REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : IVA /M2/2020

Mémoire

présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Images et vie artificielle**

Rendu volumique par les textures 3D

Par :

OTMANE INES

Soutenu le 27/10/2020 devant le jury composé de :

Nom et prénom	Grade	
BAHI NAIMA	MCB	Rapporteur
Nom et prénom	Grade	Examinateur

REMERIEMENTS

Au terme de mon mémoire, je tiens à remercier tous ceux qui m'ont aidé et contribué à la réalisation de ce travail.

Je remercie mon encadreur Bahi naima de m'avoir encadré durant cette année de préparation du projet de master, pour ses orientations, et ses conseils, je la remercie de m'avoir donné la chance d'explorer le domaine de Rendu volumique et surtout de m'avoir encouragé pour élaborer ce travail.

Je remercie ……d'avoir accepté de présider le jury de ma soutenance et …… d'avoir accepté de faire partie du jury et d'avoir examiné ce travail.

Mes remerciements vont enfin à ma grande famille, à mes enseignants et à mes amies pour leur soutien continu.

Table des matières

INTRODUCTION 1
Chapitre 01 3
Notions fondamentales 3
1.1 Introduction 4
1.2 Donnée volumique 4
1.3 Équation du rendu volumique 6
1.4 Rendu volumique direct 8
1.4.2 Rendu volumique direct 8
1.4.3 Les algorithmes de rendu volumique direct 9
1.4.3.1 Le lancer de rayons (Ray casting) 10
Figure 5: Schéma explicatif de lancer de rayons. (Coninx, 2013) 11
1.4.3.2 Splatting 11
1.4.3.3 Shear warp 11
1.4.4 Comparaison des algorithmes de rendu de volumique direct 13
1.4.5 Textures volumiques 15
1.4.5.1 Rendu basé sur l'utilisation de textures 16
1.5 Les travaux connexes 17
1.5.1 Représentation 17
1.5.2 Techniques de rendu basé sur les couches d'image 17
1.5.3 Techniques de rendu basées purement sur l'image 18
1.5.3.1 Les movie-maps 18
1.5.3.2 Les imposteurs 18
1.5.3.3 Panoramas cylindriques 19
1.5.3.4 Le rendu Light-Field 19
1.5.3.5 Régulier de positions 20
1.5.3.5 Les mosaïques 20
1.6 La modélisation de la surface 21
1.6.1 Notion de boîte 21
1.6.2 La génération du volume de référence 23
1.7 Le rendu du nouveau modèle 24

1.7.1 Rendu de toute la surface texturée. 24
1.7.2 Rendu par boîte 24
1.7.3 Rendu par tranche identique 25
1.8 Conclusion 26
Chapitre 02 27
Conception 27
2.1 Introduction 28
2.2 Motivation 28
2.3 Objectif 28
2.4 Conception globale 29
2.4.1 Les images de Fichier des données brutes 30
2.4.2 Le tampon utilisé pour stocker les donnée brutes 30
2.4.3 Fonction de transfert 30
2.4.4 Rendu 31
2.5.5 Transparence et valeurs alpha 32
2.5.6 Les données brutes (Raw data) 32
2.6 Les étapes de construction de l'application 33
2.6.1 Etape 1 33
2.6.2 Etape 2 36
2.6.2.1 La projection orthographique 36
2.6.2.2 L'utilisation de texture 37
2.6.2.3 La plaquage de texture 38
2.6.2.4 La translation de volume 39
2.7 Conclusion 40
Chapitre 03 40
Mise en œuvre et résultats 40
3.1 Introduction 41
3.2 Environnement de l'application 41
3.2.1 Environnement de développement matériel 41
3.2.2 Environnement de développement logiciel 41
3.3 Description du système 41
3.3.1 OpenGL 41
3.3.2 Fichiers d'entête précompilés (Precompiled Header) 43
3.4 Description de l'application 44

3.4.1 Les étapes de réalisation de l'application 44
3.4.2 Les classes de l'application 45
3.4.2.1 Classe processeur-de-données 45
3.4.2.2 Classe Renderer-Helper 46
3.4.2.3 Classe Transformation-Manager 47
3.4.2.4 Classe rendu_volumique_de_dialogue 48
3.5 Problème et solution de l'application 49
Solutions 50
3.6 Résultats 51
3.6.1 Guide pour les touches clavier 51
3.6.2 Des résultats sans appliquer une fonction de transfert 52
3.6.2.1 l'utilisation des autres bases de données 54
3.6.3 Avec l'application d'une fonction de transfert 54
3.6.3.1 Selon la position du voxels 54
3.6.3.2 Selon la valeur de densité 56
3.7 Temps de calcul 58
3.8 Conclusion 59
CONCLUSION 60
RÉFÉRENCE 61

Liste des figures

Figure 1: Voxels constituant un objet volumétrique après sa discrétisation. (hadwager, 2008) 5
Figure 2: Taxinomie des représentations cellulaires en fonction de leur topologie. (hadwager, 2008) 6
Figure 3: Schéma illustrant le transport de rayon dans une scène simple (rendu volumique direct). (Game devlopement, 2014) 7
Figure 4: Rendu volumique direct. (Coninx, 2013) 10
Figure 5: Schéma explicatif de lancer de rayons. (Coninx, 2013) 11
Figure 6: L'algorithme du shear-warp avec une projection parallèle. (Levoy., 1988) 13
Figure 7: L'algorithme du shear-warp avec une projection perspective. (Levoy., 1988) 13
Figure 8: Quelque type des scènes complexes répétitives (texture 3D). (HEMIDI, 2005) 16
Figure 9: Grille cartésienne 3D. (Schott, 2011) 16
Figure 10: Imposteur dynamique et un imposteur en couches est composé de plusieurs polygones transparents (HEMIDI, 2005). 19
Figure 11: Exmple de Quicktime VR (vimeo, s.d.). 19
Figure 12: Schéma explicative de Light-Field (Schaufler, 1998) 20
Figure 13: Régulier de positions sur le plan (u, v) (Schaufler, 1998). 20
Figure 14: Fragments d'images acquises par des caméras (Schaufler, 1998) 21
Figure 15 : Plaquage d'un texel sur une surface. (HEMIDI, 2005) 22
Figure 16: Calcul des coordonnées des faces texturées d'une boîte Méthode de Meyer pour une nouvelle représentation du volume de référence. (HEMIDI, 2005) 22
Figure 17: Algorithme de remplissage de la colonne. Gauche : vue en 3D. Droite : Vue en coupe verticale. (HEMIDI, 2005) 24
Figure 18: Ordre d'affichage des faces : Cas A de 0 à n-1. Cas B de n-1 à 0 (HEMIDI, 2005). 24
Figure 19: Architecture générale. 29
Figure 20: Schéma de fonction de transfert 31
Figure 21: Schéma explique les étapes de création de la texture 3D. 31
Figure 22: La valeur d'alpha à gauche inférieure à 0.5 au milieu égale 0.5 adroite supérieure à 0.5 32
Figure 23: Séquence d'images d'une tète humain avec ces tailles et ces positions dans l'axe Z. 33

Figure 24 : Schéma représente la conversion des images 2D à un fichier Raw data..... 34
Figure 25 : Schéma montrant les étapes de lecture et de la construction des données. 35
Figure 26: Schéma montre la suppression du l'information de profondeur. 37
Figure 27: Les valeurs d'orthogonale qui nous avons utilisé. 37
Figure 28: Texture 3D normalisée. 38
Figure 29: Schéma montre la placage de texture sur le Quad. 38
Figure 30 : La placage de texture sur le Quad et le rendu de l'image finale. 39
Figure 31: Le volume au milieu de repère global de la scène. 39
Figure 32: la transmission de données depuis le CPU vers le GPU (SERRANO, 2015). 42
Figure 33: Contenu de l'entête précompilé. 44
Figure 34: Représentation des vues d'une tête humaine par scanner sans l'application de fonction de transfert. 53
Figure 35: Représentation des vues d'une tête humaine par scanner après avoir changé la valeur alpha. 53
Figure 36: Représentation des vues d'une tête humaine avec le crâne partiellement retiré. ... 54
Figure 37: Etude CT d'une tête de cadavre. 54
Figure 38: Des rotations sur l'axe Y. 55
Figure 39: Des rotations sur l'axe X. 55
Figure 40: Représentation des vues d'une tête humaine par scanner après avoir appliqué la fonction de transfert. 56
Figure 41: Les étapes d'affichage des tranches de volume de gauche à droite. 57
Figure 42: Une coupe longitudinale de la boite crânienne. 58
Figure 43: Comparaison du temps de calcul entre nos modèles. 59

Liste des tableaux

Tableau 1 : La comparaison des différents algorithmes de rendu de volume (HEMIDI, 2005). 15
Tableau 2 : Quelques techniques de rendu basées sur l'image ou sur le spectre géométrie/image (Lakhdar, 2015)..... 18
Tableau 3: Raccourcis clavier pour les opérations disponible dans l'application. 52
Tableau 4 : Temps de calcul des différents modèles. 58

INTRODUCTION

A l'heure actuelle, la visualisation des données volumique est l'un des axes majeur de la recherche. Ce domaine très vaste grâce à la visualisation scientifique et l'infographie , dans laquelle le rendu de volume est un ensemble de techniques utilisées pour afficher une projection 2D d'un ensemble de données échantillonnées discrètement 3D (M. Ikits, 2007) .Ces algorithmes les plus populaires et les plus récents sont souvent classées en deux catégories, la première catégorie est constituée d'algorithmes dits de rendu volumique direct et la seconde est constituée d'autres algorithmes dits de rendu volumique indirect. Nous sommes intéresse ici uniquement aux algorithmes de rendu volumique direct qui nécessite que chaque valeur échantillonnée au sein du volume soit associé à une opacité et une couleur.

Les images de synthèse sont omniprésentes dans notre environnement, que ce soit dans notre vie personnelle ou professionnelle l'homme a toujours cherché à représenter visuellement tant son environnement réel que son monde imaginaire, dans notre cas l'imagerie médicale est le domaine que nous avons représenté comme exemple dans notre étude ci présente.

L'imagerie médicale est certainement l'un des domaines de la médecine qui a le plus progressé dans ces vingt dernières années. Ces récentes découvertes permettent non seulement un meilleur diagnostic mais offrent aussi de nouveaux espoirs de traitement pour de nombreuses maladies. Cancer, épilepsie… l'identification précise de la lésion facilite déjà le recours à la chirurgie, seule solution thérapeutique pour certains malades (Doctissimo, 2017). De telles techniques permettent également de mieux comprendre le fonctionnement de certains organes encore mystérieux, comme le cerveau.

Pour bien appliquer le domaine d'imagerie médicale on utilise la technique de visualisation scientifique en fonction des images médicales. La visualisation scientifique s'applique à des domaines larges d'activités tels que la visualisation de données :

- Médicales (données 2D, 3D, ainsi que temporelles).
- Biologiques (ADN, molécule, etc)
- Mécaniques (détection de zones sous contraintes, sous pression, etc)
- Physique (plasmas, physique des particules, etc)
- Mathématique (surfaces abstraites, haute dimensions)
- Capteurs (images satellites, fluide, etc)

- de Graphes (Big Data, données internet, financières, etc)

Le type de donnée visualisé peut être varié. Il peut s'agir de l'affichage de données scalaires (champs de pression, température, etc), vectorielles (vitesse, gradients, etc), tensorielles (contraintes mécanique, diffusion dans le cerveau ou le coeur, etc). Ces données peuvent être difficiles à visualiser de par leur nature complexe, du bruit d'une acquisition, ou de leur grand nombre.

Notre objectif consiste applique le rendu volumique à base de texture volumique Le but de ce travail est d'accélérer considérablement les calculs de rendu en exploitant la représentation à base d'image des données volumiques médicales. Sont stockées dans une Raw data. L'algorithme à base de textures 3D est avantageux par rapport à d'autres techniques de visualisation de volume 3D en raison de sa qualité d'image élevée (Lakhdar, 2015), et le gain de temps et sur tout de montré comment faire un placage de texture sur un volume qui ne contient pas de surface, et à pouvoir retenir les données médicales qui sont issues de l'acquisition de différente modalité servant à reproduire l'ensemble des organes du corps et des matériaux scannés. Dans ce domaine la qualité et le temps du rendu graphique sont primordiaux afin que le diagnostic du médecin devienne précis et immédiat.

Pour arriver à nos fins nous avons organisé notre mémoire comme suit :

Le chapitre un est consacré à la description de l'équation de rendu et le rendu volumique, puis à la définition de rendu volumique direct et ces méthodes. Le chapitre sera achevé par la définition de texture volumique et les méthodes qui ont basé texture, puis une aperçue de quelques travaux connexes.

Dans le chapitre deux nous commençons par donner un aperçu sur l'architecture générale de notre application avec une petite explication de tous les éléments, puis on expliquant brièvement les classes utilisé dans notre application.

Dans La troisième chapitre, se fond sur la précision de l'environnement de développement matériel et logiciel utilisé pour mettre en œuvre notre application et l'analyse des résultats obtenus.

Enfin, ce mémoire se termine par une conclusion et des perspectives.

Chapitre 01

Notions fondamentales

1.1 Introduction

La visualisation de volume est une méthode d'extraction d'informations significatives à partir de données volumiques à l'aide d'images et de graphiques interactifs. Il concerne la représentation, la modélisation, la manipulation et le rendu des données de volume. Les données volumiques sont des entités 3D (pouvant varier dans le temps) qui peuvent contenir des informations, ne pas consister de surfaces et de coins tangibles, ou être trop volumineuses pour être représentées géométriquement. Elles sont obtenues par des techniques d'échantillonnage, de simulation ou de modélisation. . Par exemple, une séquence de coupes 2D obtenues par imagerie par résonance magnétique (IRM), tomodensitométrie (TDM), IRM fonctionnelle (IRMf) ou tomographie par émission de positrons (TEP) est reconstruite en 3D.

1.2 Donnée volumique

Un ensemble de données volumiques est généralement un ensemble V d'échantillons (x, y, z, v), également appelés *voxels*, représentant la valeur v d'une propriété de données, à un emplacement 3D (x, y, z) (Figure.1.). Si la valeur est simplement un 0 ou un entier i dans un ensemble I, avec une valeur de 0 indiquant l'arrière-plan et la valeur de i indiquant la présence d'un objet Oi, les données sont alors appelées données binaires. Les données peuvent au contraire être à valeurs multiples, la valeur représentant une propriété mesurable des données, y compris, la couleur, la densité, la chaleur ou la pression. La valeur v peut même être un vecteur, représentant la vitesse à chaque emplacement, résulte de multiples modalités de balayage, telles que l'imagerie anatomique (CT, IRM) et fonctionnelle (PERT, IRMf) ou couleur (RVB). (Benazouz, 2003)

Cependant, les données de volume peuvent varier dans le temps, quel que soit le cas, V devient un ensemble d'échantillons 4D (x, y, z, t, v) .En général, les échantillons peuvent être prélevés à des emplacements purement aléatoires dans l'espace, mais dans la plupart des cas, l'ensemble V est isotrope. (Benazouz, 2003)

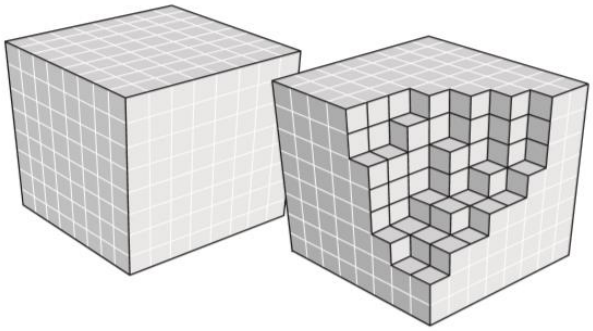


Figure 1: Voxels constituant un objet volumétrique après sa discrétisation. (hadwager, 2008)

Échantillons prélevés à intervalles réguliers le long des trois axes orthogonaux. Lorsque l'espacement entre les échantillons le long de chaque axe est constant, mais qu'il peut y avoir trois constantes d'espacement différent pour les trois axes, l'ensemble V est anisotrope. Étant donné que l'ensemble d'échantillons est défini sur une grille régulière, un tableau 3D (également appelé tampon de volume, raster 3D ou simplement volume) est généralement utilisé pour stocker les valeurs, l'emplacement de l'élément indiquant la position de l'échantillon sur la grille. Pour cette raison, l'ensemble V sera appelé le tableau de valeurs V (x, y, z), qui est défini uniquement aux emplacements de grille. En variante, on utilise des grilles rectilignes, curvilignes (structurées) ou non structurées. Dans une grille rectiligne, les cellules sont alignées par zone, mais les espacements de grille le long des axes sont arbitraires. Tant qu'une grille a été transformée de manière non linéaire tout en préservant la topologie de la grille, elle devient curviligne. Habituellement, la grille rectiligne définissant l'organisation logique est appelée espace de calcul, et la grille curviligne est appelée espace physique. Sinon, la grille est dite non structurée ou irrégulière. Un volume de données irrégulier ou irrégulier est un ensemble de cellules dont la connectivité doit être spécifiée explicitement. Ces cellules peuvent être de formes arbitraires telles que des tétraèdres, des hexaèdres ou des prismes. (Arie, (2005).)

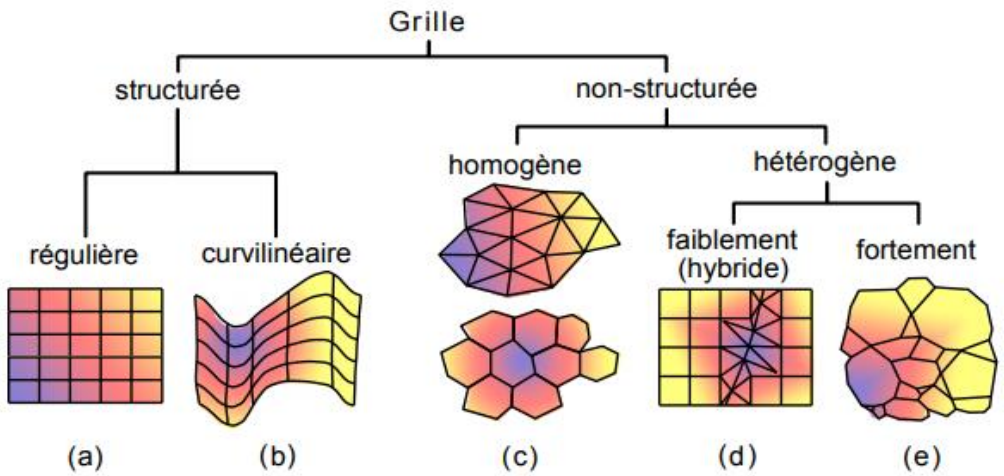


Figure 2: Taxinomie des représentations cellulaires en fonction de leur topologie.

(hadwager, 2008)

1.3 Équation du rendu volumique

Une fonction scalaire sur un volume 3D peut être visualisée de différentes manières, par exemple en contours de couleur sur une coupe 2D ou par approximation polygonale d'une surface de contour. Volume direct le rendu fait référence à des techniques qui produisent une image projetée directement à partir des données de volume, sans constructions intermédiaires telles que les polygones de surface de contour. Ces techniques nécessitent quelques modèle de la manière dont le volume de données génère, réfléchit, diffuse ou occulte la lumière. Cette partie présente une séquence de tels modèles optiques avec des degrés croissants de réalisme physique, ce qui peut faire ressortir différentes caractéristiques des données. (Max, 1995)

L'objectif de base du rendu de volume est de trouver une bonne approximation du modèle optique qui exprime la relation entre l'intensité du volume et la fonction de l'opacité et l'intensité dans le plan d'image. Dans cette section, nous décrivons un modèle physique typique basé sur les algorithmes de rendu de volume. Notre intention est d'expliquer les fondements théoriques sur les algorithmes de rendu de volume. Les modèles optiques physiques bénéficient en théorie du transport radiatif qui tente d'afficher un volume comme un nuage peuplé des particules. Le transport de la lumière est étudié en tenant compte des différents phénomènes à l'œuvre. La lumière d'une source peut être soit dispersée ou absorbée par les particules. Il pourrait y avoir une augmentation nette lorsque les particules émettent eux-mêmes la lumière. Les modèles qui tiennent compte de tous les phénomènes ont tendance à être très compliqués.

Dans la pratique, les modèles locaux beaucoup plus simples sont utilisés par la plupart des algorithmes de rendu de volume standards utilisent une équation intégrale de rendu de volume (Max, 1995) .Définit par :

$$I_\lambda(x,y) = \int_0^L C_\lambda(S)\mu(s)e^{-\int_0^s \mu(t)dt}ds$$

Equation .1.

- I_λ: est la quantité de lumière de la longueur d'onde λ venant de la direction de rayon **r** qui est reçu à l'emplacement x sur le plan d'image.
- L: est la longueur du rayon **r.**
- **μ**: est la densité des particules en volume qui reçoivent la lumière provenant de toutes les sources lumineuses environnantes et qui reflètent cette lumière vers l'observateur en fonction de leur sécularité et les propriétés des matériaux diffus.
- c_λ: est la lumière de longueur d'onde λ réfléchie et/ou émise au lieu de s dans la direction de **r**.

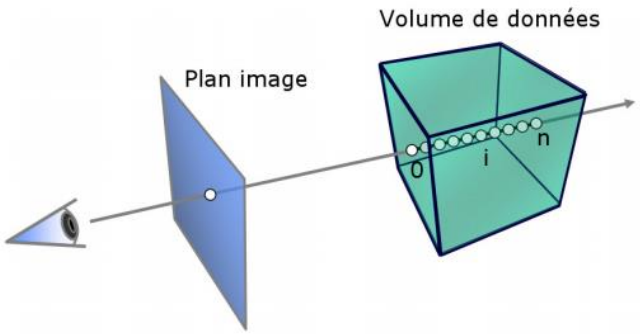


Figure 3: Schéma illustrant le transport de rayon dans une scène simple (rendu volumique direct). (Game devlopement, 2014)

Dans le cas général, l'équation ne peut pas être calculée analytiquement. Par conséquent, dans des applications pratiques, la plupart des algorithmes de rendu de volume doivent obtenir une solution numérique de l'équation grâce à l'emploi d'une quadrature d'ordre zéro de l'intégrale intérieure avec une approximation de premier ordre de l'exponentielle. L'intégrale extérieure est également résolue par une somme finie des échantillons uniformes. (Max, 1995)Alors nous obtenons l'équation suivante :

$$I_\lambda(x,y) = \sum_{k=1}^{M} C_\lambda(s_k)\alpha(s_k) \prod_{i=1}^{k-1}(1-\alpha(s_i))$$

Equation .2.

- $\boldsymbol{\alpha(s_k)}$: sont les échantillons d'opacité le long du rayon,
- $\boldsymbol{C_\lambda\ (s_k)}$: sont les valeurs de couleurs locales dérivées du modèle d'illumination,
- Cette expression est désigner comme discrétisé, où l'opacité $\boldsymbol{\alpha}$ **=1.0-transparence**.

L'équation représente un cadre théorique commun à tous les algorithmes de rendu de volume. Qui permettent d'obtenir des couleurs et des opacités dans des intervalles discrets le long d'un chemin linéaire et les associes en avant vers l'arrière. Cependant, les algorithmes peuvent se distingués par le procédé dans lequel,

- Les couleurs$\boldsymbol{C_\lambda\ (s_k)}$ et les opacités $\boldsymbol{\alpha(s_k)}$ sont calculés à chaque intervalle **k**.
- la largeur de l'intervalle $\boldsymbol{\Delta s}$ est choisie. **C** et $\boldsymbol{\alpha}$ sont maintenant des fonctions de transfert, généralement mise en œuvre comme des tables de consultation. Les densités de volume brutes sont utilisées pour indexer les fonctions de transfert pour la couleur et l'opacité, donc les petits détails des données de volume peuvent être exprimés dans l'image finale par l'utilisation de différentes fonctions de transfert (Max, 1995).

1.4 Rendu volumique direct

Dans cette partie, nous allons décrire les algorithmes de rendu volumique les plus populaires et les plus récents. Ces méthodes sont souvent classées en deux catégories, la première catégorie est constituée d'algorithmes dits de rendu volumique direct et la seconde est constituée d'autres algorithmes dits de rendu volumique indirect. Nous sommes intéresse ici uniquement aux algorithmes de *rendu volumique direct* (Roussel, 2007).

1.4.2 Rendu volumique direct

Le rendu volumique direct nécessite que chaque valeur échantillonnée au sein du volume soit associé à une opacité et une couleur.

Cela est effectué grâce à ce que l'on appelle une fonction de transfert F :

F (simple value) → (couleur, opacité)

Celle-ci est généralement modélisée par une fonction linéaire par partie (tableau à deux dimensions contenant les valeurs des points anguleux de la fonction) mais également par tous autres types fonctions (fonction en escalier, ...).

Les valeurs d'opacité et de couleur sont converties en composantes RGBA (canaux red, green, blue pour la couleur, et canal alpha pour l'opacité) et le composé résultant est projeté sur le pixel correspondant du *frame buffer*. La façon dont cela est fait dépend de la technique de rendu. Ces techniques peuvent être combinées afin de tirer parti des différents avantages qu'elles comportent.

Par exemple une implémentation de type *shear sharp* peut utiliser les routines de texture matériel de la carte graphique afin de dessiner les tranches alignées.

1.4.3 Les algorithmes de rendu volumique direct

Principe

Les algorithmes de rendu volumique direct utilisent les données originelles du volume. Chaque pixel de l'image est considère comme un rayon qui traverse un volume (voir figure 4). La complexité de l'algorithme réside ici dans la manière dont le rayon va interagir avec les échantillons (ou voxels) du volume. Ainsi, tous les algorithmes de rendu volumique direct fonctionnent en échantillonnant des valeurs le long du rayon et en calculant la contribution de ces valeurs, suivant un modèle de rendu, à la couleur finale du pixel. Il existe en fait deux manières possibles de calculer l'image finale (Coninx, 2013).

La première consiste à calculer la contribution de chaque élément du volume sur les pixels de l'image. Cette façon de calculer l'image est appelée object-order car les voxels sont passés en revue un par un dans un ordre de visibilité donne.

A l'inverse, la seconde manière consiste à calculer chacun des pixels de l'image un par un. Il faut alors pour chaque rayon traverser le volume afin de trouver les voxels qui contribuent à la couleur finale. On peut ainsi éviter de parcourir les parties cachées du volume. Cette méthode est appelée image-ordre (Coninx, 2013).

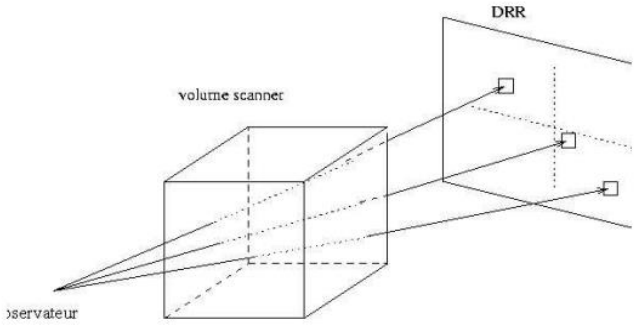


Figure 4: Rendu volumique direct. (Coninx, 2013)

1.4.3.1 Le lancer de rayons (Ray casting)

Le moyen le plus commun de projeter une image 3D est le lancer de rayons à travers le volume. Cette technique consiste à générer un rayon pour chaque pixel de l'image désirée. L'algorithme de lancer de rayons est l'algorithme le plus célèbre dans la catégorie "image-ordre". Il consiste à émettre des rayons à partir du point de vue jusqu'au plan de l'image pour chaque pixel en traversant le volume. Lorsqu'un rayon traverse le volume, le volume est échantillonne selon un certain nombre d'intervalles, en général, un ou deux par voxel. La valeur à chaque point d'échantillonnage est obtenue par une interpolation tri linéaire des valeurs voisines de ce point. Les contributions de ces échantillons sont accumulées jusqu'à ce que le rayon ait quitté le volume. La valeur finale est alors calculée suivant la formule d'accumulation et placée sur le pixel que le rayon atteint (Levoy., 1988). Le principal avantage du lancer de rayons est la qualité du rendu (Coninx, 2013). Cependant, cet algorithme étant un algorithme de type image-ordre, il n'accède pas aux données du volume dans l'ordre naturel du stockage, puisqu'il le traverse dans des directions arbitraires. De fait, il passe plus de temps à calculer les positions des points d'échantillonnage et à effectuer les calculs d'adresses qu'à calculer le rendu. De plus, les rayons peuvent passer plusieurs fois par le même voxel, et ainsi il est nécessaire de recharger des données déjà chargées précédemment. Enfin, l'élimination des régions vides du volume, bien que faisable, est moins facile qu'avec des méthodes de type object-order, et donc plus couteuse (Coninx, 2013).

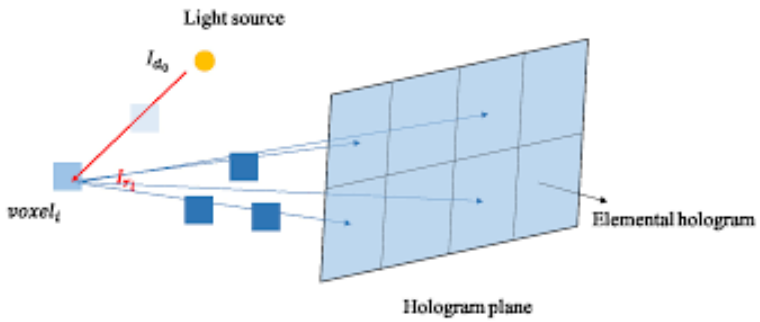


Figure 5: Schéma explicatif de lancer de rayons. (Coninx, 2013)

1.4.3.2 Splatting

Cette méthode a été proposée par Westover (Westover., 1990), cet algorithme est de type object-order. Cette méthode consiste à projeter voxel par voxel le volume sur le plan image. En projection parallèle, l'empreinte d'un voxel sur ce plan de vue est constante pour tous les voxels. Comme un voxel ne se projetera pas exactement sur un pixel, un filtre est utilisé pour calculer une contribution en couleur et en transparence aux pixels voisins (Levoy, 1994). Comme cet algorithme itère sur le volume, il peut parcourir celui-ci selon son ordre naturel de stockage. En revanche, le calcul du filtre d'échantillonnage est très couteux. En théorie, cet algorithme peut fournir les mêmes images qu'avec l'algorithme du lancer de rayon. En pratique, du fait que le calcul des paramètres des filtres est difficile, des approximations sont utilisées, qui permettent soit d'obtenir une exécution efficace, soit des images de qualité, mais jamais les deux simultanément.

1.4.3.3 Shear warp

L'algorithme shear-warp a été proposé par Lacroute et al (Levoy, 1994). Et est actuellement considère comme l'algorithme de rendu volumique le plus rapide (Feschet., 1999).Il est basé sur la factorisation de la transformation liée au point de vue pour simplifier la projection du volume vers l'image. Dans cet algorithme, on n'itère pas sur les pixels de l'image, mais sur les voxels de la scène. Cependant, on évite la complexité des algorithmes de splatting en décomposant la projection des voxels sur le plan image en deux étapes :

1. Une première étape de composition appelée "shear", composée de la transformation des données du volume dans l'espace objet transformé et de l'accumulation de ces données sur le plan image. Cette étape crée une image intermédiaire "distordue" correspondant à l'espace objet transformé.

2. Une deuxième étape appelée "warp", transformant l'image intermédiaire en image finale.

Passer par une image intermédiaire permet aux lignes de balayage des données du volume d'être alignées avec les lignes de balayage de l'image intermédiaire. On balaye alors l'objet et l'image simultanément dans leur ordre de stockage. Cet algorithme combine les avantages des algorithmes d'ordre objet et des algorithmes d'ordre image. Les conditions d'utilisation des techniques d'optimisation séquentielles telles que l'exploitation de la cohérence de données et la terminaison anticipée de rayon sont réunies grâce à ce parcours simultané des deux structures de données (objet et image) (Feschet., 1999).

L'implantation de l'algorithme du shear-warp peut être décompose en trois unités fonctionnelles : calcul d'une table d'opacité, l'étape de composition et l'´étape de warp de l'image intermédiaire.

Le calcul de la table d'opacité est un pré calcul d'opacités. Le calcul des opacités est dépendant du point de vue. Il dépend de l'intervalle d'échantillonnage. Le volume de données contient des opacités associées à un intervalle $\Delta x0$. On calcule à chaque nouveau point de vue la correction nécessaire à chacune de ces valeurs pour le nouvel intervalle Δx, d'où le calcul d'une table d'opacité

Pour la phase de composition qui est la combinaison des étapes de transformation du volume et d'accumulation, on doit échantillonner chaque coupe de voxels lorsqu'elle est translatée dans l'espace objet déformé. L'interpolation utilisée est l'interpolation bilinéaire (contrairement à la plupart des algorithmes de rendu volumique qui utilisent l'interpolation tri linéaire) utilisant deux lignes de balayage d'une coupe pour produire une ligne résultat dans l'image intermédiaire. Une manière simple d'accumuler les opacités le long du rayon consiste à les additionner (Feschet., 1999).

L'étape de "warp" consiste en la projection de l'image intermédiaire distordue sur le plan image, pour obtenir l'image finale. Il est clair que le principal avantage de cette méthode est sa rapidité de rendu. Elle cumule ainsi beaucoup d'avantages qui vont dans ce sens. La mémoire est parcourue de

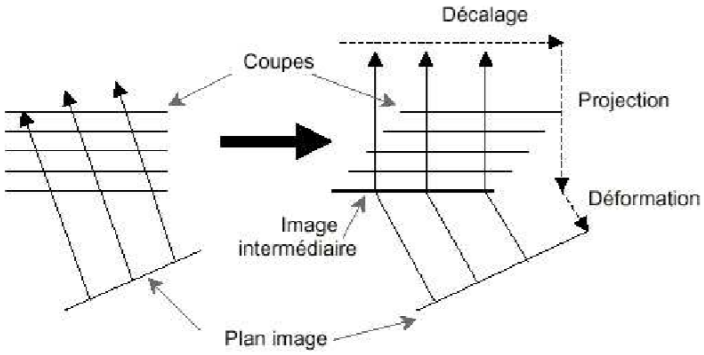


Figure 6: L'algorithme du shear-warp avec une projection parallèle. (Levoy., 1988)

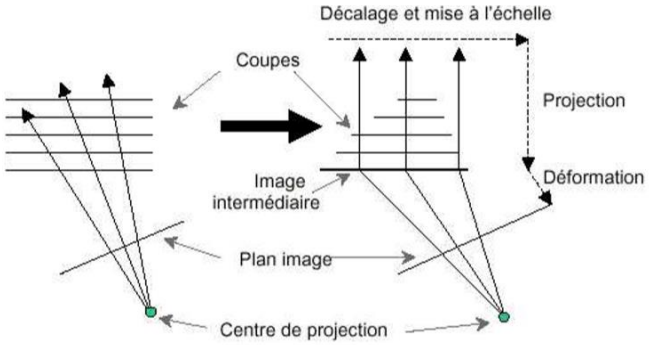


Figure 7: L'algorithme du shear-warp avec une projection perspective. (Levoy., 1988)

Manière optimale que ce soit pour l'accès au volume ou pour l'accès au plan image. La projection est aussi très simplifier (donc rapide) puisqu'il y a correspondance entre un voxel et un pixel. Cependant cette méthode comporte beaucoup d'inconvénients quant à la qualité de l'image résultante. Le fait de considérer le volume comme une série de coupes et d'utiliser une reconstruction bilinéaire au lieu d'une reconstruction globale du volume génère des artefacts lorsque l'angle de vue s'approche des 45 degrés (Feschet., 1999).

1.4.4 Comparaison des algorithmes de rendu de volumique direct

Chaque approche de rendu de volume possède ses propres avantages et inconvénients. Généralement, lorsque l'approche de rendu vise à réaliser des images de haute qualité, elle doit perdre un peu de performance, et vice versa. Shear warp et placage de texture 3D sont conçus pour maximiser les fréquences d'images sur le détriment de la qualité de l'image, tandis que splatting et raycasting sont conçus pour obtenir une haute qualité d'image sur détriment de la performance. Habituellement, la qualité d'image obtenue avec un placage de texture montre de graves artefacts dus à la non-opacité des couleurs pondérées, ainsi que l'escalier. Celui-ci est

dû à la précision de bits limitée du tampon d'image, et peut être réduite en augmentant le nombre de tranches. Certaines approches ont été proposées pour améliorer la qualité de rendu de volume basée sur la texture d'image (Rezk-Salama, 2000). L'image créée à l'aide de shear-warp montre un aliasing de flou significatif sous la forme d'escalier. Cela est dû à la fréquence d'échantillonnage de rayons étant inférieur à 1,0 et peut être gênant dans la visualisation. Splatting aligné à l'image offre une qualité de rendu similaire à celle du raycasting. Cependant, il produit des images plus lisses due au noyau de z-moyenne et l'effet anti-aliasing du la plus grand filtre gaussien. Shear-warp est toujours sensiblement plus rapides que ceux de raycasting et Splatting, dans la plupart des cas par un ordre de un ou deux amplitudes. Splatting peut générer une image de haute qualité plus rapidement que raycasting lors de la visualisation des données de grand volume, et ne provoque pas le flou extensif de shear-warp. Contrairement à raycasting, splatting considère chaque voxel une seule fois (pour une interpolation 2D sur l'écran) et non plusieurs fois (pour une interpolation 3D). En outre, comme une approche basée objet-ordre, seuls les voxels concernés doivent être pris en considération, qui, dans de nombreux cas, ne représentent que 10 % des voxels de volume (Arie, (2005).). Le tableau 1.1 établit une comparaison des caractéristiques distinctives et les différences conceptuelles des algorithmes typiques de rendu de volume.

	Raycasting	**Splatting**	**Shear-warp**
Taux d'échantillonnage	Sélection libre	Sélection libre	Fixé [1.0 ,0.58]
Evaluation d'échantillon	Post-classification	Moyenne à travers de Δ_s	Point échantillonné
Noyau d'interpolation	Tri linéaire	Gaussienne	Bilinéaire
Pipeline de rendu	Post-classification	Post-classification	Post-classification couleur opacité pondéré
Accélération	La terminaison précoce de rayon	La terminaison précoce de rayon	Encodage d'opacité RLE
Précision /canal	Virgule flottent	Virgule flottent	Virgule flottent
Voxel considéré	Tout	Pertinent	Pertinent
Vitesse	$+^a$	+	++
qualité	+++	+++	++
Projection perspective	y^b	Y	Y
Irrégulier grille	Y	Y	Y
Accélération matérielle	Spécial	Y	Y
Combinassent avec polygones	Spécial	N	N
Information richesse	++	++	++

$+^a$: *générale* , ++ *: Bon,* +++*très bon ;*

y^b: *oui*, N: *non*

Tableau 1 : La comparaison des différents algorithmes de rendu de volume *(HEMIDI, 2005).*

1.4.5 Textures volumiques

Les textures volumiques offrent un rendu réaliste des scènes complexes répétitives. Qui n'ont pas une surface bien définie (fourrure, certains organes humain internes, … etc.), Parmi les approches de rendu à base d'image, traitant les problèmes de la complexité géométrique et temporelle, se présente celle de rendu à base de couches d'images. Cette technique permet un rendu réaliste de scènes et d'objets à un moindre coût en calcul. Ce gain en temps de calcul est dû à l'exploitation des capacités des cartes graphiques. De ce fait, chaque couche d'image doit être représentée par un polygone texturé. Pour le rendu, il suffit de projeter et de composer successivement les couches sur le plan image (ZBuffer) et obtenir, ainsi, l'image finale. Chaque couche projetée est combinée avec le résultat précédent en tenant compte de la densité de chaque pixel. La projection d'une couche est plus rapide que les calculs de projection pour

chaque voxel le long d'un rayon, ce qui permet un gain de temps appréciable. (HEMIDI, 2005).

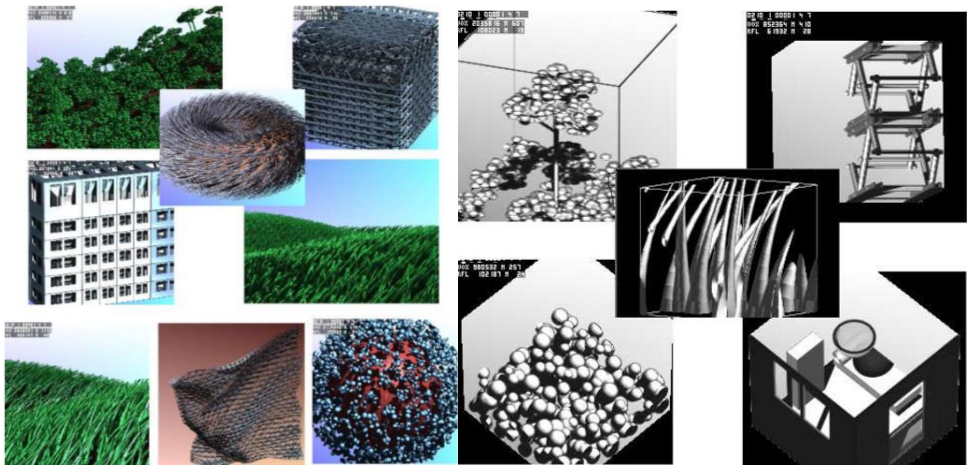


Figure 8: Quelque type des scènes complexes répétitives (texture 3D). (HEMIDI, 2005)

1.4.5.1 Rendu basé sur l'utilisation de textures

L'idée est de considérer le volume comme une texture 3D. L'utilisation de l'application de textures en rendu volumique implique une contrainte forte sur la structure des données volumiques : les données doivent être sous la forme d'une grille cartésienne de voxels. Toute autre structuration impose un ré-échantillonnage des données vers une grille cartésienne pour la visualisation (Bahi, 2017).

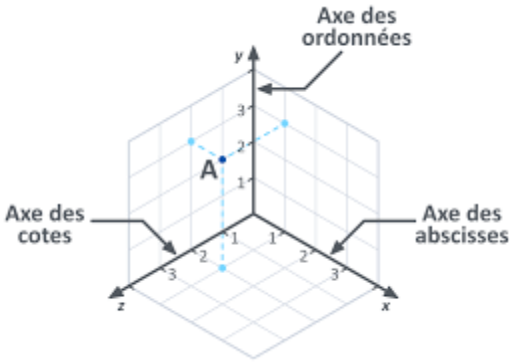


Figure 9: Grille cartésienne 3D. (Schott, 2011)

Cette contrainte est due au fait que les textures sont des tableaux, ils correspondent donc parfaitement au maillage cartésien. Des coupes parallèles au plan image et interpolées tri-linéairement peuvent alors être extraites de cette texture. Les textures 3D permettent d'effectuer la majorité des calculs par les GPU qui possèdent des mémoires généralement plus

performantes que la mémoire centrale de l'ordinateur. Mais, le volume entier est traité alors que généralement seule une petite partie du volume contient l'information nécessaire au rendu. D'autre part, le nombre de plans à extraire pour avoir une bonne qualité de rendu est assez grande. Enfin le volume ne tient pas tout le temps dans la mémoire vidéo, ce qui force alors des échanges entre cette mémoire et la mémoire centrale et ralentit considérablement les temps de rendu (Bahi, 2017) .

1.5 Les travaux connexes

1.5.1 Représentation

L'idée d'avoir une couche épaisse sur un objet est une notion qui a été introduite par Kajiya et Kay en 1989, en proposant un modèle fourrure. Par la suite ce modèle a largement été amélioré par F. Neyret (Neyret., 2005). Pour avoir un modèle général plus efficace en évitant les contraintes soulevées par Kajiya. Une primitive de réflectance particulière, comme c'est le cas dans le modèle de Kajiya et Kay, n'autorise que des objets particuliers. F. Neyret a donc proposé une primitive paramétrable (l'ellipsoïde) qui est capable de modéliser de nombreux types de formes.

Cependant, le modèle des textures volumiques introduit par Neyret, valable pour le lancer de rayon, n'était pas adapté à l'utilisation avec un algorithme comme le z-buffer et une représentation du volume de référence en couches. On l'a donc adapté au traitement en couche et au z-buffer de manière à pouvoir utiliser les fonctionnalités d'OpenGL (Neyret., 2005).

1.5.2 Techniques de rendu basé sur les couches d'image

Les techniques de placage de textures comptent sur des fonctions de placage pour spécifier le rapport des coordonnées spatiales d'image de la texture (2D) avec leurs positions correspondantes sur un modèle tridimensionnel. La spécification de ce placage est difficile, consomme du temps et exige souvent une intervention humaine considérable. En conséquence, les méthodes de placage utilisées généralement sont limitées aux descriptions géométriques très simples, comme les facettes polygonales, les sphères et les cylindres (Lakhdar, 2015). Récemment, des techniques de modélisation et de rendu basées sur l'image ont gagné une attention considérable à cause de leur potentiel de création d'images très réalistes. En utilisant des images (qui peuvent être synthétisées ou prises du monde réel) comme primitives de modélisation et de rendu, ces approches permettent de pallier deux problèmes importants et permanents dans l'infographie :

- Le besoin de techniques de modélisation plus simples appropriées à la représentation de scènes complexes,
- Le besoin de l'accélération de rendu.

On peut classifier ces techniques selon leurs positions relatives le long du spectre géométrie/image (indiqué par un point) (Lakhdar, 2015).

	Images	Hybride	Géométrie
IBR	• Panoramas cylindriques	• Images avec profondeur	• Lumigraph
	• Mosaïques concentriques	• LDIs	• Light-Field surfacique
	• Rendu Light-Field	• Objets bases image	• Placage Light-Field
		• Textures de relief	

Tableau 2 : Quelques techniques de rendu basées sur l'image ou sur le spectre géométrie/image (Lakhdar, 2015).

1.5.3 Techniques de rendu basées purement sur l'image

Parmi les techniques IBR pures, utilisant seulement quelques images capturées de scènes puis ré échantillonnées pour le rendu de nouvelles vues, on peut citer les techniques suivantes :

1.5.3.1 Les movie-maps

Elles sont introduites en 1980 par lippmann permettent (Lippmann, 1980), de créer une base de données contenant des milliers d'images de la scène et de les stocker sur un support interactif. Le processus de rendu est remplacé par une interrogation d'une base de données dans un ensemble énorme d'images de référence pour afficher celle la plus proche du point de vision virtuelle. Cette technique était incapable de reconstruire toutes les vues désirables malgré le développement rapide des médias de stockage de très haute capacité. (HEMIDI, 2005)

1.5.3.2 Les imposteurs

Les imposteurs sont des techniques permettant le placage d'images (imposteurs ou sprites) sur un plan pour représenter la partie réelle de la scène. Une simple image ne peut pas fournir la parallaxe appropriée (l'apparition d'effets d'occlusion ou de masquage). Pour cela, deux extensions ont été introduites : La première ajoute une information de profondeur à l'image alors que la deuxième utilise une maille approximative sous-jacente et plaque des images sur la géométrie. Plusieurs techniques d'imposteurs sont présentes dans le domaine. Parmi celles-ci, on peut citer : les imposteurs statiques, les imposteurs dynamiques, les nailboards et imposteurs avec couches et les imposteurs maillés. (HEMIDI, 2005)

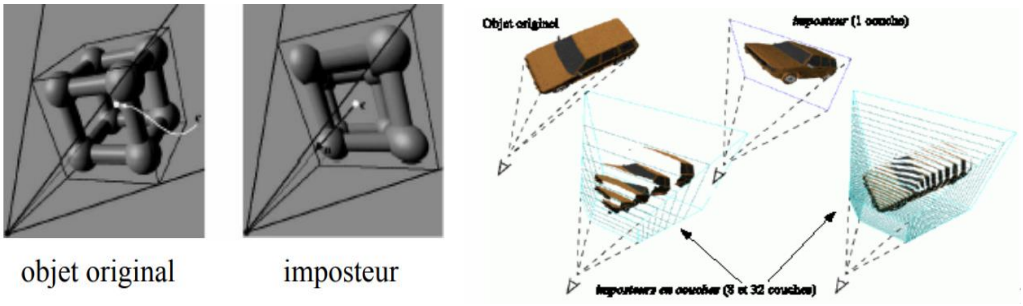


Figure 10: Imposteur dynamique et un imposteur en couches est composé de plusieurs polygones transparents (HEMIDI, 2005).

1.5.3.3 Panoramas cylindriques

Dans la technique des panoramas cylindriques, comme par exemple le QuickTimeVR, une scène est représentée par un ensemble d'images cylindriques pour fournir une indépendance d'orientation horizontale en explorant un environnement à partir d'un seul point. Une nouvelle vue est synthétisée en déformant une image cylindrique. Cette technique maintien un taux d'affichage interactif, mais n'est pas appropriée à la promenade dans des environnements virtuels parce que l'observateur est fixé à un seul emplacement. (HEMIDI, 2005)

Figure 11: Exmple de Quicktime VR (vimeo, 2017).

1.5.3.4 Le rendu Light-Field

Permet de paramétrer des rayons par leurs intersections avec deux plans arbitraires (u, v et s, t). Dans leurs travaux, les auteurs ont construit une caméra commandée par ordinateur pour l'acquisition du champ de lumière d'une scène réelle. La caméra est translatée à travers un réseau. (HEMIDI, 2005)

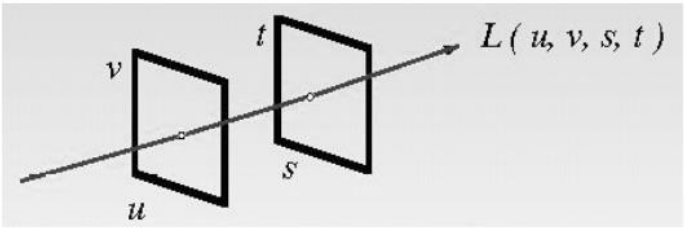


Figure 12: Schéma explicative de Light-Field (Schaufler, 1998).

1.5.3.5 Régulier de positions

Dans le plan (u, v) une image acquise pour chaque position. Chacune de ces images est ensuite projetée (par placage de texture) sur le second plan (s, t). (Gortel et al). Ont développé le système Lumigraph en même temps que les Light-Fields. Ils ont utilisé la même méthode du Light-Field pour paramétrer un rayon dans l'espace 3D. Les échantillons de Lumigraph peuvent être acquis en utilisant une caméra régulière prise à la main. Puisqu'une caméra arbitrairement positionnée est utilisée, les positions d'échantillonnage ne peuvent être ni spécifiées ni contrôlées ; ce qui ne garantit pas que ces échantillons soient régulièrement répartis. Ces deux techniques Sont robustes et simples à implanter, mais elles exigent un temps de calcul et un espace de stockage très grand. (HEMIDI, 2005)

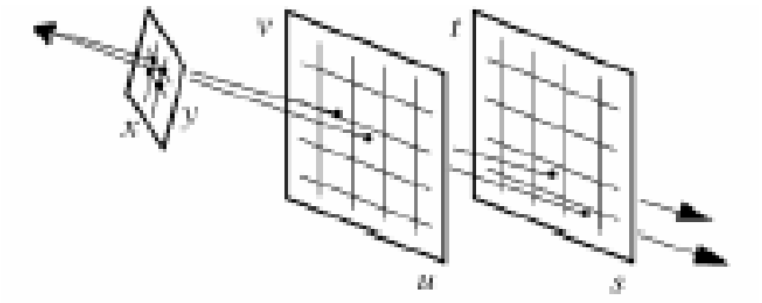


Figure 13: Régulier de positions sur le plan (u, v) (Schaufler, 1998)**.**

1.5.3.6 Les mosaïques

Les mosaïques concentriques en composant des petits fragments d'images acquises par des caméras le long de leurs chemins sur des cercles concentriques, pour le rendu de la scène à partir d'une nouvelle position de vision. Ainsi, un panorama cylindrique est équivalent à une mosaïque pour laquelle l'axe de rotation passe par le centre de la caméra de projection. Dans

un ensemble de mosaïques concentriques, toutes les images associées à une colonne donnée sont acquises au même angle. Cette technique n'exige pas un grand espace de stockage, mais elle manque de parallaxe verticale accompagnée par des distorsions verticales dans les images rendues. (HEMIDI, 2005)

Figure 14: Fragments d'images acquises par des caméras (Schaufler, 1998).

1.6 La modélisation de la surface

La surface utilisée comme support de notre texture volumique (l'ensemble des texels) est décrite par des polygones quelconques (en se basant sur un maillage surfacique de la surface à habiller). Pour la souplesse et la généralité de notre modèle, il faut que le nombre et l'orientation des texels que l'on plaque sur la surface ne dépendent pas du maillage. Pour respecter ceci on va introduire la notion de boîte.

1.6.1 Notion de boîte

Pour chaque polygone décrivant la surface, on appelle boîte la portion de couche volumique qui lui correspond. Dans le cas du modèle de Kajiya et Kay où la surface est décrite par des quadrilatères et où chaque texel se plaque exactement sur un quadrilatère (voir figure 15 à gauche) ; la boîte contient en fait tout le texel. Mais dans le cas général un texel est placé sur la surface de manière complément indépendante du maillage de celle-ci (voir figure 15 à droite).

Le texel va se déformer pour être collé exactement à la surface. Donc, chaque texel sera en fait composé de plusieurs boîtes, éventuellement même de fractions de boîtes.

Une boîte est donc le volume se trouvant sur chaque polygone de la surface (voir figure 15 à droite). (Meyer, 1998).Les arêtes verticales d'une boîte sont en fait des vecteurs réglables par l'utilisateur, initialement confondues avec les normales à la surface (que l'on obtient en faisant par exemple une moyenne des normales des faces adjacentes au point). Ces arêtes verticales peuvent être perturbées, elles ne sont pas contraintes à rester les normales à la surface. L'utilisateur définit une épaisseur qui correspond à la longueur de ces normales et donc aussi à la distance entre la base et le sommet de la boîte. (Meyer, 1998)

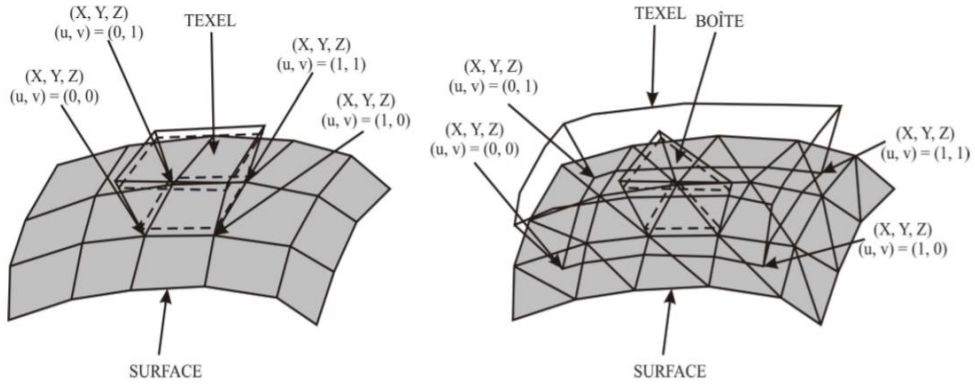


Figure 15 : Plaquage d'un texel sur une surface. (HEMIDI, 2005)

Une boîte est définie par ses points de la base et du sommet. Il est donc aisé de calculer les coordonnées des différentes tranches par une interpolation linéaire entre un point de la base et un point du sommet (voir figure 16). Les coordonnées des points de ces faces ne sont recalculées que quand la boîte change, c'est-à-dire quand elle se déforme. Ce qui se produit soit quand la surface bouge, soit quand on applique une déformation sur les boîtes (pour simuler du vent par exemple). (Meyer, 1998)

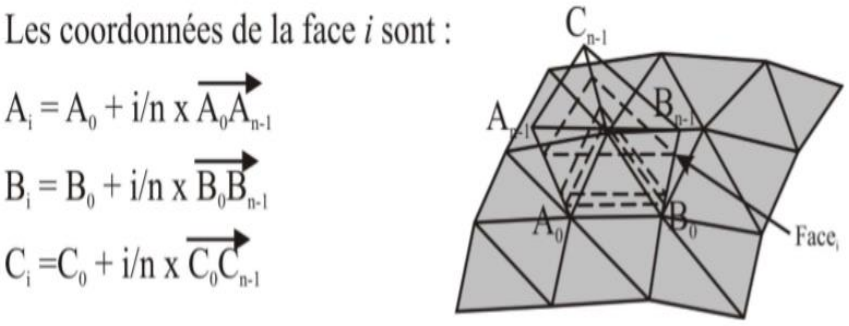


Figure 16: Calcul des coordonnées des faces texturées d'une boîte Méthode de Meyer pour une nouvelle représentation du volume de référence. (HEMIDI, 2005)

Le nouveau volume de référence peut être vu comme un ensemble de N images 2D de taille X×Y au format RGBA (les trois composantes de couleur Rouge, Vert, Bleu et la transparence Alpha). On peut aussi le voir comme un volume de N×X×Y voxels ayant chacun quatre composantes (les trois de couleurs et la transparence) .Ce volume de référence ne contient plus d'informations sur la réflectance puisque la couleur pré calculée est stockée à la place, ce qui implique que le rendu des ombres et de l'éclairage est fixé lors de sa construction. C'est une limitation importante, mais c'est le prix à payer pour accélérer le temps de rendu. A noter que le coefficient alpha pour la transparence est indispensable sinon la face du dessus cacherait une partie des faces d'en dessous. Comme on le verra un peu plus loin, il est stratégique que toutes les données tiennent dans la mémoire texture, sinon les performances se dégradent fortement. Pour le moment, on vient de présenter l'information essentielle que l'on stocke dans ce nouveau volume de référence mais, on verra ensuite que d'autres informations sont nécessaires. (Meyer, 1998)

1.6.2 La génération du volume de référence

Il existe plusieurs méthodes de création de volume de référence Meyer (Meyer, 1998). S'intéresse plus particulièrement à deux méthodes pour générer des volumes de références :

- La conversion d'objets polygonaux vers la représentation en couche ; Comme la nouvelle représentation du volume de référence est très différente de celle utilisée avec l'algorithme de lancer de rayon.
- L'utilisation des textures de Perlin (Perlin, 2014). vont nous servir à créer des texels qui ont une forme très complexe, tordue et détaillée mais dont la forme globale n'est pas très bien connue à priori. Cette texture de Perlin 2D est utilisée comme champ de hauteur. Le niveau de gris de chaque pixel va nous donner la hauteur de la colonne dans le texel. Au préalable, il faut d'abord rendre continu les bords (voir la figure 17) car la forte pente présente dans ces données génère des contours incomplets. Pour cela on parcourt tous les points du texel et, on complète la colonne en dessous de lui jusqu'au voisin le plus bas en interpolant la couleur. D'autre part, on peut générer un deuxième modèle de volume de référence par une coloration de la colonne en dessous jusqu'à la base de la texture.

La représentation en couche introduite ici conduit à montrer l'intérieur des objets sous certain point de vue. Pour pallier à cet inconvénient on décrit donc un algorithme simple de remplissage. (Meyer, 1998)

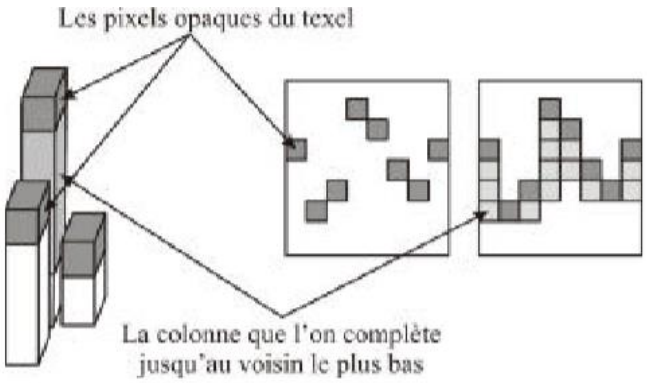


Figure 17: Algorithme de remplissage de la colonne. Gauche : vue en 3D. Droite : Vue en coupe verticale. (HEMIDI, 2005)

1.7 Le rendu du nouveau modèle

1.7.1 Rendu de toute la surface texturée.

Nous disposons donc d'une peau volumique, découpée en un ensemble de boîtes correspondant aux facettes recouvrant la surface. Nous allons voir dans cette partie deux algorithmes qui sont envisageables pour effectuer le rendu d'une telle scène : Le rendu par boîte et le rendu par tranches identiques (HEMIDI, 2005).

1.7.2 Rendu par boîte

Une première méthode de rendu consiste à prendre chaque face de la surface et à afficher la boîte se trouvant au-dessus. L'inconvénient de cette méthode est d'obliger le moteur graphique à changer de contexte texture pour chaque polygone tracé.

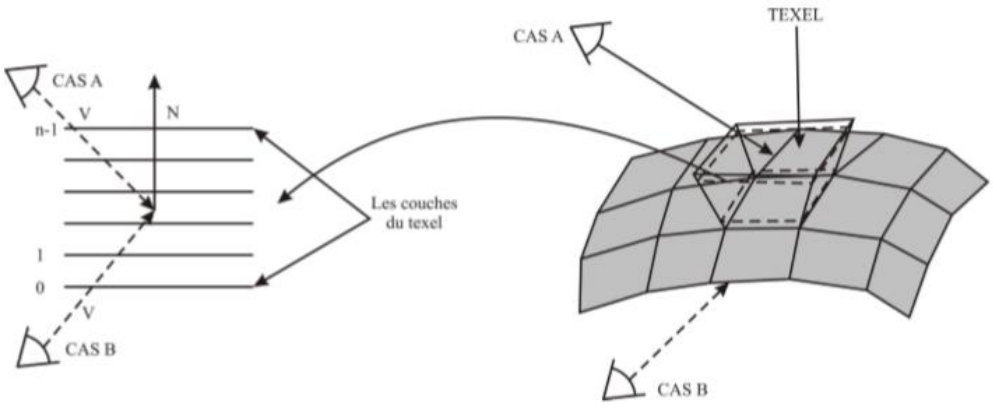


Figure 18: Ordre d'affichage des faces : Cas A de 0 à n-1. Cas B de n-1 à 0 (HEMIDI, 2005)**.**

- L'algorithme utilisé est le suivant :

```
Pour i = toutes les boîtes faire
   Pour n = toutes les tranches de la boîte faire
      Charger la texture n
      Afficher la tranche n de la boîte i
   Fin pour
Fin pour
```

Si l'ensemble des textures représentant le volume ne tient pas dans la mémoire interne réservée aux textures le système doit constamment recharger des textures de la mémoire principale vers cette mémoire texture (la mémoire principale sert de zone de swap). Ceci est très coûteux et les performances s'écroulent. Ce cas se présentera si on implémente cet algorithme sur une "petite machine" où la mémoire texture est limitée. Ce cas où la mémoire texture n'est pas suffisante est pénalisant pour les performances mais même si toutes les textures tiennent en mémoire texture le changement de contexte reste une opération qui a un coût qu'il faut limiter au maximum Source spécifiée non valide. Comme le chargement d'une texture (couche) dans la mémoire associée correspond à un changement de contexte (l'utilisation de la fonction *glBindTexture* pour attribuer un identificateur à la texture et *glTexImage2D* pour charger la texture), nous aurions un changement de contexte $I \times N$ fois, où I est le nombre des boîtes sur la surface et N est le nombre de couches dans une boîte. (HEMIDI, 2005)

1.7.3 Rendu par tranche identique

Même si l'ensemble des textures tient dans la mémoire texture, le fait de changer de contexte texture prend du temps. Ce qui a conduit à présenter un deuxième algorithme qui réduit au maximum les changements de contexte.

Cette deuxième approche consiste à afficher toutes les faces en utilisant la même texture dans une même étape. Comme le volume de référence se reproduit sur toute la surface on affiche la tranche i de toutes les boîtes puis on passe à la tranche suivante (HEMIDI, 2005) .L'algorithme utilisé est le suivant :

```
Pour n = toutes les tranches du texel faire
   Charger la texture n
   Pour i = toutes les boîtes faire
      Afficher la tranche n du texel i
   Fin pour
Fin pour
```

Cette deuxième approche est plus rapide que la première. En effet, avec la première méthode on va changer de contexte de texture Nb fois plus que dans la deuxième où Nb est le nombre de boîte (c'est-à-dire le nombre de polygones de la surface) ce qui fait beaucoup (HEMIDI, 2005).

1.8 Conclusion

Dans ce chapitre, nous avons mis en évidence les notions fondamentales à la compréhension de la complexité de rendu volumique, cette étude consiste à explorer les méthodes de rendu volumique en générale et les textures volumique en Particulier. Ensuite nous avons présenté dans le chapitre suivant la partie de conception Bien détaillé.

Chapitre 02

Conception

2.1 Introduction

Avant de commencer à coder la partie applicative, nous nous intéressons à la phase de spécification pour bien définir, clarifier les grandes fonctionnalités de notre application. Ce chapitre consiste à donner une définition précise des besoins fonctionnels et non fonctionnels ainsi que les objectifs visés.

2.2 Motivation

Malgré toutes les avancés que les textures volumiques ont apportées, et malgré la monté en puissance des machines de calcul spécialisées telles que les stations graphiques, le problème du temps pour avoir un résultat final qui reste relativement élevé. De plus, l'implantation de ce modèle sur une machine individuelle pose, en plus, le problème de l'espace mémoire utilisé pour stocker un volume de référence à haute résolution. La solution apportée à ce problème est l'utilisation d'une représentation alternative du volume de référence, la texture volumique à base de couches d'images, qui permet de profiter des performances du matériel graphique. La méthode a était inspirée, par Meyer (Meyer, 1998), à partir des travaux de Lacroute et Levoy (Levoy, 1994), pour la résolution du problème du temps très élevé pour l'obtention d'un rendu volumique classique.

2.3 Objectif

Le but de ce travail est d'accélérer considérablement les calculs de rendu, en exploitant la représentation à base d'image des données volumiques médicales. Sont stockées dans une Raw data. L'algorithme à base de textures 3D est avantageux par rapport à d'autres techniques de visualisation de volume 3D en raison de sa qualité d'image élevée, et le gain de temps du rendu graphique, sont primordiaux afin que le diagnostic du médecin devienne précis et immédiat.

2.4 Conception globale

L'architecture proposée pour la réalisation de notre application se schématise comme suit :

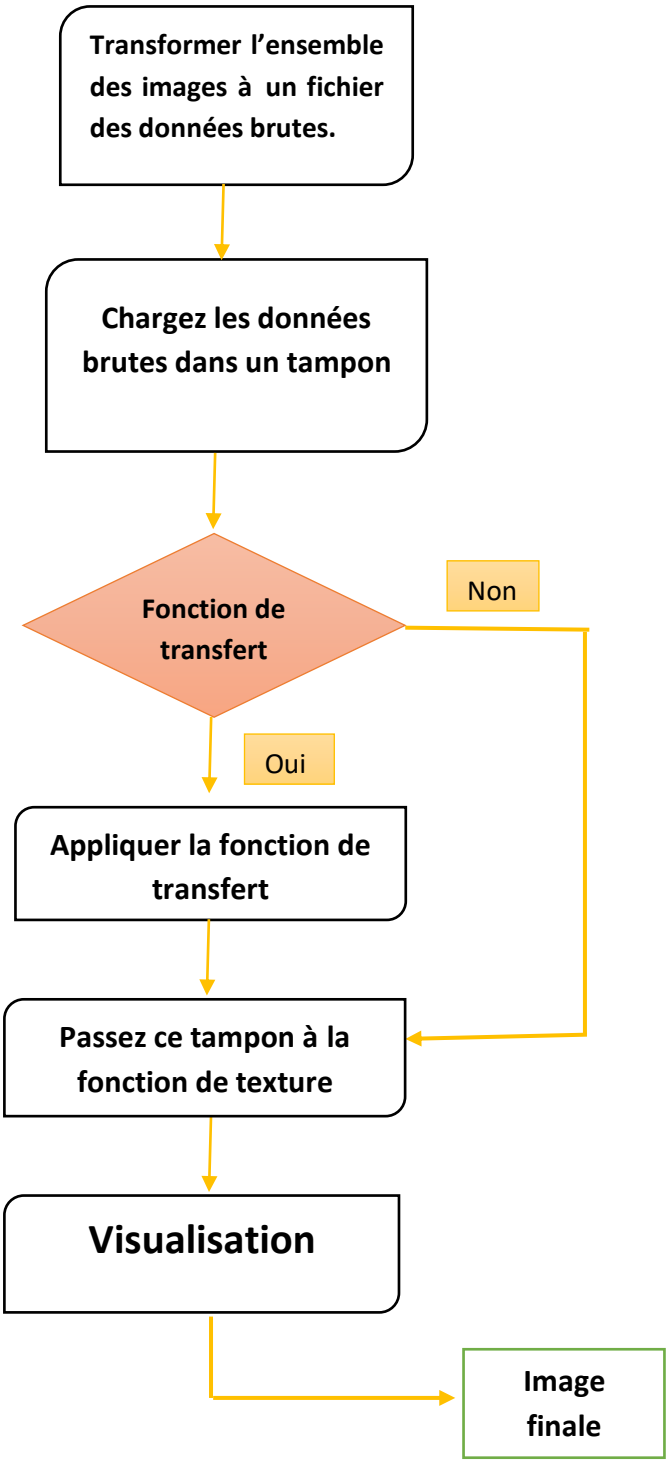


Figure 19: Architecture générale.

2.4.1 Les images de Fichier des données brutes

Les images qui nous avons utilisé pour construire un fichier des données brutes sont des images de type (TIF), qui permet de garder une qualité optimale de la taille (256x256) 8bit per pixel, c'est-à-dire en niveaux de gris ,tel que l'image composé d'une seul couche , les valeurs de pixel prendre soit la valeur 0 est le noir soit la valeur 255 est le blanc ,ne contient pas d'entête alors les image enregistré dans le fichier sous forme d'une suite de bits . Par commodité, on interprète cette suite de bits comme un nombre binaire.

2.4.2 Le tampon utilisé pour stocker les donnée brutes

La mémoire tampon (buffer en anglais) sert à stocker temporairement des données dans la mémoire vive ou dans le disque dur d'un ordinateur. C'est en quelque sorte la "salle d'attente" des données et de toutes les informations qui transitent au sein d'un ordinateur moderne. Sans la mémoire tampon et sans les tampons, les ordinateurs fonctionneraient beaucoup moins efficacement et les temps d'attente seraient très longs. Le concept de la mémoire tampon a été mis au point pour éviter l'encombrement des données d'un port entrant à un port de transfert sortant et dans notre application on créant tampon de la taille (Hauteur x largeur x nombre d'images) et nous l'avons mis en les données de fichier brutes .

2.4.3 Fonction de transfert

L'importance de la fonction de transfert en générale est de découvrir et de mettre en évidence des structures et des phénomènes intéressants incorporés, et dans la médecine particulièrement pour voire l'anatomie exacte des organes du corps et faciliter le diagnostic médical.

Dans le schéma général (Figure 19) ; le module de la fonction de transfert est la configuration ou bien l'association d'une couleur et d'une opacité à chaque densité, ce qu'est un vrai challenge, il est pratiquement très difficile d'assigner les bonnes valeurs d'opacité et de couleur , pour représenter une densité donnant la variété des valeurs de densités équivalentes aux parties du corps humain (os, tissus mous, sang, poumons etc.) , des valeurs erronées peuvent rendre des parties du corps humain , qui doit être opaques (comme les os) transparentes ou l'inverse , la meilleure approche pour résoudre ce problème est d'essai et d'erreur jusqu'a la réalisation des résultats satisfaisants,

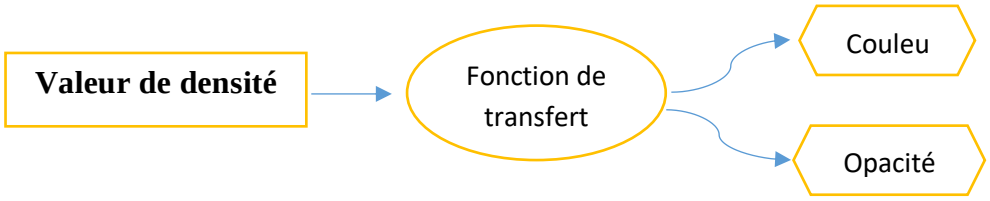


Figure 20: **Schéma de fonction de transfert.**

2.4.4 Rendu

C'est le module rend la visualisation possible, d'abord, il applique la technique à base de texture 3D. L'algorithme de rendu volumique direct appartient à la catégorie des méthodes ordre image (backwards), il s'agit de projeter chaque texel sur l'écran, il faut citer que la méthode de projection utilisée dans notre application est la projection parallèle et le point de vue initial est le centre de l´écran.

2.5 Conception détaillée

Ce diagramme résume les étapes les plus importantes de la création d'une texture 3D

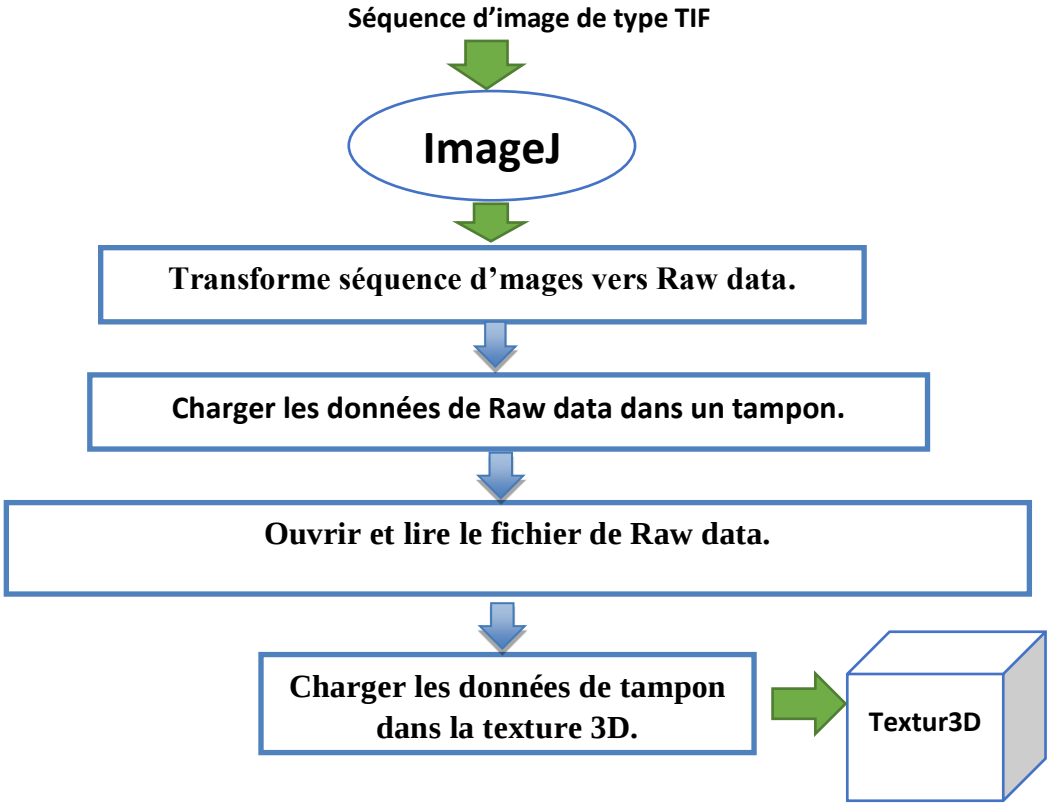


Figure 21: **Schéma explique les étapes de création de la texture 3D.**

2.5.1 Transparence et valeurs alpha

La transparence et son complément, l'opacité, sont souvent appelés alpha dans l'infographie. Par exemple, un polygone opaque à 50% aura une valeur alpha de 0,5 sur une échelle de zéro à un. Une valeur alpha de un représente un objet opaque et zéro représente un objet complètement transparent.

Figure 22: La valeur d'alpha à gauche inférieure à 0.5 au milieu égale 0.5 adroite supérieure à 0.5.

2.5.2 Les données brutes (Raw data)

Les données brutes ne sont rien d'autre que des frames 2D (voire la figure 22) continues. Voici un aperçu des tranches obtenues en ouvrant les données brutes jointes. Ces frames 2D proviendront de différentes positions de l'axe Z. Dans notre application on utilise séquence d'images de l'extension (.TIF), puis on le transformer à une Raw data avec l'éditeur d'image imageJ.

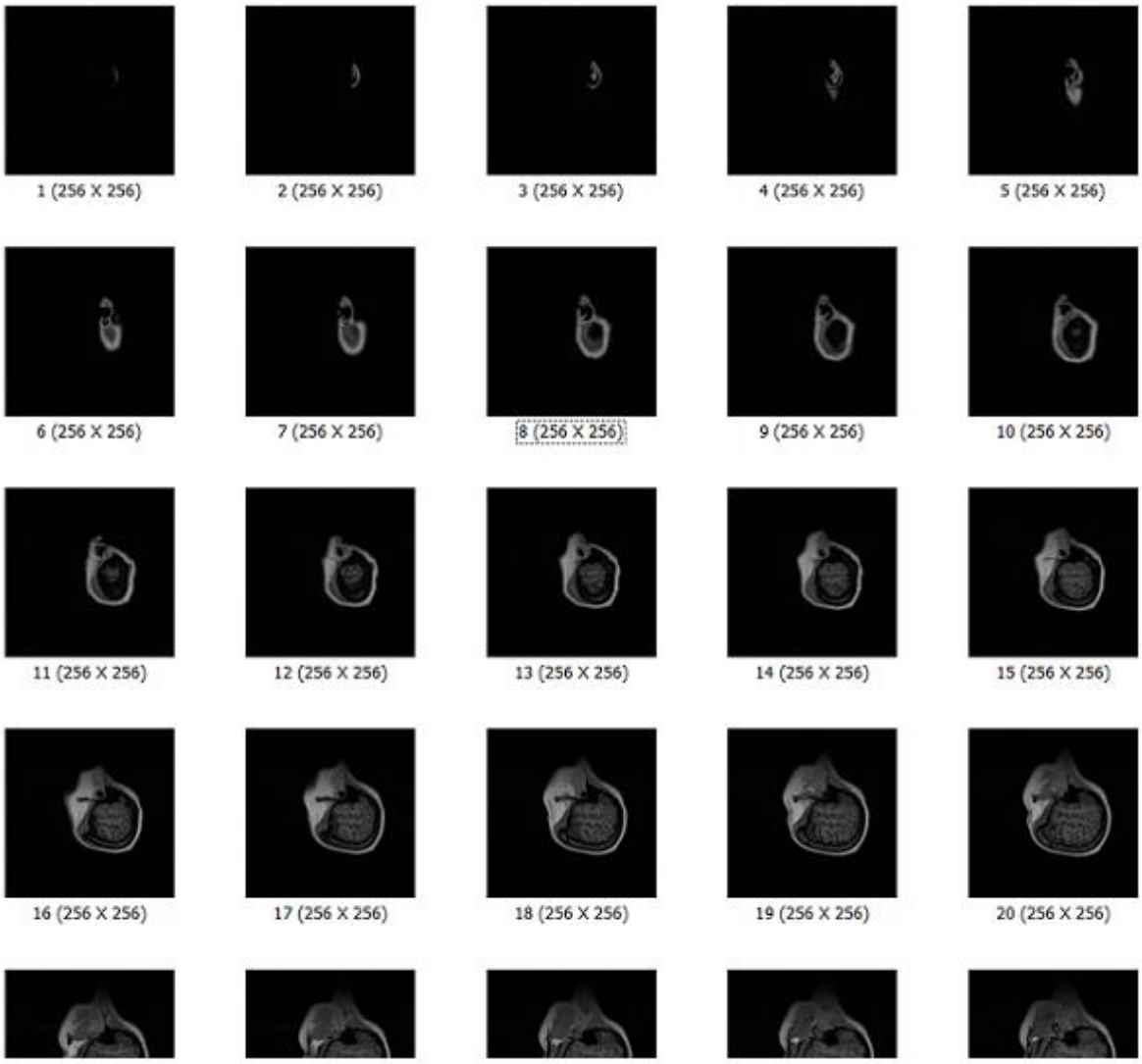


Figure 23: Séquence d'images d'une tète humain avec ces tailles et ces positions dans l'axe Z.

2.6 Les étapes de construction de l'application

2.6.1 Etape 1

La lecture des données du fichier Raw data et de leur conversion en texture 3D.L'objectif principal de l'application que nous nous implémentons une technique qui permet d'ajouter le rapidité et la bonne qualité à une rendu volumique par l'intégration des texture volumique, tel que les principaux objectifs à atteindre peuvent être résumé ainsi :

Le format de fichier d'image brute (Raw data) est le format de fichier d'image en échelle de gris. Ne contient aucun entête, mais contient directement les valeurs de pixel. Dans notre cas on utilise éditeur d'image (*imageJ*) pour transformer les images 2D à un fichier Raw data.

Donc on saisir la largeur et la hauteur de l'image et suivre les points suivantes pour recevoir une texture 3D : charger les données de Raw data dans un tampon de la taille longueur de l'mage multiplier par la largeurs de l'image et le nombre d'images utilisée (longueur x largeurs x nombre des images) puis on générer la texture 3D, après on lier une texture nommée à une cible de texturation puis on charge les données de tampon dont le texture 3D.

2.6.1.1 Conversion des images 2D à un fichier Raw data

Pour utiliser l'éditeur d'image imageJ on a une série d'images dans un dossier sous forme de pile (Rang de 1 à n selon l'axe Z). Les images doivent toutes être de la même taille et du même type, et le nombre de bit par pixel doit être connu. Ils peuvent être au format TIF, JPEG, DICOM, BMP, GIF, FITS… dans notre cas on utilisant le format d'image TIF au niveau de gris (8 bit per pixel), les images au niveau de gris entrelacées ont des pixels stockés de manière contiguë (256 valeur possible) dans un seul plan d'image. L'écart (Gap) entre les images est le nombre d'octets entre la fin d'une image et le début de la suivante. Définissez cette valeur sur largeur x hauteur x octets par pixel x n pour sauter n images pour chaque image lue.

Figure 24 : Schéma représente la conversion des images 2D à un fichier Raw data.

2.6.1.2 Charger les données de fichier Raw dans un tampon et la conversion en texture3D

Le fichier Raw data vas nous donner des images 2D sous format binaire, nous allons lire le fichier on utilisant la fonction Read. Nous lui envoyons la taille de tampon(hauteur x largeur x nombre d'image), puis la fonction Read va créer un tampon de type Char et stocker les valeurs de toutes les images dans l'ordre fournis ou spécifiés dans le fichier Raw data, tel que chaque cellule du tampon représente une valeur de profondeur sur l'axe Z (depth),puis nous créant une autre tampon de la taille (hauteur x largeur x nombre d'image x 4), donc nous obtenons un

tampon chaque quatre cellule représente les canaux R rouge, G vert , B bleu ,A transparence . Nous attribuons donc à chaque cellule du tampon une valeur qui appartient à l'intervalle [0,255], ainsi, les valeurs de tampon deviendra des valeurs des couleurs.

Après nous envoyons le tampon à la fonction 'glTexImage3D' avec la largeur et la langueur et la profondeur de l'image 3D , les positions des voxels (X,Y)vont donc être obtenue en parcourant la texture 3D de 0 au maximum de la largeur et de 0 au maximum de la langueur et de 0 au maximum de la profondeur ,sachant que le tampon de couleur que nous avons envoyé est ordonné et va nous donner les positions correct pour chaque voxel pour construire le volume.

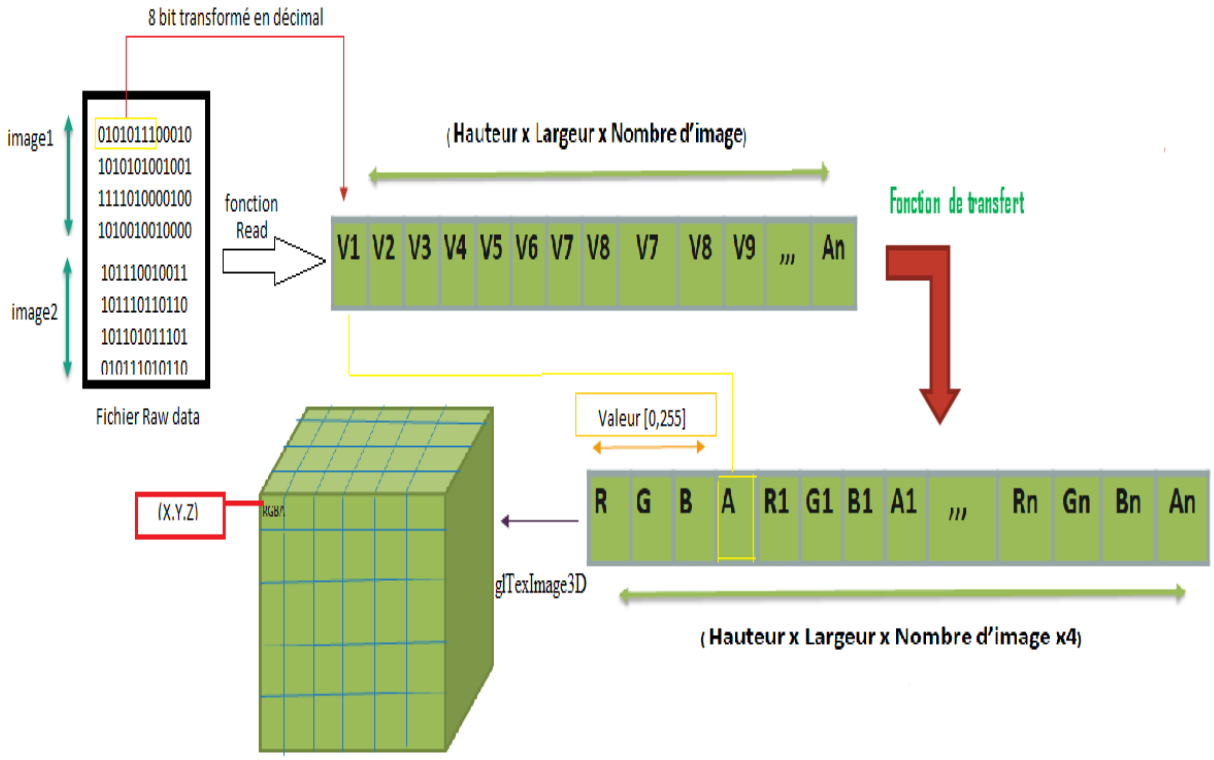


Figure 25 : Schéma montrant les étapes de lecture et de la construction des données.

L'algorithme suivant représente les étapes de la création d'une texture 3D

Algorithme 1 : Classe processeur-de-données.

Entré
Height: entiers // le hauteur de l'image.
Width : entier // la largeur de l'image.
Count : entier // nombre des images utilisé.
Texture : texture 3D // pour la création de texture.
Buffer : caractères // tampon de type caractères pour mettre les données de fichier bruts.
File : fichier //un fichier de données brutes.
Variable
X : entier
Début
X ← Width x Width x Count //représente la talle de tampon
Open (File) ; // ouvrir le fichier de Raw data.
Read (File) ; // lire le fichier Raw data.
Fill (Buffer[X], File) ; //met le fichier dans un tampon.
Transferfunction (Buffer[X]) ; //fonction de transfert
Generate_textur(Texture) ; //générer un texture.
Create_textur(Texture) ; //création de texture.
Loading_textur(Texture,Buffer[x]) ;//charger les données de tampon dans la texture 3D.
Activated_texture(Texture) ; //activer les texture.
Fin
Sorté Texture ; // texture 3D.

2.6.2 Etape 2

Placage de texture 3D sur le Quad et rendu de volume 3D.Dans cette étape on spécifie la coordonnée de texture et le sommet correspondant du Quad, pour appliquer cela, nous devons savoir la projection ortho et la texture 3D normalisé.

2.6.2.1 La projection orthographique

La projection orthographique (parfois également appelée projection oblique) est plus simple que les autres types de projections et en apprendre d'avantage est un bon moyen d'appréhender le fonctionnement de la matrice de projection en perspective. Le but de cette matrice de projection orthographique est en fait de remapper toutes les coordonnées

contenues dans une certaine boîte englobante (cube unitaire) dans l'espace 3D dans le volume de visualisation canonique. La matrice orthographique est de la remapper en un volume de vue canonique.

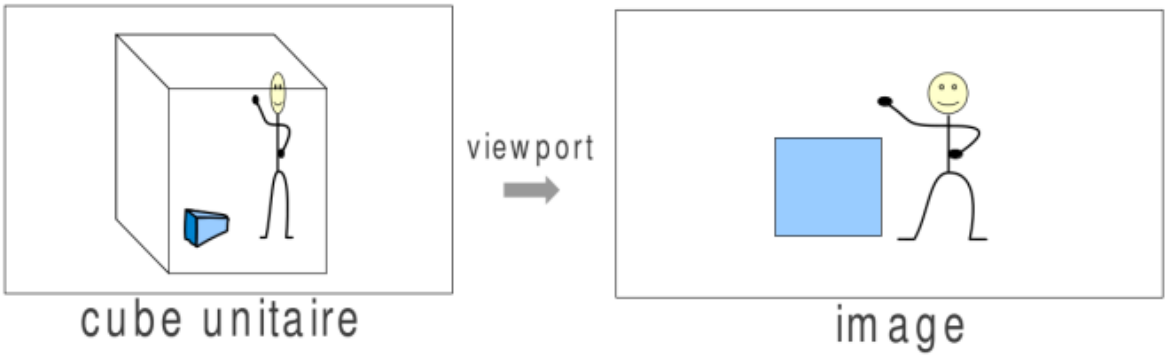


Figure 26: **Schéma montre la suppression du l'information de profondeur**.

Dans notre application nous utilisons la projection orthogonale avec les valeurs suivantes :

Gauche = -1 / Droite = +1.

Haut = +1 / Bas = -1.

Près = -1 / Loin = +1.

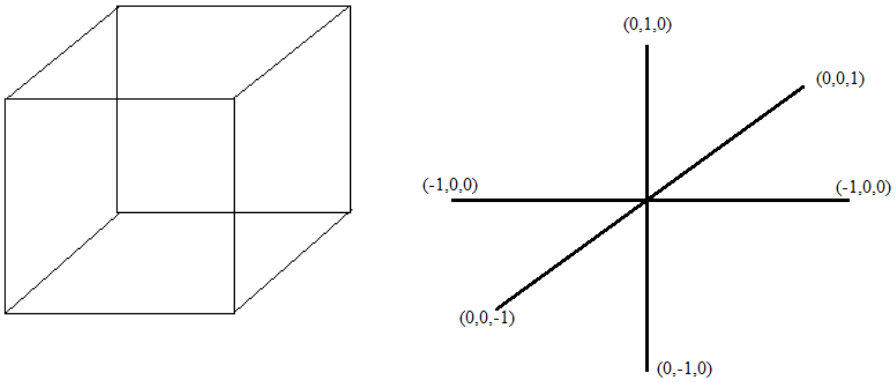


Figure 27: **Les valeurs d'orthogonale qui nous avons utilisé.**

2.6.2.2 L'utilisation de texture

Les textures 3D ont l'avantage de l'axe z. Tout comme la largeur et la hauteur sont normalisées à 1 dans une texture 2D, le nombre de tranches ou l'axe z est également normalisé à 1 dans une texture 3D, tel que on devisé les coordonnées de texture (X, Y, Z) sur la taille

de texture (Largeur, longueur, hauteur) respectivement .Ce texture représente les coordonnées de notre texture après la normalisation.

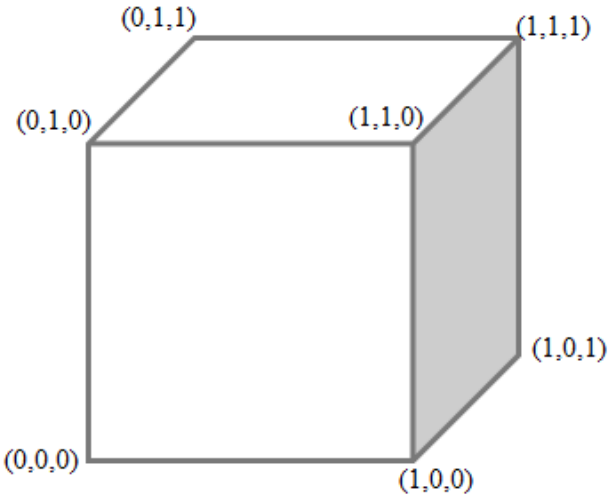


Figure 28: **Texture 3D normalisée.**

2.6.2.3 La plaquage de texture

Il suffit d'interpoler chaque sommet de quad (x,y,z) avec chaque coordonnée de texture (u,v,n) la figure suivante représente le plaquage de texture 3D :

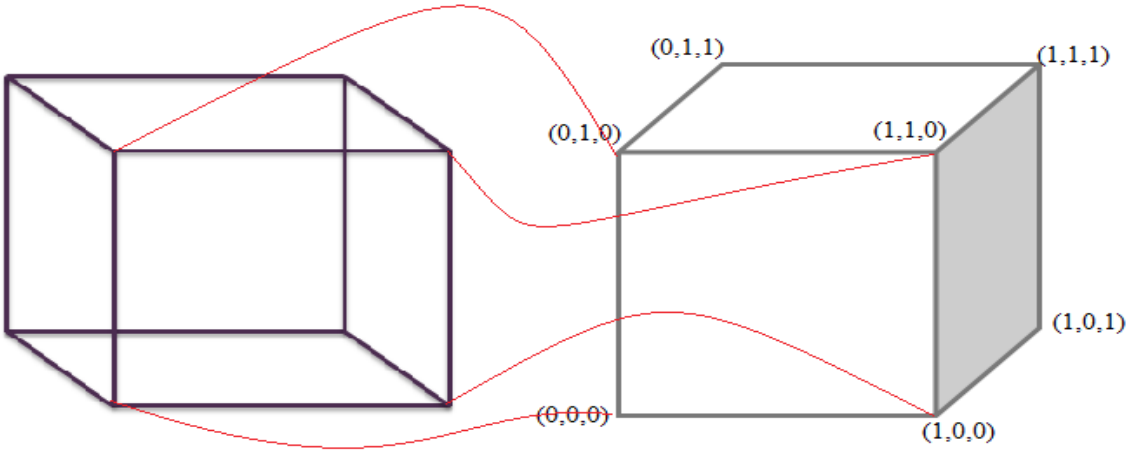


Figure 29: **Schéma montre la placage de texture sur le Quad.**

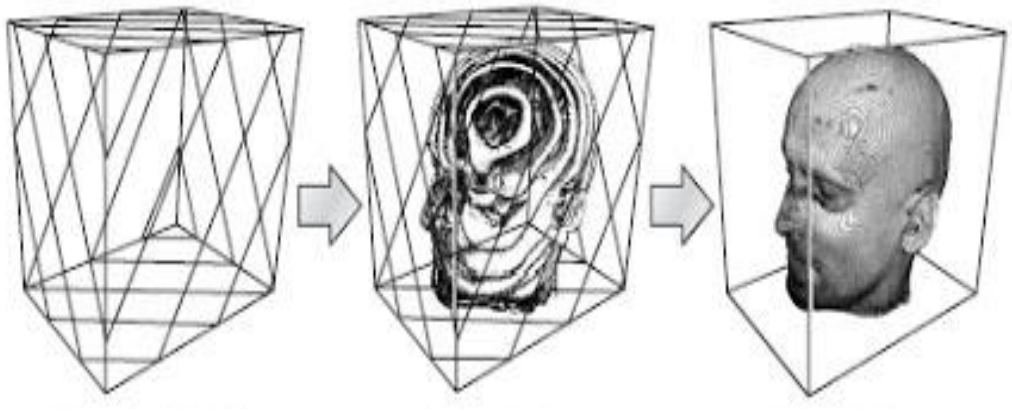


Figure 30 : Le placage de texture sur le Quad et le rendu de l'image finale.

2.6.2.4 La translation de volume

On fait la translation de volume au centre de la scène avec a valeur 0.5 la figure suivante montre l'opération :

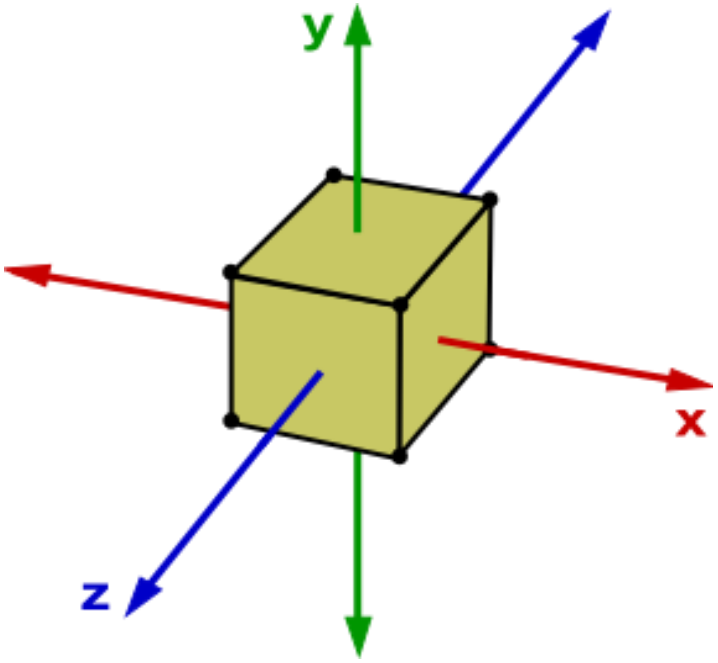


Figure 31: Le volume au milieu de repère global de la scène (Tutoriel : Développez vos applications 3D avec OpenGL 3.3, 2016)**.**

Cet algorithme résume les étapes les plus importantes pour le placage de texture 3D :

Algorithme 2 : Classe placage de texture 3D et rendu.

Entré
Texture : Matrice2D[m] [n] ; //matrice de texture 2D
Quad : Matrice2D[m] [n] ; //matrice de quad 2D
//m est le nombre de ligne et n nombre de colonne
Début
Pour tous les coordonnées de texture et les sommets de quad faire
Interpolation (Quad[x, y, z], Texture [u, v, n]) ; //plaqué chaque coordonnée de texture sur le sommet de quad approprié.
//(x,yz) reprisent les valeurs de sommets de la matrice Quad.
//(u,v,n) reprisent les valeurs de coordonnées de la matrice Texture.
Fin
Translate (volume3D) ; //placer le volume 3D au centre de scène.
Pour i dans l'intervalle [-1 ,1] pas de 0.01 faire
Scale(Quad,Texture) ; // changement d'échelle pour le faire varier de (-1) à (+1).
Fin
Fin
Sorté volume3D ;

2.7 Conclusion

Au cours de ce chapitre, nous avons détaillé notre projet, c'est-à-dire les objectifs majeurs à prendre en compte, une analyse profonde de la solution adaptée en précisant les différentes fonctionnalités des classes. Dans le chapitre suivant, nous allons entamer la phase de l'implémentation.

Chapitre 03

Mise en œuvre et résultats

3.1 Introduction

Ce chapitre représente la dernière partie de ce mémoire .celui-ci sera consacrée à définir le processus de la réalisation de notre application, nous commencerons par décrire les outils (matériels et logiciels) ainsi que les plateformes que nous avons utilisées, puis le diagramme résumant notre travail et les techniques que nous avons intégré. Nous concluions ce chapitre par la présentation des résultats de notre application.

3.2 Environnement de l'application

La mise en œuvre de n'importe quelle application requiert un environnement de développement matériel et logiciel et le nôtre, définie de la manière suivante :

3.2.1 Environnement de développement matériel

Pour l'implémentation de notre application nous avons utilisé une machine ACER PC avec les caractéristiques suivantes :

- Processeur : Intel(R), i3-5500U CPU @ 2.40GHz.
- RAM : 4,00 GB.
- Carte graphique : Intel HD Graphics 4400.
- Disque dur : 500 GB.

3.2.2 Environnement de développement logiciel

Notre application a étais implémentée sous un système d'exploitation Windows 10 32-bits, plusieurs outils de programmation ont étais utilisés, nous avons choisi *Microsoft Visual Studio 2019* comme un IDE (Environnement de développement Intégré) et le **C++** comme un langage de programmation pour beneficier de sa paradigme orientée-objet.

3.3 Description du système

3.3.1 OpenGL

Est une librairie exploitant la carte graphique d'un ordinateur permettant ainsi aux développeurs de programmer des applications 2D et 3D. Cela permet entre autres de développer des jeux-vidéo.

L'avantage principal d'OpenGL est qu'il est multi-plateforme, c'est-à-dire qu'un programme codé avec OpenGL sera compatible avec Windows, Linux et Mac, sous réserve que la gestion de la fenêtre et des inputs soient également multi-plateforme (comme la SDL). Actuellement,

deux API existent pour exploiter notre matériel graphique : DirectX (uniquement utilisable sous Windows) et OpenGL (multi-plateforme). Nous déduisons que : OpenGL est dans un sens plus intéressant du fait de sa portabilité.

Un des autres avantages d'OpenGL est que son utilisation est totalement gratuite, vous pouvez très bien coder des programmes gratuits voire commerciaux sans rendre de compte à personne. Vous pouvez aussi fournir votre code source pour en faire profiter la communauté

En plus d'être gratuite, cette librairie met à disposition son propre code source. Chacun d'entre nous peut aller voir le code source d'OpenGL librement.

On notant que le C++ est le langage le plus utilisé dans le monde de rendu, c'est la raison pour laquelle que nous allions l'utiliser. (Tutoriel : Développez vos applications 3D avec OpenGL 3.3, 2016).

Les objets OpenGL (OpenGL load Objects)

Les objets OpenGL sont des structures composées d'états et de données et sont responsables de la transmission de données vers et depuis le GPU. Les objets OpenGL doivent d'abord être créés et liés avant de pouvoir être utilisés. Il existe plusieurs types d'objets OpenGL. Par exemple, un objet tampon de sommet peut stocker des sommets d'un caractère. Alors qu'une texture, un autre type d'objet OpenGL, peut stocker des données d'image. (SERRANO, 2015).

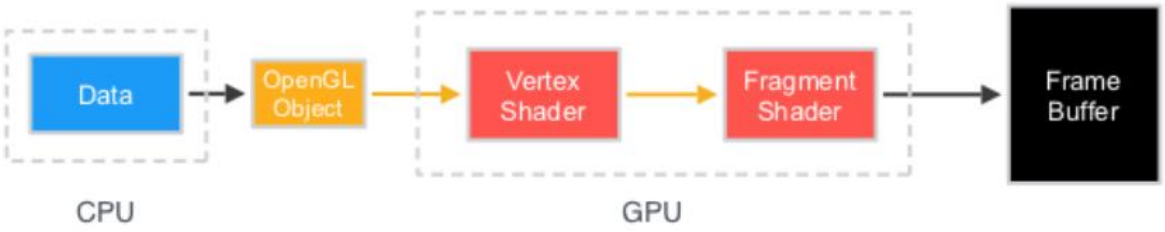


Figure 32: la transmission de données depuis le CPU vers le GPU (SERRANO, 2015)**.**

3.3.2 Fichiers d'entête précompilés (Precompiled Header)

L'entête précompilé est une approche générale utilisée par des nombreux compilateurs pour *réduire le temps de compilation*. La motivation sous-jacente de l'approche est qu'il est courant que les mêmes fichiers d'entête (et souvent volumineux) soient inclus par plusieurs fichiers source. Par conséquent, les temps de compilation peuvent souvent être grandement améliorés en mettant en cache une partie du travail (redondant) effectué par un compilateur pour traiter les entêtes. Les fichiers d'entête précompilés, qui représentent l'une des nombreuses façons de mettre en œuvre cette *optimisation*, sont littéralement des fichiers qui représentent un cache sur disque contenant les informations vitales nécessaires pour réduire une partie du travail important pour traiter un fichier d'entête correspondant. Alors que les détails des entêtes précompilés varient selon les compilateurs, les entêtes précompilés se sont révélés très efficaces pour *accélérer la compilation de programmes* sur des systèmes avec de très grands entêtes système. (Clang Compiler User's Manual) . Donc lorsqu'on crée un nouveau projet dans Visual Studio, un fichier d'entête précompilé nommé pch.h est ajouté au projet. (Dans Visual Studio 2017 et versions antérieures, le fichier s'appelait stdafx.h.) Le but du fichier est d'accélérer le processus de génération. Tous les fichiers d'entête stables, par exemple les entêtes de bibliothèque standard telle que <vector>, doivent être inclus ici. L'entête est précompilé et compilé uniquement lorsque tous les fichiers qu'il inclut sont modifiés. Si on ne modifie que le code source de notre projet, la compilation ignorera la compilation de l'entête précompilé.

Le code précompilé est utile pendant le cycle de développement pour réduire le temps de compilation, surtout si :

- On utilise un grand corps de code qui change rarement.
- Notre programme comprend plusieurs modules et qui utilise tous un ensemble standard de fichiers include et les mêmes options de compilation. Dans ce cas, tous les fichiers d'inclusion peuvent être précompilés dans un entête précompilé.

Figure 33: **Contenu de l'entête précompilé.**

3.4 Description de l'application

3.4.1 Les étapes de réalisation de l'application

L'objectif principal de notre application est de réaliser et implémenter une technique qui permet d'ajouter la rapidité et la bonne qualité à un rendu volumique par l'utilisation de la texture 3D, tel que les principaux objectifs à atteindre peuvent être résumer comme suit :

D'abord le format de fichier d'image brute (Raw data) est un format de fichier d'image au niveau du gris .Il ne contient aucun en-tête, mais contient directement les valeurs de pixel. Donc l'utilisateur doit saisir la largeur et la hauteur de l'image et il faut suivre les étapes suivantes pour recevoir une texture 3D :

- Charger les données de Raw data dans un tampon de la taille (longueur x largeur x nombre des images).

- générer le nom de texture on utilisant la fonction **glGenTextures()**.
- lier une texture nommée à une cible de texturation on utilisant la fonction glBindTexture**()**.
- définir les paramètres d'environnement de texture on utilisant la fonction **glTexEnvi()**.
- Le paramétrage de la méthode de placage de texture se fait grâce à la fonction : void glTexParameteri(Glenum cible ,Glenum,GL valeur)
  - "Cible" définit le type de texture que l'on veut paramétrer GL_TEXTURE_1D, GL_TEXTURE_2D ou GL_TEXTURE_3D.
  - "nomparam" désigne le paramètre que l'on souhaite modifier, soit la méthode de filtrage lors d'un étirement GL_TEXTURE_MAG_FILTER ou d'un rétrécissement GL_TEXTURE_MIN_FILTER.
  - "Valeur" correspond à la nouvelle valeur à affecter au paramètre, soit GL_LINEAR ou GL_NEAREST.

L'étape suivant consiste à définir la texture a utilisée, par exemple une texture 3D comme dans cas que nous avons appliqué :

void **glTexImage3D** (GLenum cible, Glint niveau, Glint formatinterne, GLsizei largeur, GLsizei hauteur, GLint bord, GLenum format, GLenum type, const Glvoid texture) ;

3.4.2 Les classes de l'application

3.4.2.1 Classe processeur-de-données

Cette classe est responsable de lire les données du fichier et les convertir en texture .Pour lire les données on testera si le fichier n'est pas initialisée, ça sera zéro. Aucune vérification n'est nécessaire sinon le fichier ne contient que des données d'image. La dimension des données doit être connue, on créant deux tampons, l'un pour la luminance et l'autre pour la couleur RGBA puis on convertit les données luminance en données RGBA .Cela signifie que nous mettons simplement la même valeur aux canaux R, V, B et A généralement pour les données brutes c'est pour afficher le volume sans appliquer la fonction de transfert mais avec la fonction de transfert nous mettons des défirent valeur entre[0,255] aux canaux R, V, B et A tel que la valeur de densité est située entre une valeur maximale et minimale spécifique pour

chaque couche de volume, la valeur alpha sera construite par une valeur seuil donnée par l'utilisateur .Et puis on convertie les données vers une texture 3D.

```
Class processeur-de-données
{
public:
CRawDataProcessor(void); // constricteur de la classe
    virtual ~CRawDataProcessor(void);// destructeur de la classe

bool ReadFile(LPCTSTR lpDataFile_i, int nWidth_i, int nHeight_i,
int nSlices_i );
//méthode lire le fichier Raw data
void CRawDataProcessor::transferFunction();
//méthode pour appliquer la fonction de transfert
    void CRawDataProcessor::Voxelcolor();
//méthode pour afficher coloration par position de voxels
    void  CRawDataProcessor::Voxelgrey();
//méthode pour afficher le volume sans l'utilisation de fonction de
transfert


} ;
```

3.4.2.2 Classe Renderer-Helper

Cette classe est responsable d'effectuer l'initialisation d'OpenGL et le rendu du volume**.** Pour l'initialisation de l'OpenGL : configuration de la boîte de dialogue pour prendre en charge OpenGL et puis on définit le format de pixel sur la boîte de dialogue actuelle. (Il n'y a rien de spécial que nous faisons ici autre que les étapes habituelles pour initialiser OpenGL. J'appelle la fonction Initialize de Dialog :: OnCreate pour initialiser. La fonction Redimensionner gère la configuration de la projection ortho. Comme dialog : :OnSize sera appelé lors du démarrage et du redimensionnement, OnSize est l'endroit où nous pouvons configurer l'ortho. Render () sera appelé depuis OneraseBackground () pour dessiner la scène. Glew.h et glew32.lib (avec une carte graphique de support) sont nécessaires lorsque nous passons aux textures 3D. En effet, la version OpenGL livrée avec Windows ne prend pas en charge les spécifications supérieures à la version 1.1.).

```
Class Renderer-Helper
{
public:
CRendererHelper(void); // constricteur de la classe
virtual ~CRendererHelper(void);
// destructeur de la classe


void Resize( int nWidth_i, int nHeight_i );

//la creation orthographique

void Render();
// méthode pour changement d'échelle de
conversion pour la faire varier de [1,0].


} ;
```

3.4.2.3 Classe Transformation-Manager

Cette classe est responsable de gérer la transformation et la conservation dans une matrice. Nous avons 256 pixels sur l'axe x, 256 sur l'axe y et nombre d'image utilisé sur l'axe z. lorsque nous utilisons la rotation de texture, nous devons utiliser la propriété de texture GL_CLAMP_TO_BORDER pour obtenir une image correcte. Parce que, lorsque les coordonnées de texture sont pivotées, les coordonnées de celle-ci peuvent même être au-delà de 0-1. En d'autres termes, nous ne dessinons rien dans le plan Y-Z. Ainsi, lorsque le plan X-Y pivote de 90 degrés, nous ne pouvons rien voir car OpenGL ne dessine pas le bord du Quad .De tel sorte OpenGL prend en charge une matrice appelée matrice de texture. Nous pouvons appliquer des transformations dans la matrice de texture, et cela sera affecté lors du mappage de texture.

```
Class Transformation-Manager

{
const double* GetMatrix()
private:

    double mdRotation[16]; //matrice de retation.

public:
void Rotate(float fx_i, float fy_i, float fz_i ); // le
réglage de camera etperspective

void ResetRotation();
// pour la rotation de volume

constricteur() ;

destructeur () ;

} ;
```

3.4.2.5 Classe rendu_volumique_de_dialogue

Dans cette classe on définit les valeurs initiales des variables membres de l'objet de dialogue, généralement dans notre gestionnaire OnInitDialog ou le constructeur de dialogue. Immédiatement avant l'affichage de la boîte de dialogue, le mécanisme DDX (Dialog Data Exchange) de l'infrastructure transfère les valeurs des variables membres aux contrôles de la boîte de dialogue, où elles apparaissent lorsque la boîte de dialogue elle-même apparaît en réponse à DoModal ou Create.

```
Classe  rendu_volumique_de_dialogue

{
private:

    CRendererHelper m_Renderer;
    HDC mhContext;
    CPoint mRotReference;
    CRawDataProcessor* m_pRawDataProc;
    CTranformationMgr* m_pTransform;

public:
    int OnCreate(LPCREATESTRUCT lpCreateStruct);//création fentre
de dialoge pour choisi le fichier raw data .
    void OnMouseMove(UINT nFlags, CPoint point);//le contrele de
volume par le mouse
};
constricteur() ;
destructeur () ;
```

3.5 Problème et solution de l'application

Quand les rotations sont appliquées pour bien voir notre volume on trouve quelques soucis résumé dans cette section avec des solutions :

L'écran devient vide lorsqu'il atteint 90 degrés

- Lorsqu'elle pivote pour atteindre 90 degrés, l'image perd progressivement des détails et devient vierge à 90 degrés. Idem pour 270 degrés.

Pourquoi cela arrive-t-il ?

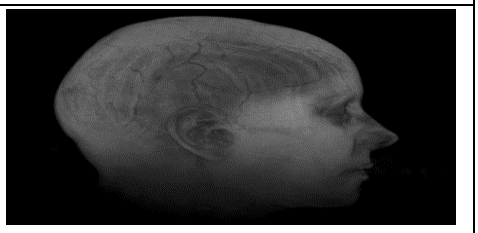
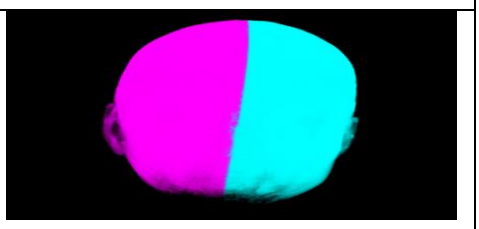
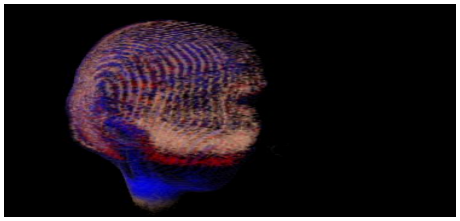
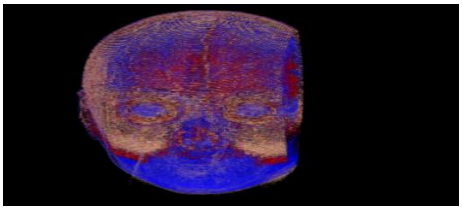
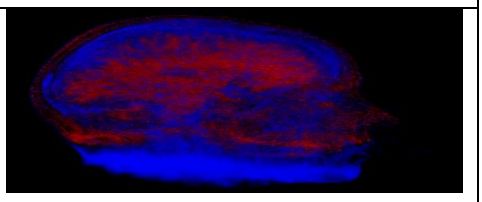
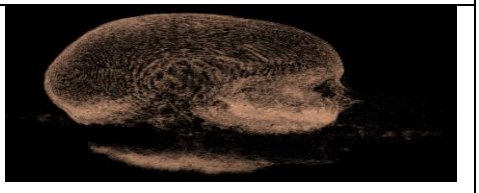
- Au fur et à mesure que nous faisons tourner les quads, nous n'obtenons pas suffisamment d'échantillons pour les mélanger chaque fois qu'ils sont tournés. Lorsqu'il atteint 90 degrés, tous les quadriceps que nous dessinons deviennent parallèles à nos yeux. En d'autres termes, nous ne dessinons rien dans le plan Y-Z. Ainsi, lorsque le plan X-Y pivote de 90 degrés, nous ne pouvons rien voir car OpenGL ne dessine pas le bord du quad. Pour obtenir l'image réelle à une rotation de 180 degrés, nous devons d'abord dessiner la texture à l'autre extrémité.

Solutions

- OpenGL prend en charge une matrice appelée matrice de texture. Nous pouvons appliquer des transformations dans la matrice de texture, et cela sera affecté lors du mappage de texture.
- Lorsque nous utilisons la rotation de texture, nous devons utiliser la propriété de texture GL_CLAMP_TO_BORDER pour obtenir une image correcte. Parce que lorsque les coordonnées de texture sont pivotées, les coordonnées de texture peuvent même être au-delà de 0-1. Nous ne voulons pas émettre de données lorsque la coordonnée est au-delà de cela.

3.6 Résultats

3.6.1 Guide pour les touches clavier

Bouton de clavier	description de la tâche	L'affichage
Bouton : [W]	Afficher le volume sans la fonction de transfert.	
Bouton : [X]	Afficher le volume en couleur en fonction de la position des voxels.	
Bouton : [T]	Suite à chaque clique on réduit une tranche du volume par la valeur de [100000voxels] de droite à gauche.	
Bouton : [R]	Suite à chaque clique on ajoute une tranche du volume par la valeur de [100000voxels] de gauche à droite.	
Bouton : [E]	Dissimuler la couche de la peau.	
Bouton : [S]	Réafficher la couche de la peau.	

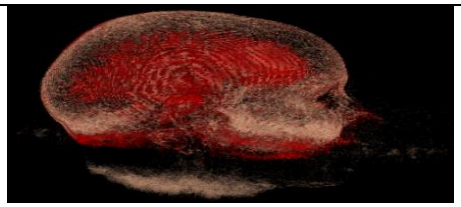
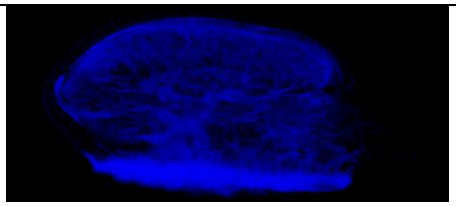
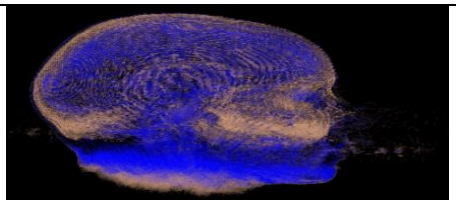
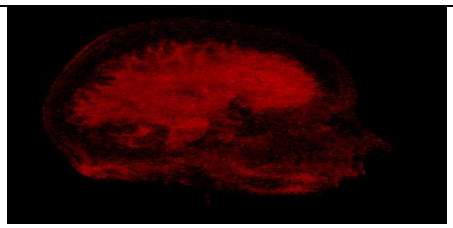
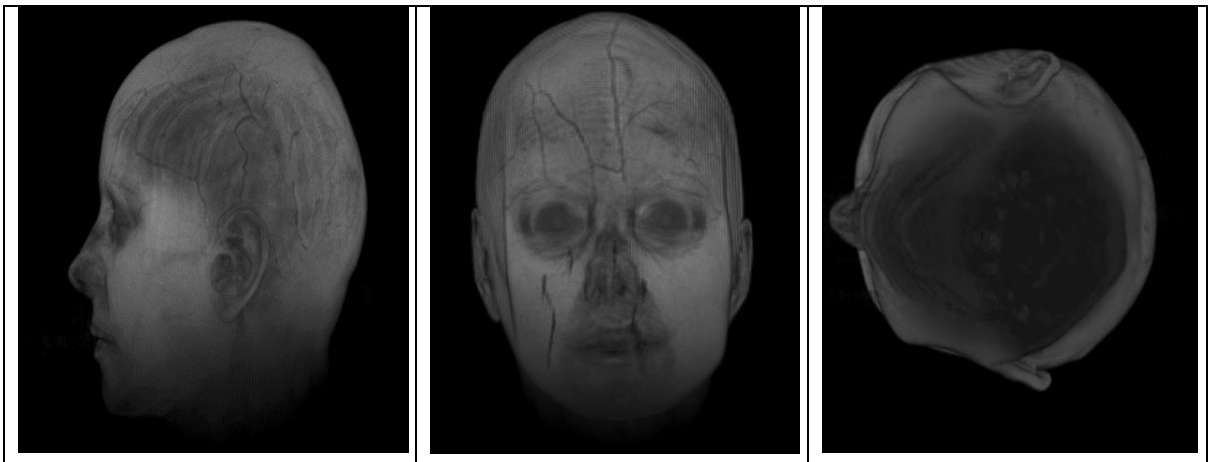
Bouton : [D]	Dissimuler la couche des méninges	
Bouton : [Z]	Réafficher la couche des méninges	
Bouton : [C]	Dissimuler le cerveau	
Bouton : [Q]	Réafficher le cerveau	

Tableau 3: Raccourcis clavier pour les opérations disponible dans l'application.

3.6.2 Des résultats sans appliquer une fonction de transfert

Les résultats suivant représentent une tète humain après l'intégration de la fonction alpha.



vue longitudinale	vue avant	vue transversale

Figure 34: Représentation des vues d'une tête humaine par scanner sans l'application de fonction de transfert.

Dans les résultats suivant en diminuant la valeur alpha .Et on note que les zones réfléchissantes (opaques) tel que la peau sont bien afficher par contre les zones non-opaques telles que le cartilage du nez et des yeux ne sont pas affichées (transparente).

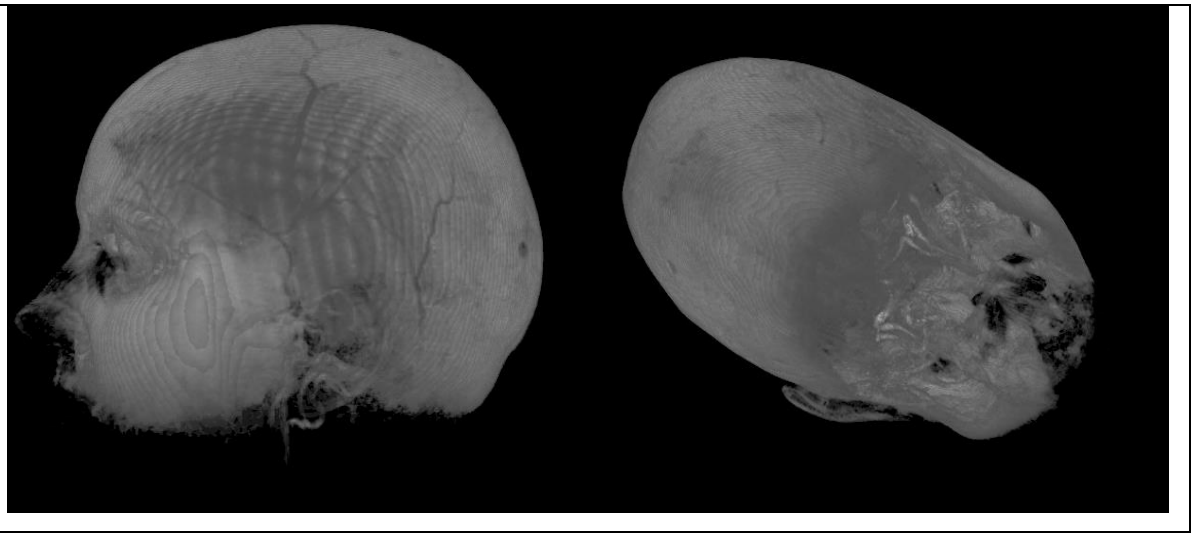


Figure 35: Représentation des vues d'une tête humaine par scanner après avoir changé la valeur alpha.

3.6.2.1 l'utilisation des autres bases de données

Dans les résultats CT de la tête avec le crâne partiellement retiré peut révéler par exemple des lésions ou des tumeurs dans le cerveau.

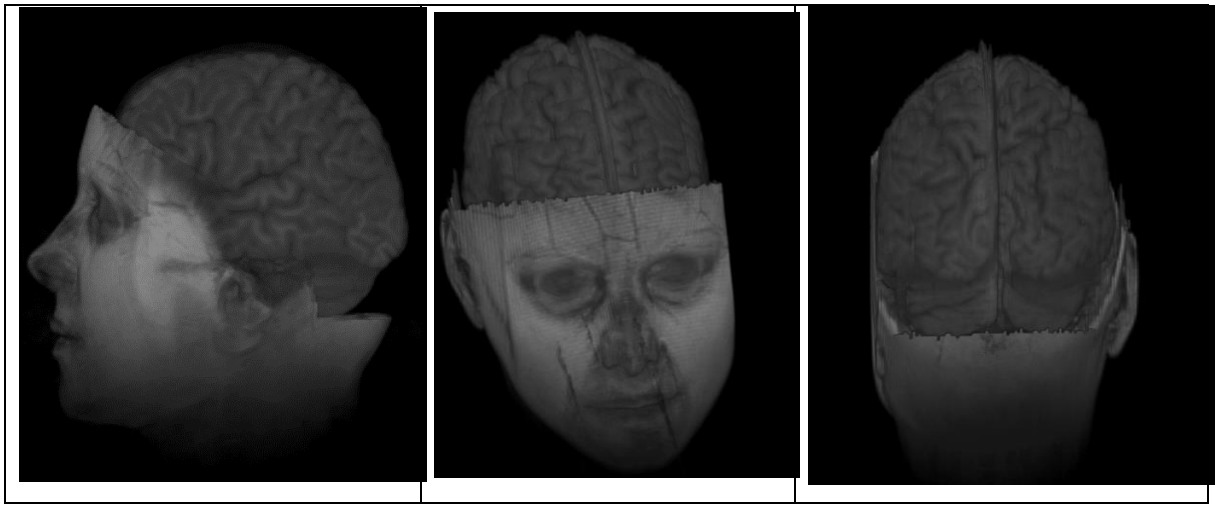


Figure 36: Représentation des vues d'une tête humaine avec le crâne partiellement retiré.

Dans l'Etude CT d'une tête de cadavre peut révéler des fractures, des traumatismes crâniens, des déformations de la mâchoire ou la denture.

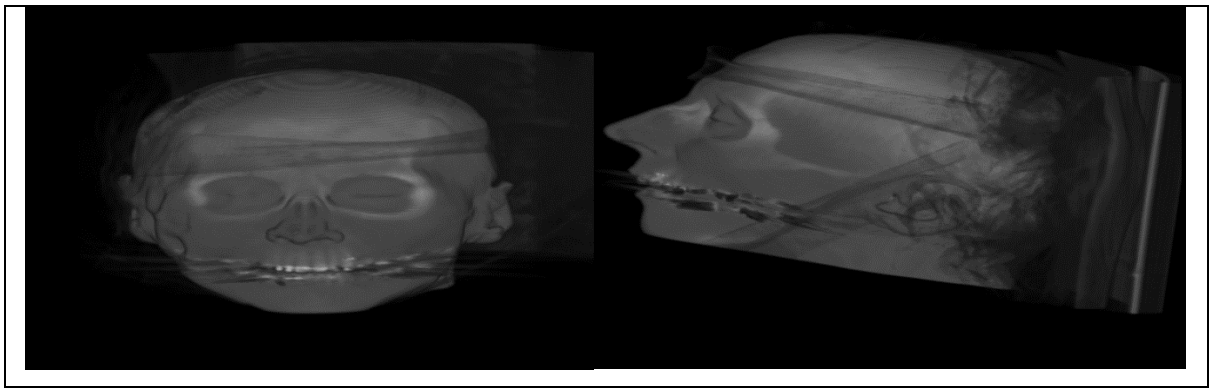


Figure 37: Etude CT d'une tête de cadavre.

3.6.3 Avec l'application d'une fonction de transfert

3.6.3.1 Selon la position du voxels

On colorant le volume selon la position du voxels : on a pris l'image 3D puis on a découpé sa taille en deux parties égales, puis on colorant chaque partie par une couleur définie.

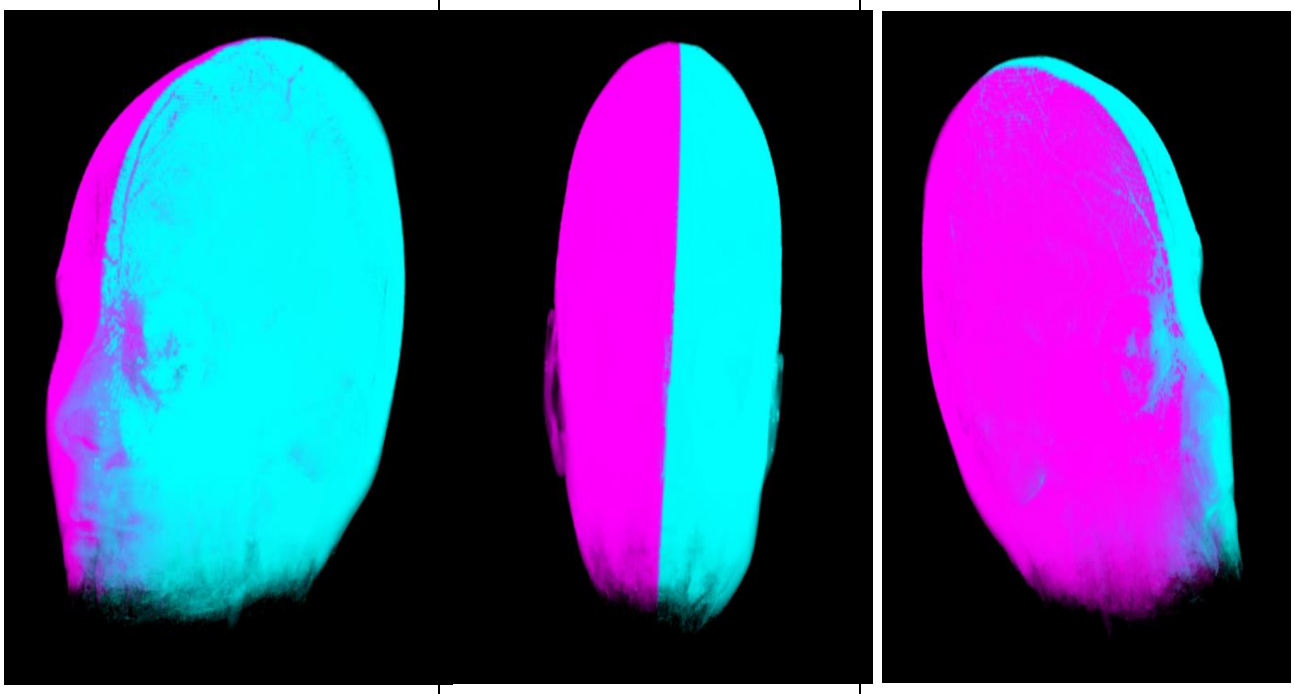
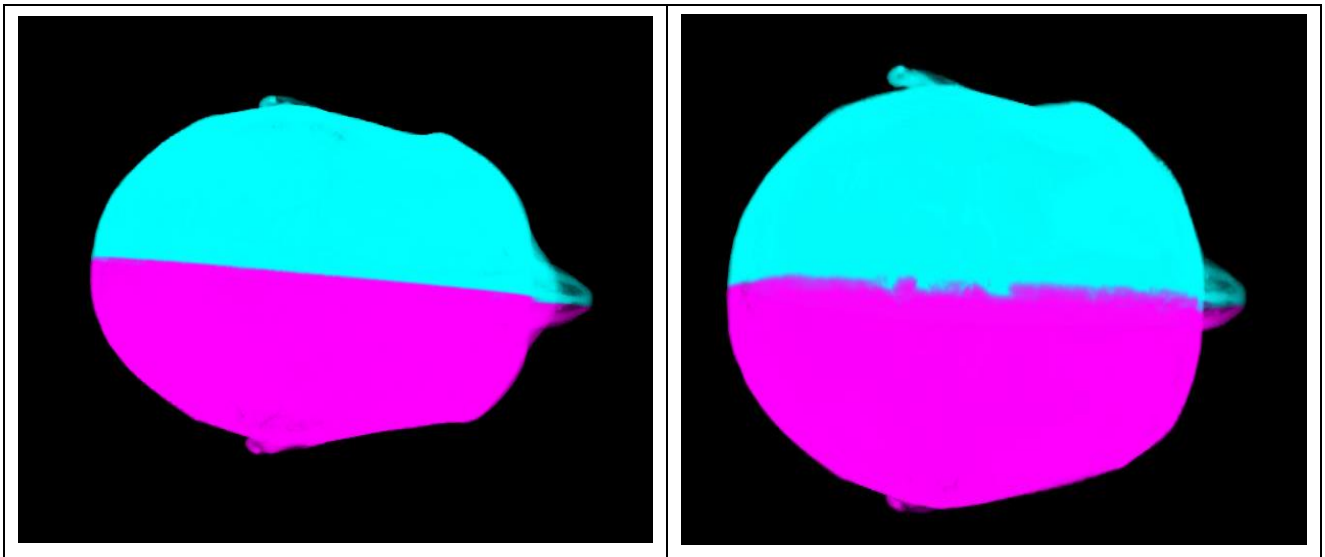


Figure 38: Des rotations sur l'axe Y.

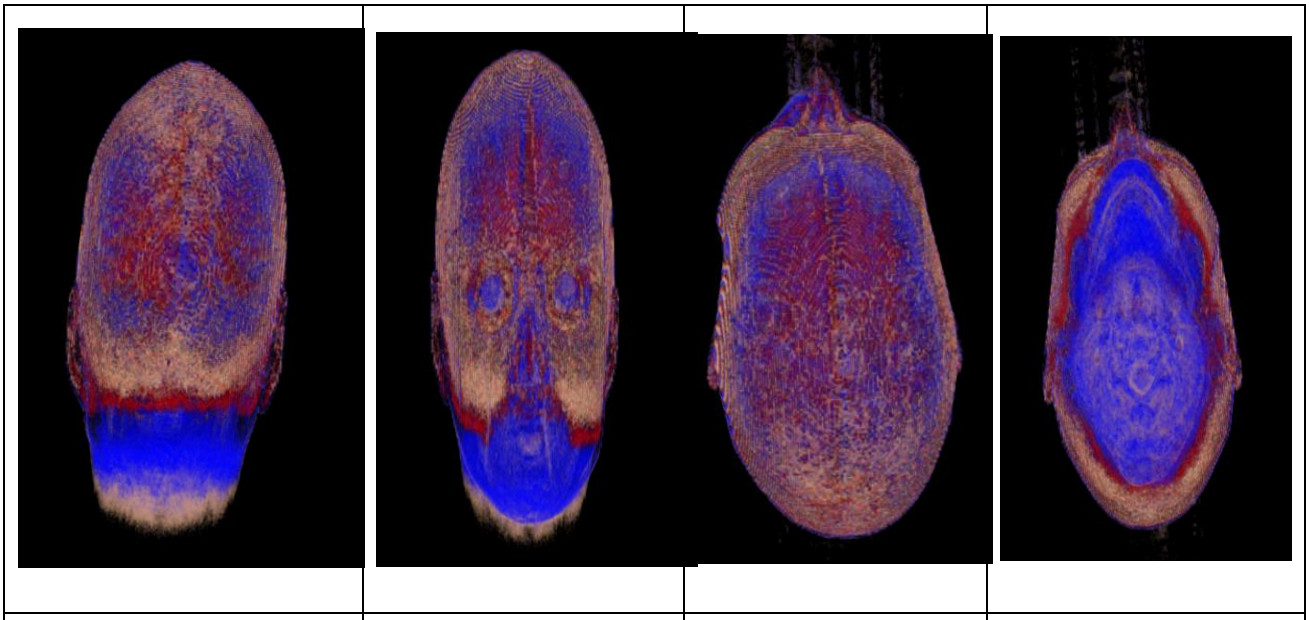


Zone supérieure	Zone inférieure

Figure 39: Des rotations sur l'axe X.

3.6.3.2 Selon la valeur de densité

On appliquant la fonction de transfert : tel que la valeur de densité est située entre une valeur maximale et minimale spécifique pour chaque couche.



Zone arrière	Zone avant	Zone supérieure	Zone inférieure

Figure 40: Représentation des vues d'une tête humaine par scanner après avoir appliqué la fonction de transfert.

Ces images représentent les étapes d'affichage des tranches de volume de gauche à droite.

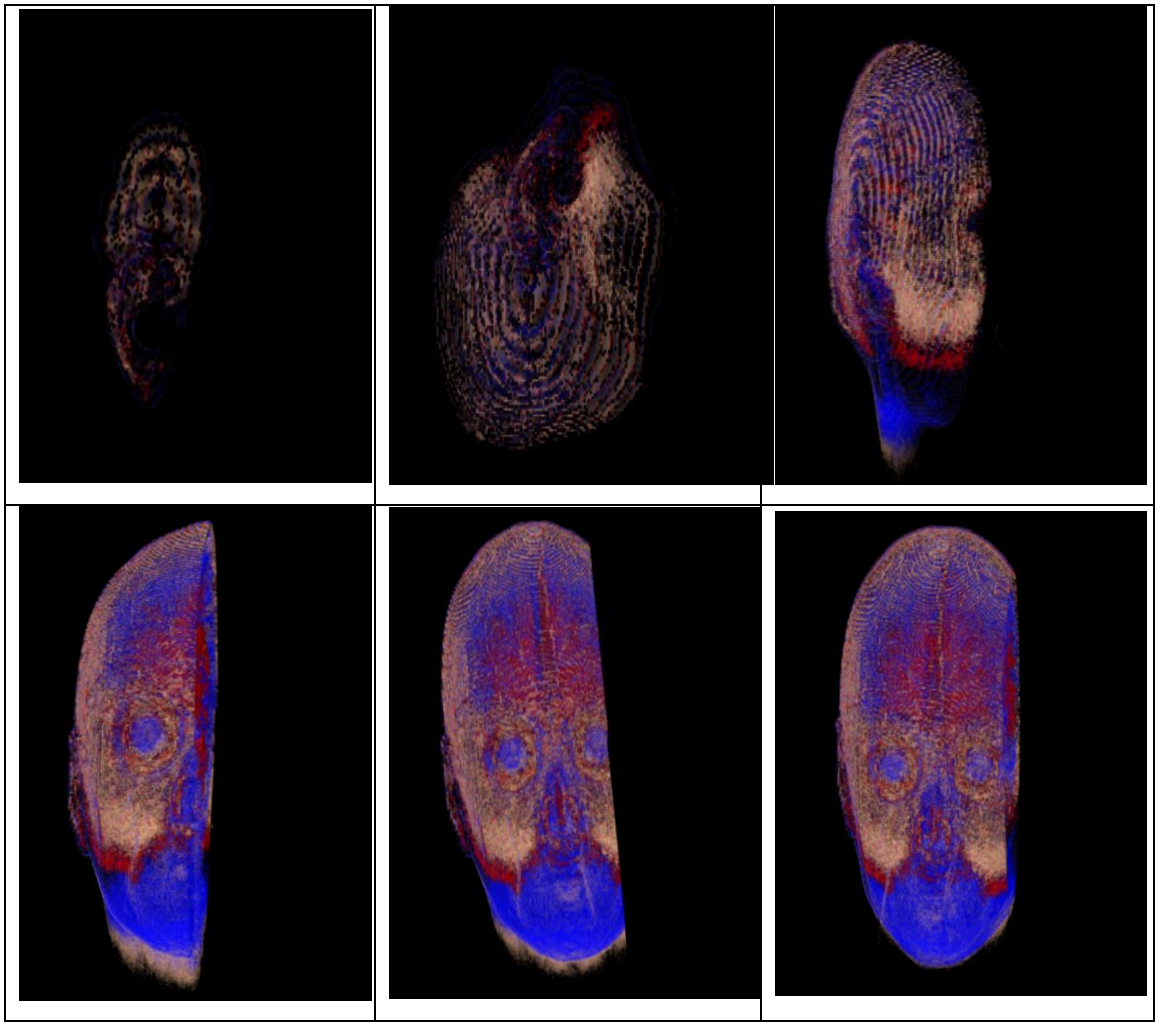


Figure 41: Les étapes d'affichage des tranches de volume de gauche à droite.

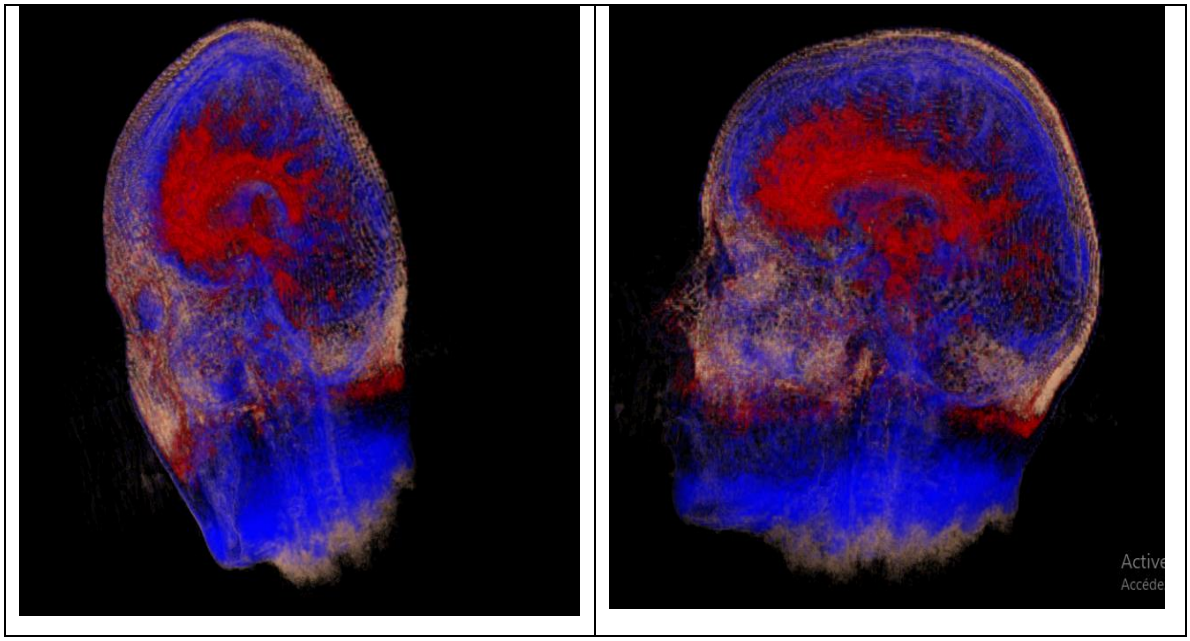


Figure 42: Une coupe longitudinale de la boite crânienne.

3.7 Temps de calcul

Dans ce tableau (tableau 4), nous présentons le temps de calcul des différents cas possibles de l'application de la fonction de transfert. On remarque généralement que :

Temps_calcul (sans fonction de transfert) < Temps_calcul (coloration par position voxels).

Temps_calcul (coloration par position voxels) < Temps_calcul (fonction de transfert).

Cas	Valeur maximale	Valeur minimale	La moyenne
Cas1 : Sans fonction de transfert	• **Temps=0.30 (seconde)**	• **Temps=0.13 (seconde)**	• **Temps=0.21 (seconde)**
Cas2 : Avec fonction de transfert selon position de voxels	• **Temps=0.66 (seconde)**	• **Temps=0.20 (seconde)**	• **Temps=0.43 (seconde)**
Cas 3 :Avec fonction de transfert selon la valeur de densité	• **Temps=2.20 (seconde)**	• **Temps=0.33 (seconde)**	• **Temps=1.26 (seconde)**

Tableau 4 : Temps de calcul des différents cas possibles de l'application de la fonction de transfert.

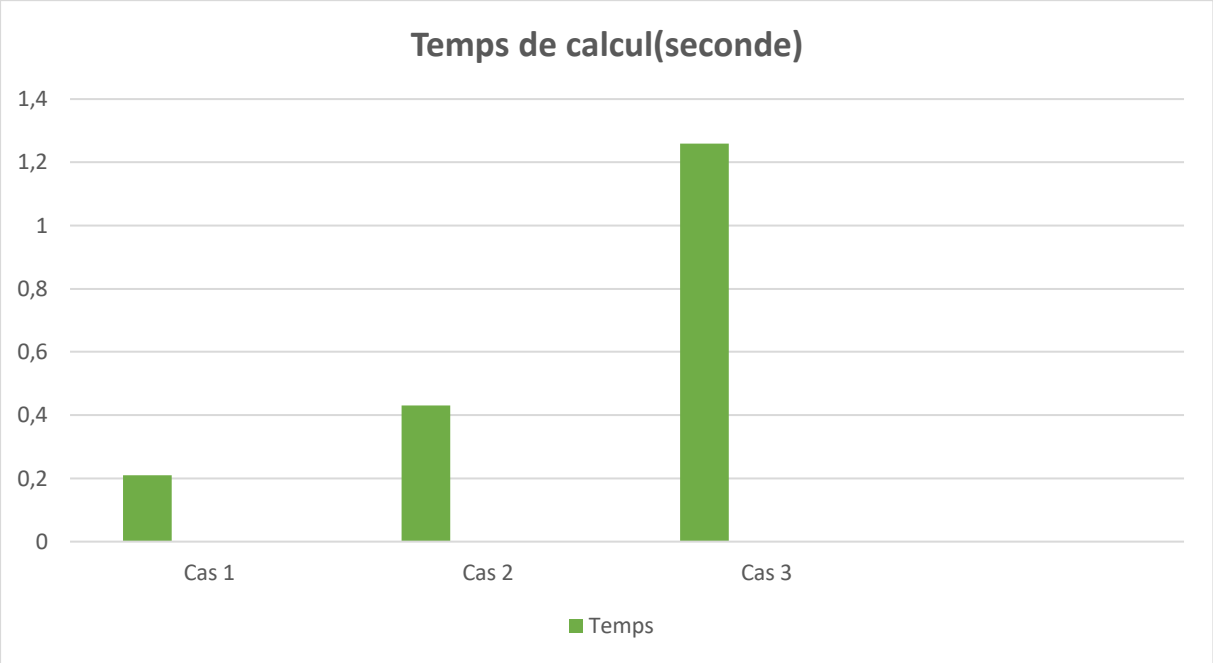


Figure 43: Comparaison du temps de calcul entre nos différents cas possibles de l'application de la fonction de transfert.

3.8 Conclusion

A l'issue de ce chapitre, notre projet d'études atteint sa fin. Tout au long de ce chapitre, nous avons abordé notre environnement de travail. Par la suite, nous avons expliqué notre architecture d'application afin de présenter les différentes principales parties d'implémentation de notre application réalisée.

CONCLUSION

Nous avons décrit dans ce mémoire une méthode de rendu volumique par l'intégration de la technique à base de texture. Ceci nous a permis de généraliser l'utilisation des textures pour obtenir des rendus d'une très bonne qualité et le gain du temps de rendu.

Pour le plaquage de texture 3D sur le quad support, nous avons utilisé une technique basée sur les couches d'images qui est très efficace pour un rendu volumique .Les résultats obtenus permettent une visualisation scientifique.

Le travail mené durant notre projet se situe dans le contexte de rendu volumique sur les textures 3D .Durant ce projet, nous avons ainsi conçu et réalisé une application qui prend en entrée des donnés discrets (image 2D) crées par des scanners médicaux, pour obtenir un volume 3D, à l'aide d'OpenGL et ses bibliothèques riches en fonction graphique.

Nous pouvons considérer que notre application est un point de départ pour l'exploration et l'intégration des texture 3D dans divers domaine comme :

- Médicales (données 2D, 3D, ainsi que temporelles).
- Biologiques (ADN, molécule, etc)
- Mécaniques (détection de zones sous contraintes, sous pression, etc)
- Physique (plasmas, physique des particules, etc)
- Mathématique (surfaces abstraites, haute dimensions)
- Capteurs (images satellites, fluide, etc)
- Graphes (Big Data, données internet, financières, etc)

Je propose au future l'intégration de réalité virtuel dans les rendus volumique et surtout dans le domaine médicale parce qu'il y a un très grande manque de diagnostique à distance et la simulation des opérations chirurgicales pour les étudiants de médecines, pour mieux comprendre l'anatomie du corps humain.

RÉFÉRENCE

Arie, K. & ((2005).). Overflow Of volume Rendring. Université de Paris.

Bahi, N. (2017). Ombrage volumique basé occultation ambiante pour une visualisation interactive de volumeOmbrage volumique basé occultation ambiante pour une visualisation interactive de volume de données. Université de Biskra, Département d'informatique.

Clang Compiler User's Manual. (n.d.). Retrieved from https://releases.llvm.org/3.1/tools/clang/docs/UsersManual.html#precompiledheaders ,(10/06/2020).

Coninx. (2013). Visualisation interactive de grands volumes de données incertaines : pour une approche perceptive, Université de Grenoble.

Doctissimo. (2017). Retrieved from https://www.doctissimo.fr/(26/4/2020).

Feschet., F. (1999). Techniques d'Imagerie Pour l'Aide Au Positionnement En Dosimétrie Conformationnelle. (Lyon): Université Lumière.

Game devlopement. (2014). Retrieved from Rendering collections of light sources : https://gamedev.stackexchange.com/questions/10858/rendering-collections-of-light-sources,(10/5/2020).

hadwager, M. (2008). Advenced illumination technique for GPU-Based volume raycasting, stanford unisersity,.

HEMIDI, D. (2005). Rendu Volumique efficace parune représentation à base de couches d'images. Département d'informatique. Université de Biskra.

Lakhdar, D. (2015). Utilisation des GPUs pour la reconstruction 3D en imagerie médicale. Département d'informatique, Université de Biskra.

Levoy, P. L. (1994). Fast Volume Rendering Using a Shear-Warp factorization of the viewing transformation. Stanford unisersity.

Levoy., M. (1988). Display of surfaces from volume data. In IEEE Computer Graphics and Applications, stanford ,unisersity. e.

Max. (1995). Optical Models for Direct Volume Rendering. University of California.

Meyer, A. (1998). « Textures volumiques interactives ». Université HAL.

Neyret., F. (2005). « Textures volumiques pour la synthèse d'images ». Université de paris.

Perlin. (2014). « coherent noise function over 1, 2 or 3 dimensions ». Lund universty.

Rezk-Salama, C. (2000). Interactive Volume on Standard PC Graphics Hardware Using Multi-textures and Multi-stage Rasterization. New York university.

Roussel, N. (2007). Introduction à OpenGL et GLUT. Retrieved from https://www.ljll.math.upmc.fr/frey/cours/Roscoff/OpenGL.pdf, Linköping University Post Print,(2/4/2020).

Schott, P. (2011). Introduction du repérage pour mieux enseigner la cinématique dans les trois repères classiques.

SERRANO, H. (2015, JANUARY 3). Understanding OpenGL Objects. Retrieved from https://www.haroldserrano.com/blog/understanding-opengl-objects,(6/7/2020).

Tutoriel : Développez vos applications 3D avec OpenGL 3.3. (2016). Retrieved from Développez vos applications 3D avec OpenGL 3.3: https://pub.phyks.me/sdz/sdz/developpez-vos-applications-3d-avec-opengl-3-3.html,(10/5/2020)

vimeo. (2017). Retrieved from How to make a Quicktime VR: https://vimeo.com/57657461,(9/5/2020).

Westover., L. (1990). Footprint evaluation for volume rendering. . The University of North Carolina at Chapel Hill.