, on voit que le déformateur dont la grille des points de contrôle comporte 125 points, est tout le temps interactif, et il atteint un taux fluide lorsque la taille des objets est moins de 30 mille splats. On note aussi que le FPS touche des valeurs temps réel dès que la taille des objets soit moins de 20 mille splats.

Dans le graphique de la *Figure. 69*, on perçoit que le déformateur dont la grille des points de contrôle comporte 343 points, est bien interactif, et il atteint un taux fluide quand la taille des objets est moins de 20 mille splats. Et d'après les expérimentations faites, cette implémentation basée GPU n'atteint un taux non interactif qu'avec les objets comportant un nombre de splats au-delà de 200 mille splats.

Finalement, Les résultats obtenus par le déformateur proposé avec l'accélération GPU ont donné lieu à un temps de réponse interactif convenable aux usagers au contraire de l'implémentation CPU qui n'était pas interactive.

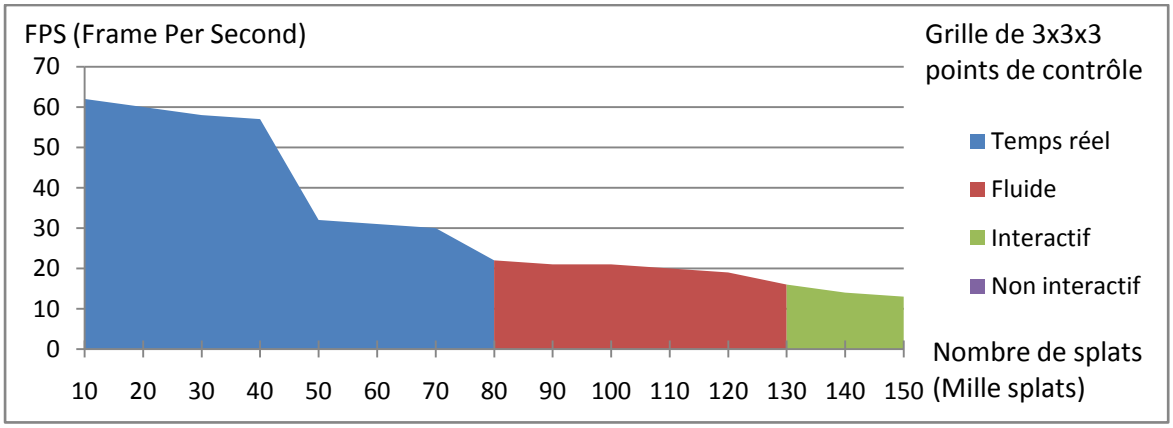


Figure. 67.*Graphique du FPS mesuré lors de l'application du déformateur dans sa version GPU, dont la grille des points de contrôle comporte 27 points.*

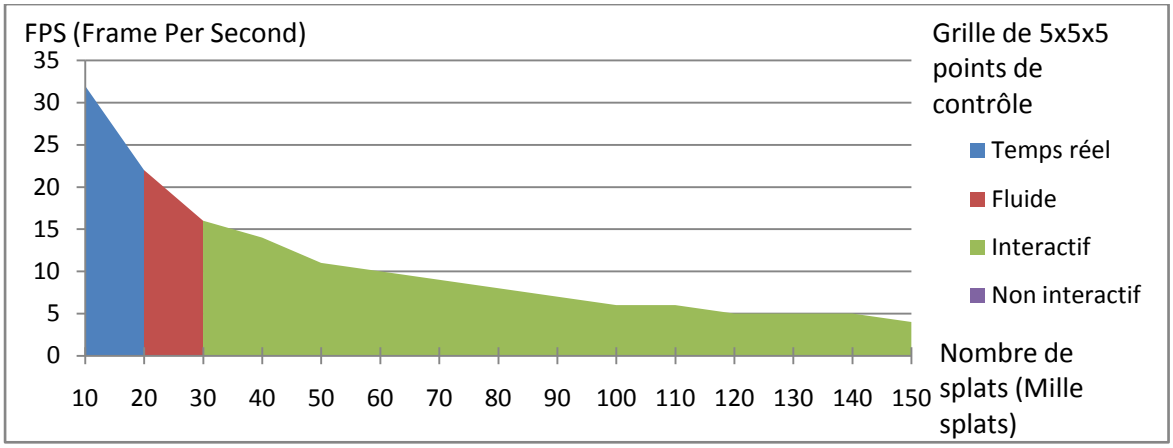


Figure. 68.*Graphique du FPS mesuré lors de l'application du déformateur dans sa version GPU, dont la grille des points de contrôle comporte 125 points.*

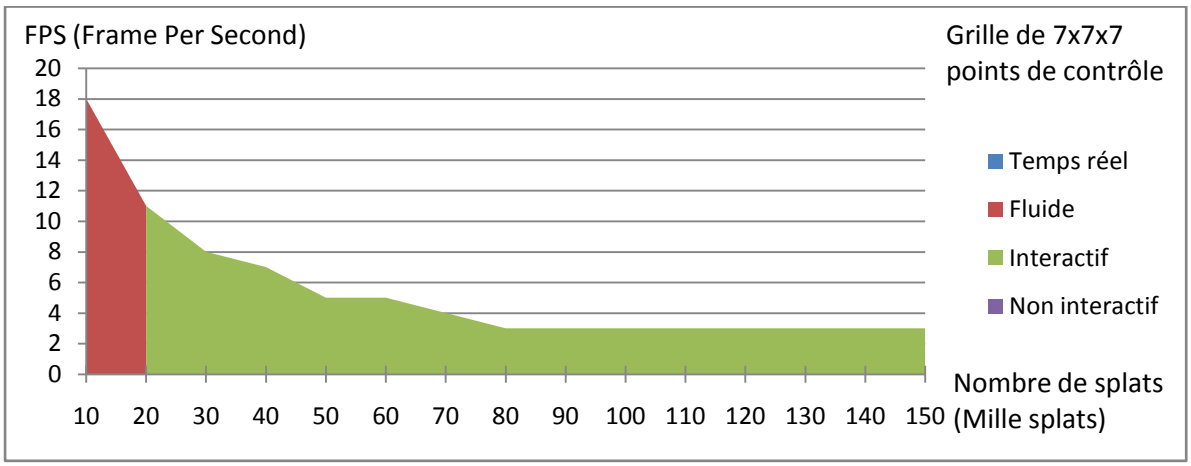


Figure. 69.*Graphique du FPS mesuré lors de l'application du déformateur dans sa version GPU, dont la grille des points de contrôle comporte 343 points*

4.9. Conclusion

Comme le montrent les résultats obtenus en haut, l'effet des progrès technologiques en matière de performance et de fonctionnalités fournies par les cartes graphiques modernes a permis de rendre la déformation des objets constitués des centaines de milliers de points très rapide et notamment interactive malgré la lourdeur des calculs d'un tel déformateur de forme libre (déformation de la grille des points de contrôle, la déformation des coordonnées des splats, et finalement la déformation des normales).

Conclusion générale

Dans ce mémoire, nous avons abordé la texturation et la déformation d'objets 3D à base de points. Le but était de trouver des méthodes efficaces pour résoudre les problèmes liés d'une part au placage de texture et d'autre part à l'application des déformations de forme libre aux objets modélisés par un nuage de points.

L'utilisation des points comme une primitive de modélisation et de rendu pour la synthèse d'image a clairement prouvé son efficacité à un point où on ne distingue pas à l'aide de notre système visuel humain que ce genre d'objets 3D est en réalité un ensemble discret de points sans aucune connectivité explicite entre eux.

Cependant lors de la manipulation de ce type d'objets, le manque d'information sur la connectivité de ces primitives (points) posent un problème où on se retrouve obligé de réappliquer cette manipulation (ou précisément déformation) sur l'ensemble des points qui est généralement assez grand et par la suite les calculs deviennent lourds, ce qui rend la manipulation non interactive. Tandis que dans le cas des objets à base de polygones, on n'applique la manipulation que sur les sommets (qui sont proportionnellement réduits) et cette manipulation se propage automatiquement (par interpolation) sur les surfaces (facettes) de cet objet.

Mais grâce aux récents progrès technologiques qu'a connus le matériel graphique, le CPU et le GPU des cartes graphique se trouvent capable à coopérer entre eux efficacement, et ce qui était un problème sérieux auparavant, est devenu une simple tâche à réaliser. Donc les déformations sur les nuages de points ont pu être assez interactives (comme nous avons vu dans le dernier chapitre), et ceci à renforcer encore cette nouvelle représentation alternative des objets 3D.

Bien que les résultats de la formalisation mathématique et du développement de ces outils de texturation et de déformation interactive sur les nuages de points aient atteint un bon degré d'acceptation, ce travail pourrait être poursuivi par d'autres améliorations :

- Développement d'une interface graphique conviviale permettant aux usagers d'utiliser les événements de la souris pour la manipulation des nuages de points.
- La réécriture de ces outils développés sous formes de plugins pour qu'ils soient directement intégrés dans le logiciel de traitement d'objets 3D à base de point : PointShop3D.

Bibliographie

[AA03b] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In SMI '03 : Proceedings of the Shape Modeling International 2003, page 272, Washington, DC,USA, 2003. IEEE Computer Society.

[ABCO+01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In VIS '01 : Proceedings of the conference on Visualization '01, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society.

[ABCO+03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surface. IEEE Transaction on Visualization and Computer Graphics, 9(1) :3–15, 2003.

[Bar84] A. H. Barr. Global and local deformations of solid primitives. In H. Christiansen, editor, SIGGRAPH '84 Conference Proceedings (Minneapolis, MN, July 23-27, 1984), pages 21–31. ACM, July 1984.

[BK03] Mario Botsch and Leif Kobbelt. High-Quality Point-Based Rendering on Modern GPUs. In 11th Pacific Conference on Computer Graphics and Applications, pages 335–343, 2003.

[BK04] Mario Botsch and Leif Kobbelt. Phong splatting. In Proceedings of Point-Based Graphics 2004, pages 25–32, 2004.

[Bli82] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In SIGGRAPH '82 : Proceedings of the 9th annual conference on Computer graphics and interactive techniques, pages 21–29, New York, NY, USA, 1982. ACM Press.

[BS86] E.A.Bier , K.R. Sloan , "Two-part texture mapping", IEEE COmputer Graphics and Applications, 6(9), 40-53, 1986.

[BSS07] T. Boubekeur, O. Sorkine and C. Schlick, "Scalable Freeform Deformation," In Proceedings of ACM Siggraph 2007 - Sketch Program, August 2007.

[BWK02] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In Proceedings of the 13th Eurographics workshop on Rendering, pages 53–64, 2002.

[Car84] Loren Carpenter. The a -buffer, an antialiased hidden surface method. In SIGGRAPH'84 : Proceedings of the 11th annual conference on Computer graphics and interactive techniques, pages 103–108, New York, NY, USA, 1984. ACM Press.

[CC78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design, 10 :350–355, September 1978.

[CHP+79] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, and M. Howard. Towards an interactive high visual complexity animation system. In SIGGRAPH '79 : Proceedings of the 6th annual conference on Computer graphics and interactive techniques, pages 289–299, New York, NY, USA, 1979. ACM Press. 132.

[CJ91] Sabine Coquillart and Pierre Jancene. Animated free-form deformation : An interactive animation technique. Computer Graphics, 25(4) :23–26, July 1991.

[Coq90] Sabine Coquillart. Extended free-form deformation : A sculpturing tool for 3D geometric modeling. volume 24, pages 187–196, August 1990.

[DVS03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In Proceedings of ACM SIGGRAPH 2003, pages 657–662, 2003.

[GD98] J. P. Grossman and William J. Dally. Point sample rendering. Rendering Techniques 98, pages 181–192, 1998.

[Gro98] J. P. Grossman. Point sample rendering. Master's thesis, Massachusetts Intitute of Technology, 1998.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuezle. Surface reconstruction from unorganized points. In Proceedings of ACM SIGGRAPH 92, 1992.

[Hec89] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California at Berkley, 1989.

[HHK92] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. Computer Graphics, 26(2) :177–184, July 1992.

[KMMTT92] Prem Kalra, Angelo Mangili, Nadia Magnenat-Thalmann, and Daniel Thalmann. Simulation of facial muscle actions based on rational free form deformations. volume 11, pages 59–69, September 1992.

[KV01] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In Proceedings of the 12th Eurographics Workshop on Rendering Techniques, pages 139–150, London, UK, 2001. Springer-Verlag.

[KV03] Aravind Kalaiah and Amitabh Varshney. Modeling and rendering of points with local geometry. IEEE Transactions on Visualization and Computer Graphics, 9(1) :30–42, 2003.

[LCSW95] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, and George Wolberg. Image metamorphosis using snakes and free-form deformations. In Robert Cook, editor, SIGGRAPH 95 Conference Proceedings, Annual Conference Series, pages 439– 448. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[Lev01] D. Levin. Mesh-independent surface interpolation. In Advances in Computational Mathematics, 2001.

[LKM01] E. Lindholm, M. Kilgard, H. Moreton, 2001. A user-programmable vertex engine. In: Proc. of ACM SIGGRAPH 01. pp. 149–158.

[LR89] Wang-He Lou et Anthony P. Reeves. The shape information distribution on a three-dimensional object. 12ème Colloque GRETSI-JUAN-LES-PINS,12 au 16 juin 1989.

[LR98] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In Rendering Techniques, pages 301–314, 1998.

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. Math. Comp., 37 :141–159, 1981.

[LW85] M. Levoy and T. Whitted. The use of points as display primitives. Technical Report Technical Report TR 85-022, The University of North Carolina at Chapel Hill,Department of Computer Science, 1985.

[LW94] Henry J. Lamousin and Warren N. Waggenspack, Jr. NURBS-based free-form deformations. IEEE Computer Graphics and Applications, 14(6) :59–65, November 1994.

[MGAK03] W. R. Mark, R. S. Glanville, K. Akeley, M. J. Kilgard, 2003. Cg: A system for programming graphics hardware in a C-like language. In: Proc. of ACM SIGGRAPH 03.

[MJ96] Ron MacCracken and Kenneth I. Joy. Free-Form deformations with lattices of arbitrary topology. In Holly Rushmeier, editor, SIGGRAPH 96 Conference Proceedings, Annual Conference Series, pages 181–188. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[MO95] Nelson L. Max and Keiichi Ohsaki. Rendering trees from precomputed z-buffer views. In Rendering Techniques, pages 74–81, 1995.

[MP89] Gavin S. P. Miller and Andrew Pearce. Globular dynamics : A connected particle system for animating viscous fluids. Computers & Graphics, 13(3) :305–309, 1989.

[OB98] Manuel M. Oliveira and Gary Bishop: “Dynamic Shading in Image-Based Rendering”. *UNC Computer Science Technical Report TR98-023*, University of North Carolina, May 31, 1998.

[PKKG03] Mark Pauly, Richard Keiser, Leif Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. In Proceedings of ACM SIGRAPH 2003, pages 641–650, 2003.

[PZvG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels :surface elements as rendering primitives. In Proceedings of ACM SIGGRAPH 2000, pages 335–342, 2000.

[Ree83] William T. Reeves. Particle systems : a technique for modeling a class of fuzzy objects. In SIGGRAPH ’83 : Proceedings of the 10th annual conference on Computer graphics and interactive techniques, pages 359–375, New York, NY, USA, 1983.ACM Press.

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat : A multiresolution point rendering system for large meshes. In Proceedings of SIGGRAPH 2000, Computer Graphics Proceedings, pages 343–352, 2000.

[RLE+82] Smith A. R., Carpenter L., Catmull E., Cole P., Cook R., Poor T., Porter T., and Reeves W. Genesis demo documentary (film), June 1982.

[SD01] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In Proceedings of the 12th Eurographics workshop on Rendering, pages 151–162, 2001.

[SGwHS98] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pages 231–242, New York, NY, USA, 1998. ACM Press.

[Sim90] Karl Sims. Particle animation and rendering using data parallel computation. In SIGGRAPH '90 : Proceedings of the 17th annual conference on Computer graphics and interactive techniques, pages 405–413, New York, NY, USA, 1990. ACM Press.

[SJ00] Gernot Schaufler and Henrik Wann Jensen. Ray tracing point sampled geometry. In Proceedings of the Eurographics Workshop on Rendering Techniques 2000, pages 319–328, London, UK, 2000. Springer-Verlag.

[SP86] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. volume 20, pages 151–160, August 1986.

[ST92] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In SIGGRAPH '92 : Proceedings of the 19th annual conference on Computer graphics and interactive techniques, pages 185–194, New York, NY, USA, 1992. ACM Press.

[WFP+01] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Strasser. The randomized z-buffer algorithm : interactive rendering of highly complex scenes. In SIGGRAPH '01 : Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 361–370, New York, NY, USA, 2001. ACM Press.

[WK04] J. Wu and Leif Kobbelt. Optimized sub-sampling of point sets for surface splatting. In Proceedings of Eurographics 2004, pages 643–652, 2004.

[ZPvBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In Proceedings of ACM SIGGRAPH 2001, pages 371–378, 2001.