Rendu temps réel pour écrans autostéréoscopiques

Mohamed El-Arbi Djebbar

Rendu temps réel pour écrans autostéréoscopiques

Rapport de stage
Master 2 de Recherche en Informatique (MRI)

Auteur : Mohamed-Elarbi DJEBBAR

Encadrés par : Kadi BOUATOUCH et Christian BOUVILLE

2009-2010

Table des matières

Chapitre 1

Introduction

Nous entendons beaucoup parler aujourd'hui de la TV 3D, la télévision du futur, la vision en relief de scènes 3D, etc. Le concept en lui-même n'est pas nouveau du fait qu'on pouvait déjà suivre des programmes TV en 3D nécessitant le port des lunettes polarisantes. Une nouvelle génération d'écrans 3D appelés écrans autostéréoscopiques (ne nécessitant pas le port de lunettes) est disponible sur le marché et offre maintenant une qualité et un confort de visualisation nettement améliorés par rapport aux modèles antérieurs. Ces technologies permettent d'afficher simultanément plus de 8 images correspondant à différents points de vue d'une même scène. De cette manière, l'observateur peut beaucoup plus facilement se positionner afin de percevoir une image en relief.

Des techniques de capture et de codage multivue existent pour l'image naturelle (films en 3D par exemple), mais le problème est plus complexe pour les applications 3D interactives (réalité virtuelle ou augmentée par exemple) car il faut calculer en temps réel le rendu de plusieurs images à entrelacer et à afficher sur des écrans autostéréoscopiques. En général, ces calculs dépassent les capacités de traitement des cartes graphiques actuelles dans le cas de scènes complexes.

Pour résoudre ce problème, il est possible de faire appel aux techniques de rendu basé image (image warping en particulier) pour calculer les différentes images correspondant aux différents points de vue. Ces techniques engendrent des défauts très gênants pour la vision stéréoscopique. Ceci est dû principalement au problème d'occultation entre objets et en deuxième lieu au problème d'aliasing dû à l'échantillonnage selon le motif plus au moins complexe des écrans autostéréoscopiques (étape nécessaire avant tout affichage).

Le but de ce stage est de **tester**, **adapter** et **proposer** des méthodes de rendu basé image pour écrans autostéréoscopiques. Ces méthodes doivent répondre aux exigences de qualité et de performances des applications 3D tout en exploitant au mieux les capacités des cartes graphiques modernes.

Ce rapport est structuré comme suit. Nous commençons par introduire les écrans autostéréoscopiques et leurs caractéristiques (chapitre 2), ensuite nous abordons quelques notions de rendu basé image, en particulier le travail de Berrety [2] et celui de Mark [13, 15] qui constitue la base de nos travaux (chapitre 3). Dans le même chapitre, nous détaillons très brièvement le problème de l'aliasing du point de vue traitement de signal. Le chapitre 4 détaillera nos travaux réalisés durant ce stage de Master. Nous terminons ce rapport par des conculsions et perspectives.

Chapitre 2

Les écrans 3D

2.1 Introduction

Les écrans 3D sont l'innovation la plus importante depuis l'apparition des écrans couleurs, ces écrans permettent une vision en relief des scènes 3D. Plusieurs types d'écrans sont apparus : écrans stéréoscopiques avec lunettes polarisantes, écrans autostéréoscopiques.

La vision stéréoscopique est une caractéristique de la vision humaine pour percevoir le monde réel. En effet le cerveau humain reçoit deux images 2D différentes mais très proches de la même scène 3D venant de nos deux yeux droit et gauche. Ces deux points de vue différents permettent au cerveau de fusionner les deux images et ainsi reconstruire la composante manquante, c'est à dire la profondeur.

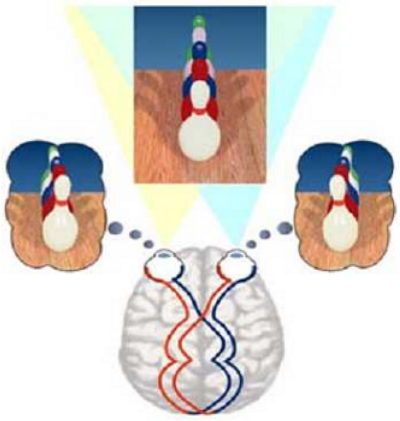

FIGURE 2.1.1: La vision stéréoscopique

En face des écrans 2D normaux, les deux yeux voient la même image affichée sur l'écran. Bien qu'on aperçoit la profondeur des objets l'un par rapport à l'autre, cette profondeur perçue n'est qu'une profondeur relative entre objets. En effet on ne peut pas voir les objets de la scène en relief, c'est-à-dire en déplaçant notre tête, on voit toujours les même parties de la scène qu'auparavant. Par vision en relief nous faisons référence à l'effet stéréoscopique (figure 2.1.1). Avec des écrans stéréoscopiques, des objets peuvent sortir de l'écran de telle manière que nous pouvons voir leurs différents détails selon notre position.

Pour pouvoir voir l'effet stéréoscopique, deux différentes images de la même scène doivent être envoyées aux deux yeux simultanément ; les écrans supportant cette technique s'appellent **les écrans stéréoscopiques**.

Avant de parler des technologies des écrans 3D autostéréoscopiques, nous allons d'abord aborder quelques notions importantes liées à la vision stéréoscopique.

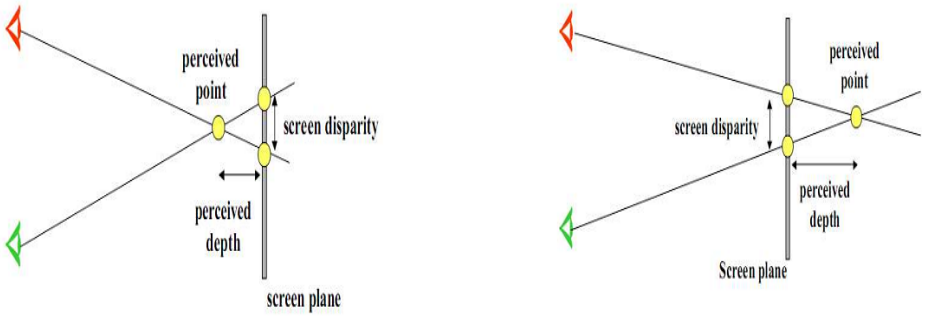

FIGURE 2.1.2: La perception de la profondeur

2.2 La parallaxe d'image :

La parallaxe d'image, par abus de langage « disparité de l'écran » (Screen disparity), est une caractéristique de l'écran 3D reflétant la différence entre les points sur l'écran, correspondants à un même point de l'objet 3D qu'on veut afficher en relief. (Figures 2.1.2).

La parallaxe se mesure soit en pixel soit en millimètres. Elle peut être :

- Positive : Les objets apparaissent derrière l'écran.
- Négative : Les objets apparaissent devant l'écran.
- Nulle : Les objets apparaissent sur l'écran.

2.3 La profondeur

La profondeur P que l'observateur aperçoit dépend de plusieurs paramètres et elle est donnée par la relation suivante (figure 2.3.1) : $P = \frac{v}{\frac{e}{d}-1}$, tel que :

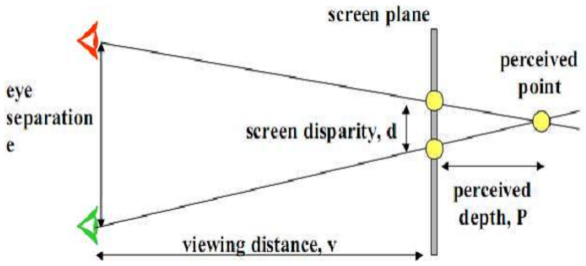

FIGURE 2.3.1: Calcul de la profondeur

- e : La distance entre les deux yeux.
- v : La distance de l'observateur de l'écran.
- d : La parallaxe.

Comme nous avons dit auparavant, pour voir en stéréoscopie, deux images différentes doivent arriver aux yeux (figure 2.1.1). Dans la vie courante, les rayons lumineux réfléchis par les objets de la scène donnent un aperçu différent pour chaque œil. Si nous voulons voir le même effet en utilisant des écrans, nous devons s'assurer que deux images différentes arrivent à nos yeux, tout en respectant les limites de fusion du cerveau humain. Les constructeurs des écrans autostéréoscopiques ont utilisé ce principe pour afficher en même temps deux images différentes (figure 2.3.2). Plusieurs technologies de ce type d'écran sont apparues.

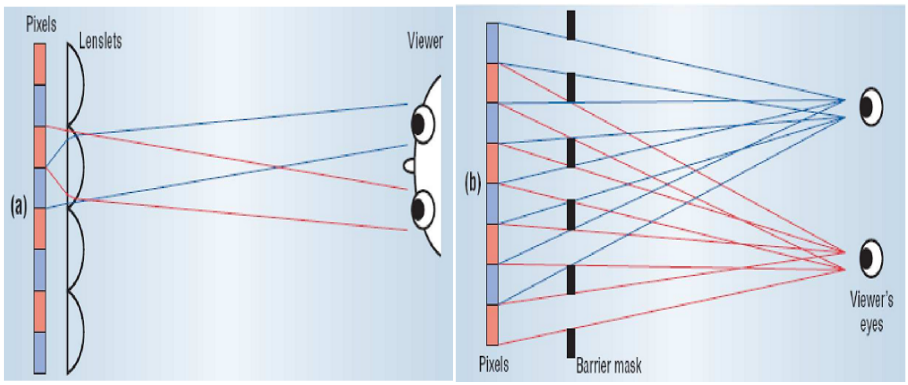

FIGURE 2.3.2: A gauche écran autostéréoscopique avec panneau lenticulaire, à droite avec barrière de parallaxe

2.4 Les technologies des écrans autostéréoscopiques

Les constructeurs des écrans autostéréoscopiques ont adopté deux technologies différentes mais qui produisent le même effet stéréoscopique, à savoir : Les écrans autostéréoscopiques avec panneau lenticulaire (à base de lentilles sphériques). Les écrans autostéréoscopiques avec barrière de parallaxe.

Autostéréoscopique signifie que ces écrans permettent une vision en relief sans utiliser des équipements supplémentaires comme des lunettes par exemple.

2.5 Multivue

Les écrans autostéréoscopiques sont conçus de façon à permettre à plusieurs observateurs de regarder en même temps la scène. Pour réaliser ceci, plusieurs vues de différents angles doivent êtres affichées en même temps pour voir la scène en relief de différents points de vue. Pour chaque vue, l'observateur reçoit deux images différentes : une par son œil droit et l'autre par son œil gauche (figure 2.4.1).

Pour donner une idée sur comment on arrive à afficher plusieurs vues en même temps, nous prenons l'exemple des écrans avec panneau lenticulaire. Dans ce type d'écran une grille régulière de petites lentilles sphériques, chaque lentille recouvre une zone de plusieurs pixels de l'écran (figure 2.5.1 à droite), La lumière de chaque sous pixel (rouge, vert et bleu) est déviée par la lentille dans une direction correspondant à une vue parmi les vues possibles (en générale il y a 9 vues). De ce fait, pour chaque vue seulement un sous ensemble de sous-pixels peut être vu (figure 2.5.1 à gauche).

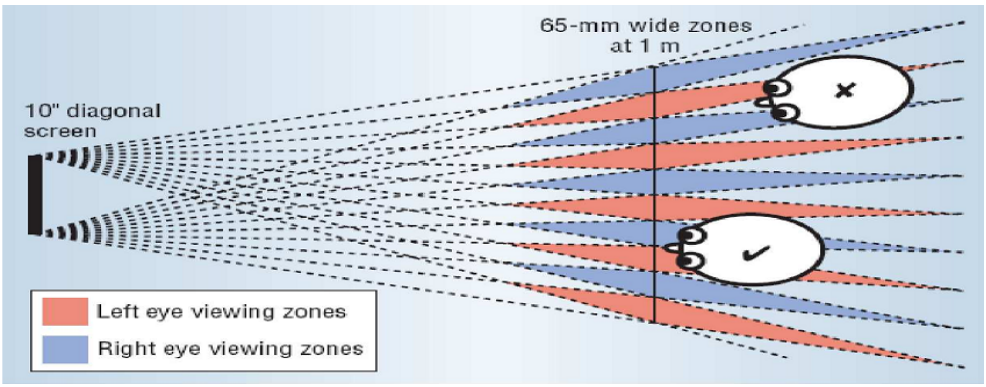

FIGURE 2.4.1: Le multivue

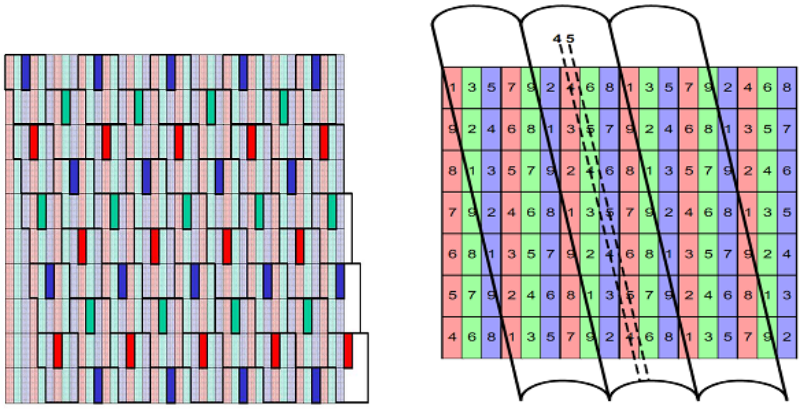

FIGURE 2.5.1: Architecture interne de l'écran

Cette technique d'affichage de vue repose sur le principe d'entrelacement des images. En effet avant de pouvoir afficher une scène en relief une étape de préparation de vues est nécessaire. Les images correspondant aux différentes vues sont d'abord acquises à partir de caméras calibrées ou bien synthétisées en utilisant des logiciels de rendu spécifiques. Une fois les différentes images acquises, elles sont entrelacées afin de produire une seule grande image affichable directement sur l'écran autostéréoscopique. Un algorithme d'entrelacement simple [3] est donné dans la figure 2.5.2.

Résumons maintenant cet algorithme. Dans l'image finale (Output) chaque pixel est composé de trois sous-pixels (rouge, vert et bleu), chaque sous-pixel appartient à une vue différente, c'est-à-dire selon l'angle de vue donné nous voyons seulement la couleur de l'un des sous-pixels. L'ensemble des sous-pixels vu à partir d'un certain angle (point de vue) constitue une vue parmi les vues possibles (en général 8 ou 9 vues). L'image finale affichable sur l'écran est constituée de sous-pixels venant des différentes images correspondant aux différents points de vue.

Ces images doivent être acquises soit par des caméras calibrées, soit par rendu correspondant à plusieurs positions de la caméra (utilisant OpenGL). Les deux solutions précédentes présentent des problèmes : le calibrage de 9 caméras n'est pas une tâche facile. D'autant plus que ce type d'acquisition est utilisé pour des scènes réelles. Pour des applications 3D interactives (Jeux vidéo 3D par exemple), nous devons effectuer

des rendus successifs pour chaque point de vue en changeant la position de la caméra, et nous répétons cette opération tout au long de l'exécution de l'application. Cette solution influe sur les performances de l'application et nécessite des cartes graphiques très puissantes. Si l'exécution se fait sur un terminal mobile, les performances à atteindre relèvent d'un vrai challenge. Une autre solution consiste à ne pas effectuer le rendu pour tous les points de vue, mais seulement pour certains points de vue (deux ou trois sur neuf par exemple) et puis effectuer le rendu pour les vues manquantes utilisant les techniques de rendu basé image.

```
Interleave(byte Input[9][wIn][hIn][3], byte Output[wOut][hOut][3])
{
    //Input images resolution: wIn x hIn (e.g. 533 x 400)
    //Output image resolution: wOut x hOut (e.g. 1600 x 1200)

    float ScaleX = wOut / wIn;
    float ScaleY = hOut / hIn;
    for( y=0; y<hOut; y++ )      //for each scanline
        for( x=0; x<wOut; x++ )  //for each pixel
            for( s=0; s<3; s++ ) //for each sub-pixel
            {
                ViewNr = .... //calculate the viewnumber of this sub-pixel
                Output[x][y][s] =
                    Sample(ViewNr, x/ScaleX, y/ScaleY, s);
            }
}

Sample(int ViewNr, float h, float v, int s) {
    return Input[ViewNr][round(h)][round(v)][s];
}
```

FIGURE 2.5.2: Un pseudo-code de l'entrelacement

Chapitre 3

Le rendu basé images : Etude théorique

3.1 Introduction

Le rendu basé image (Image-based rendering : IBR) est un terme apparu au milieu des années 90, pour décrire une nouvelle classe de méthodes de rendu [17].

Le rendu basé image consiste à calculer une image, correspondant à un certain point de vue, à partir d'un ensemble fini d'images de référence, au lieu de rendre cette image à partir d'un modèle 3D. Le problème majeur de ces techniques est l'apparition de trous dus aux problèmes d'occultation, et l'aliasing dû au traitement des images de référence (projection, déformation, ré-échantillonnage, etc.) pour reconstruire une image correspondant à une nouvelle vue. Dans la littérature en trouve plusieurs travaux traitant ces deux problèmes. Dans cette étude bibliographique nous allons aborder brièvement deux travaux : à savoir le travail de William Mark [15] et celui de Berretty [2]

3.2 Post-rendering image warping

William Mark [13, 15] propose une solution pour la génération de nouvelles vues de la scène, étant donné un modèle 3D de la scène. Il insiste sur le fait que la génération d'une image correspondant à un nouveau point de vue se fait en se basant seulement sur deux images de référence correspondant à deux points de vue différents. Cette technique est appelée « warping » (distorsion). Elle est justifiée par le fait que la distorsion des images de référence (warping) est simple en termes de complexité et de temps de calcul comparée au calcul d'un nouveau rendu à partir du modèle 3D de la scène pour chaque nouveau point de vue. Mark utilise un estimateur de mouvement pour trouver les positions des images références qui vont être utilisées pour la génération de plusieurs images correspondant aux nouveaux points de vue. La figure 3.2.2 donne une idée sur la méthode de Mark.

Cette solution présente plusieurs problèmes que Mark a essayé de résoudre :

Le problème de visibilité : l'image de certaines parties de la scène 3D ne peut être calculée en combinant les deux images de référence (trous dus aux parties occultées dans l'une des deux images). Mark a par conséquent développé un algorithme de remplissage de ces trous.

Le problème de reconstruction : Mark propose les étapes suivantes pour faire la reconstruction qui correspond à un nouveau point de vue à partir de deux images de références :

1. Construction de surfaces 3D à partir des images de référence.

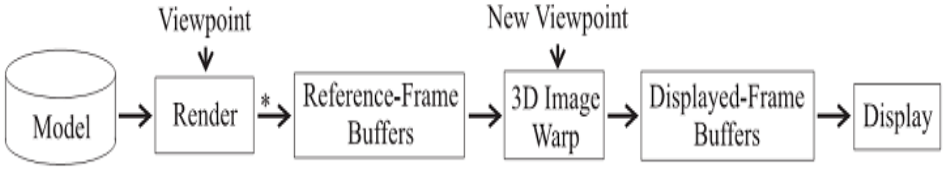

FIGURE 3.2.1: le pipline proposé par William Mark

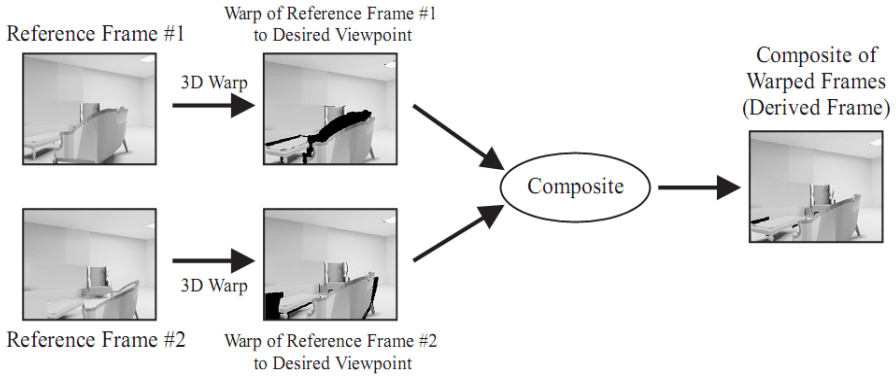

FIGURE 3.2.2: Le *3D warping* de Mark

2. Projection des surfaces dans l'espace de l'image à afficher.
3. Composition des surfaces projetées pour produire l'image finale.

Le travail de Mark William est plus adapté au rendu de scènes 3D pour des jeux de vidéo où le rendu est effectué au fur et à mesure que le joueur navigue dans la scène. L'estimation des positions des points de vue des images de référence est basée sur l'estimation du mouvement dans la scène. Une fois l'estimation des positions de point de vue de référence faite, les images de référence sont rendues à partir du modèle 3D de la scène. Les images intermédiaires seront calculées à partir de ces images de référence (figure 3.2.2).

3.3 Berretty : Le rendu RGBD

Le terme RGBD est la concaténation de RGB + D, où RGB représente les composantes rouge, vert et bleu, D fait référence au terme anglais « depth » qui signifie profondeur. Berretty [2] utilise la notion d'image 2.5D : couleur + profondeur. L'information sur la profondeur peut être exprimée soit comme étant la distance de l'objet par rapport à la caméra, ou bien en termes de parallaxe (figure 2.3.1). Nous avons vu dans la section 2.4 que l'image composite « multivues » des écrans autostéréoscopiques est le résultat de l'entrelacement des images correspondant à un certain nombre de vues possibles. Le problème qui se pose est la génération de ces images pour ensuite les entrelacer pour produire une seule image composite affichable sur notre écran autostéréoscopique. A partir d'une seule vue (la vue centrale) au format 2.5D, Berretty propose un algorithme de rendu qui permet de générer les vues manquantes (c'est-à-dire les autres images à entrelacer) en exploitant la disparité calculée à partir de l'information de profondeur fournie par l'algorithmique de rendu.

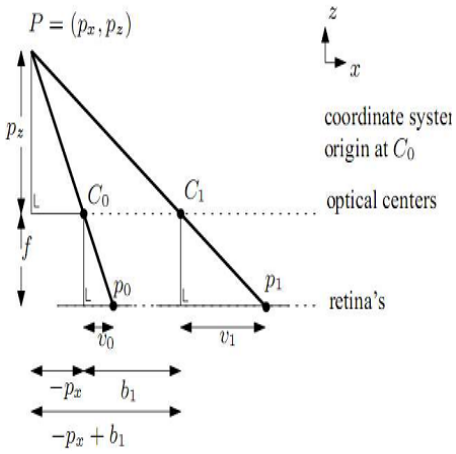

FIGURE 3.3.1: La disparité

3.3.1 La disparité d'un point par rapport à une caméra

Berretty a utilisé la notion de disparité pour calculer le décalage des pixels induit par un éventuel changement de point de vue.

Soit P un point de la scène projetée sur le plan de la rétine de deux caméras parallèles ayant pour centres optiques C_0 et C_1. La différence $d_1 = v_1 - v_0$ (figure 2.3.1) est connue sous le nom de disparité du point P dans la caméra C_1 par rapport à la caméra C_0. Sous hypothèse que les deux caméras ont la même distance focale f, la disparité d est donnée par : $d_1 = \frac{fb_1}{p_z}$,b_1 étant la distance séparant C_0et C_1 et p_z la distance du point P au plan parallèle à l'écran et contenant C_0 et C_1 Pour les autres caméras la disparité est donnée par $d_i = \frac{fb_i}{p_z}$, ce qui donne $d_i = c_i d_1$ avec $c_i = \frac{b_i}{b_1}$. Donc connaissant seulement la disparité d_1 et les distances inter-caméras (b_i) nous pouvant facilement déduire les autres disparités. Une fois la disparité de chaque point (d_1) calculée, nous procédons au calcul de son abscisse s_i sur l'écran correspondant à la caméra i (figure 3.3.2).

Les abscisses sont données par la relation suivante : $s_i = s_0 + \gamma k(\delta - d_1)$ avec kε{-4,-3,-2,-1, 0, 1, 2, 3, 4}est le numéro de la vue, $s_0 = -\alpha v_0$, $\delta = \alpha c_1$ et $\gamma = \frac{b_1}{\alpha}$.

Les paramètres α et δ sont des paramètres de contrôle. Tous les points ayant une disparité d1>δ (respectivement d1<δ) vont être observés devant l'écran (respectivement derrière l'écran). Les points ayant d1=δ seront affichés sur l'écran (ne sortent pas de l'écran). α contrôle la profondeur perçue du point.

En conclusion, ayant une image de référence en format 2.5D (RGBD), nous pouvons facilement calculer les coordonnées des autres vues utilisant la notion de disparité d'un point.

3.3.2 Les occultations

Après avoir calculé les disparités pour chaque point, la deuxième étape consiste à gérer les occultations. Berretty propose un mécanisme simple pour la gestion des occultations, il utilise une variable auxiliaire « extent » selon le principe décrit ci-dessous. Si on change le point de vue en allant vers la droite (figure 3.3.3), nous parcourons les pixels de l'image originale de la gauche vers la droite (c.-à-d. dans le sens inverse), et nous calculons les nouvelles projections des pixels de l'image de référence (Input Image) dans l'image ciblée. La variable extent retient le minimum des abscisses x des projections des pixels traités. Si la nouvelle abscisse du pixel (de l'image originale) ne diminue pas la valeur de la variable extent, alors ce pixel est occulté par rapport au nouveau point de vue. Dans la figure 3.3.4, on traite les pixels a, b et c en

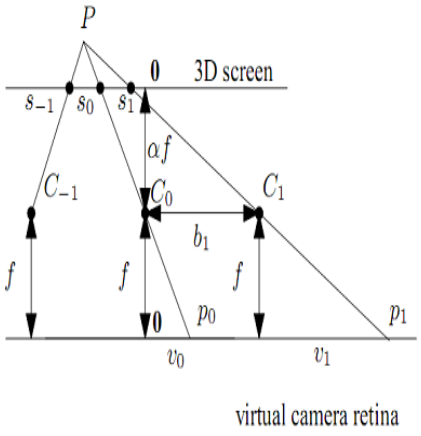

FIGURE 3.3.2: Les coordonnées d'un point dans l'écran 3D

respectant cet ordre. Le pixel c est occulté car son abscisse est plus grande que la variable extent qui contient l'abscisse du pixel b.

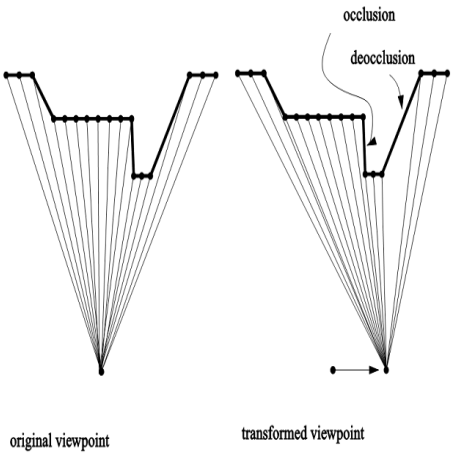

FIGURE 3.3.3: Les occultations

Le phénomène d'apparition de nouveaux objets est en général traité en répétant le fond (Background) dans les parties qui deviennent visibles. Le fond correspond aux pixels les plus profonds. Depuis l'apparition de l'article de Berretty [2], plusieurs travaux ont été faits pour optimiser la gestion des occultations [1, 8]

3.3.3 Le ré-échantillonnage

Un ré-échantillonnage est nécessaire pour avoir une image représentée par une grille régulière, formée de pixels de même taille (Figure 3.3.5). Berretty propose une méthode de ré-échantillonnage basée sur les travaux de Heckbert [9], cette méthode prend en compte la notion de multivues basée sur le calcul des disparités. Berretty utilise un préfiltre proposé par Koen Meinds [14] pour obtenir une image régulière (voir la section 3.4 pour plus de détails).

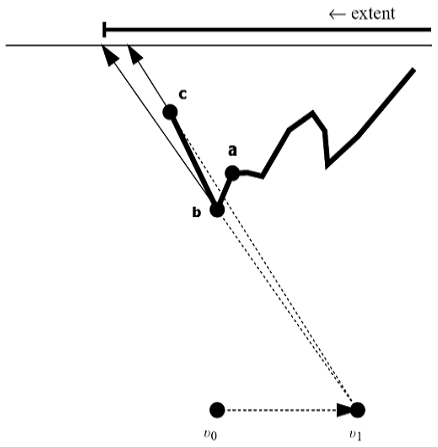

FIGURE 3.3.4: Gestion des occultations

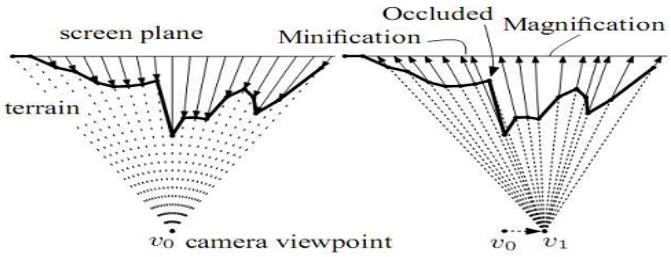

FIGURE 3.3.5: La nécessité du ré-échantillonage

Une implémentation du travail de Berretty a été réalisée et adaptée par Compen [3] pour faire le rendu des applications 3D (jeux 3D) sur des écrans autostéréoscopiques. Compen a utilisé la profondeur des pixels dans le Z-buffer pour construire l'image 2.5D nécessaire à la construction de 8 images (re-projection et ré-échantillonnage) qui correspondent au 8 vues de l'écran autostéréoscopique utilisé (écran Philips à 9 vues possibles). Les 9 vue seront entrelacées et affichées sur l'écran autostéréoscopique. RGBD à partir du contenu stéréo De nos jours le contenu 3D des programmes TV ou bien des films 3D projetés en salles de cinéma fait la une des médias. Ce contenu 3D est appelé stéréo du fait que l'on a deux signaux vidéo, cequi ne permet de visualiser qu'un seul point de vue. La séparation des deux signaux se fait en utilisant des lunettes polarisantes. Berretty [1] propose un mécanisme pour générer une image 2.5D à partir d'un signal 3D stéréo. Son travail est motivé par le fait que le format 2.5D peut être vu comme étant une couche intermédiaire indépendante du dispositif d'affichage (type d'écran).

La manipulation des images de référence, pour produire de nouvelles images qui correspondent aux nouveaux points de vue, fait appel à quelques notions du traitement de signal qui sont importantes pour la génération d'images intermédiaires de bonne qualité. C'est méthodes sont principalement l'image warping et le ré-échantillonnage (anti-aliasing). Dans la section suivante nous donnerons une vision générale de ces notions.

3.4 Image warping et l'anti-aliasing

3.4.1 Introduction

L'affichage d'une scène à partir d'un nouveau point de vue fait appel à la notion d'image warping qui signifie la déformation d'une image originale afin de produire une nouvelle image. Cela revient à définir une nouvelle relation spatiale entre les points de l'image. L'image warping est une technique de traitement d'image apparue dans les années 60. Le but principal était la correction des distorsions dues à l'utilisation de capteurs des caméras [18]. Le champ d'application de cette technique a connu depuis une grande explosion dans le domaine du graphisme : la génération des effets visuels, imagerie médicale, le plaquage des textures, la vision par ordinateur, etc. L'image warping consiste en trois étapes principales [9] : Reconstruire le signal continu à partir du signal discret (image de référence). Appliquer la déformation voulue (transformation spatiale, projection,...) au signal continu (warping). Faire un pré-filtrage (anti-aliasing) pour réduire la largeur de bande spectrale du signal et l'adapter à sa nouvelle fréquence d'échantillonnage. Ré-échantillonner le signal continu déformé selon la grille de sortie (nouvelle image). L'image warping fait appel à deux notions importantes : transformation spatiale et ré-échantillonnage.

3.4.2 Transformations spatiale

C'est la transformation géométrique qui décrit la correspondance entre l'image originale et l'image en sortie (dans notre cas l'image correspondant au nouveau point de vue). Les premiers travaux consistent essentiellement à appliquer ces transformations pour plaquer des textures [9]. Ensuite, avec l'apparition des notions du rendu basé image, l'utilisation du warping est devenue une nécessité pour produire « des images à partir d'images ». Mais l'application de ces transformations peut générer des images échantillonnées irrégulièrement, c'est-à-dire avec des pixels de coordonnées non nécessairement entières. De ce fait un ré-échantillonnage selon le motif de l'écran devient nécessaire [14].

3.4.3 Le ré-échantillonnage

Le ré-échantillonnage contient deux étapes principales : La reconstruction d'image et l'échantillonnage de cette dernière.

Initialement les pixels de l'image ont des positions dont les coordonnées sont entières et sont disposés selon une grille régulière. Mais une fois l'image déformée, le ré-échantillonnage peut générer de nouvelles positions de pixels ayant des coordonnées non entières et n'étant pas disposés selon une grille régulière. Pour faire face à ce problème une étape d'interpolation est nécessaire.

3.4.4 L'anti-aliasing

L'aliasing est un terme utilisé en traitement de signal. Il est utilisé pour décrire les défauts d'images et des artefacts désagréables qui résultent du non-respect du théorème de Shanon [9, 5]. L'aliasing apparait si la fréquence maximale du signal dépasse la moitié de la fréquence d'échantillonnage (la fréquence de Nyquist). Pour diminuer l'effet d'aliasing, le signal original est pré-filtré pour éliminer les hautes fréquences. L'élimination des hautes fréquences se fait en utilisant un filtre passe bas, mais cette solution peut introduire du flou dans l'image (perte de netteté), donc il faut éviter le plus possible cette perte d'information tout en assurant une bonne atténuation des hautes fréquences inutiles. Heckbert [9] présente plusieurs types de filtre utilisés pendant le pré-filtrage.

3.4.5 Etude de cas : Forward mapping

Berrety [2] propose une méthode de ré-échantillonnage adaptée aux écrans autostéréoscopiques multi-vues. En partant d'une image d'entrée correspondant à la vue centrale (input image), nous appliquons les étapes suivantes afin de produire l'image de sortie correspondant à l'entrelacement des 9 vues (output) qui sera affichée sur un écran autostéréoscopique.

La reconstruction

A partir du signal discret de l'image d'entrée (input image), nous appliquons un filtre de reconstruction r pour obtenir un signal continu : $f_c(u) = f(u) * r(u)$ (opération de convolution).

La déformation de l'image (image warping)

Dans cette étape le signal continue f_c est déformé à l'aide de l'application **m**. Autrement dit pour chaque x de l'espace de sortie, nous calculons son image inverse $m^{-1}(x)$ qui correspond à un point (ou un ensemble de points) dans l'espace d'entrée. La couleur de $m^{-1}(x)$ détermine celle de x. $g_c(x) = f_c(m^{-1}(x))$

Le pré-filtrage

Cette étape sert à éliminer les hautes fréquences dans le signal de sortieg_c, plus exactement limiter la bande de fréquences de ce signal (Band limit) en appliquant un pré-filtre h $g'(x) = g_c(x) * h(x)$

L'échantillonnage

La dernière étape consiste à échantillonner le signal continu (déformé et limité en bande) en multipliant le signal g' par un train d'impulsions i : $g(x) = g^{'}(x)i(x)$.

En combinant les 3 premières étapes nous aurons : $g^{'}(x) = \int h(x-t) \sum f(k) r(m^{-1}(t) - k) dt$ Ce qui donne $g'(x) = \sum f(k) \rho(x)$: avec $\rho(x) = \int h(x-t) r(m^{-1}(t) - k) dt$, qui est appelé le filtre du ré-échantillonnage. Le signal de sortie est une somme pondérée des contributions issues du signal d'entrée. Donc chaque pixel de l'image d'entrée contribue à un ou plusieurs pixels de l'image de sortie ; ce type de ré-échantillonnage est appelé forward mapping. Berrety [2] utilise le forward mapping pour produire l'image finale (output image) qui sera affichée sur l'écran autostéréoscopique. En fait il a développé un filtre qui intègre le calcul des disparités et la gestion des occultations, suivi d'un préfiltre qui traite les irrégularités dans l'image de sortie à savoir les réduction (minification), les agrandissement (magnification) et les positions non entières et non régulières des pixels (figure 3.3.5).

Ce chapitre a présenté un tour d'horizon des différentes notions théoriques du rendu basé image. Ainsi que deux méthodes de base pour ce type de rendu. Dans le chapitre suivant, nous allons décrire d'autres méthodes en relation étroite avec nos travaux. Nous y abordons aussi, avec un peu plus de détails, le travail réalisé durant ce stage.

Chapitre 4

Nos travaux

Introduction

Comme nous avons vu dans les sections précédentes, le rendu pour des écrans autostéréoscopiques nécessite l'entrelacement de plusieurs images correspondant aux différentes vues. A partir d'un modèle 3D de la scène 8 (ou 9 selon le modèle de l'écran) images correspondant aux vues sont générées en modifiant les positions de la caméra. Ces images seront entrelacées en une seule image qui sera affichée sur l'écran autostéréoscopique.

Berretty [2] a proposé un mécanisme de génération de l'image entrelacée en se basant sur une seule image qui correspond à la vue centrale mais augmentée d'information de profondeur. Mark [15] de son coté a proposé un mécanisme générique pour calculer des images intermédiaires à partir d'une ou plusieurs images de référence. Le principal problème des deux méthodes est qu'elles reposent sur un traitement pixel par pixel effectué par le CPU. Or dès que l'exigence en termes de résolution de l'image dépasse un certain seuil, le temps de génération devient prohibitif et ne répond donc pas à l'exigence temps réel de certaines applications de rendu 3D.

L'objectif du stage est de développer un logiciel de rendu temps réel multivue pour des écrans autostéréoscopiques. La mise en œuvre est faite sur une carte graphique programmable. Partant d'un sous-ensemble minimal d'images calculées, nous reconstruisons les images manquantes en exploitant au mieux les fonctionnalités des cartes graphiques.

4.1 Problématique et présentation synthétique

Durant ce stage, nous avons étudié le rendu temps réel pour les écrans autostéréoscopiques. La figure 4.1.1 montre la structure générale d'un logiciel de rendu 3D autostéréoscopique. Un modèle 3D de la scène est transmis à un module de génération de vues intermédiaires. Les vues intermédiaires, une fois générées, sont fournies au module d'entrelacement qui génère l'image finale affichable sur l'écran autostéréoscopique.

Dans nos travaux, nous avons ciblé le module de génération de vues intermédiaires car c'est ce module qui conditionne les performances et la qualité de notre logiciel de rendu autostéréoscopique. Le fait que les nouveaux écrans autostéréoscopiques peuvent accepter jusqu'à 40 images intermédiaires, rend l'étape de génération de vues intermédiaires cruciale car les performances globales du logiciel de rendu autostéréoscopique en dépendent fortement.

Dans cet esprit, nous avons **testé** deux méthodes de générations de vues intermédiaires qui ont été proposées dans la litérature pour ce type d'écran, à savoir la méthode multipasse [7] et la méthode utilisant

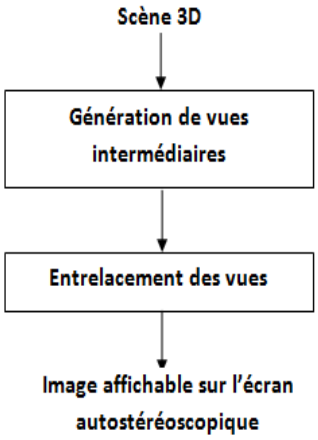

FIGURE 4.1.1: Les étapes du rendu autostéréoscopique

le Geometry Shader [4]. Nous avons aussi **adapté et testé** deux autres méthodes qui ont été proposées pour d'autres contextes : la méthode du 3D warping proposée par Jiang [12] et la méthode proposée par Mark[13, 15]. Enfin nous avons **proposé et testé** une méthode dite mixte qui combine les deux méthodes de Jiang et de Mark.

Nous avons classifié les méthodes en deux classes : les méthodes utilisant le rendu classique et les méthodes utilisant les techniques de rendu basé image (Figure 4.1.2).

Dans ce qui suit nous allons détailler les deux classes de méthode et montrer comment nous avons pu exploiter les nouvelles capacités offertes par les nouvelles cartes graphiques dans notre contexte bien particulier. Plusieurs détails techniques sont expliqués afin de bien comprendre le pour et le contre de chaque méthode. Nous avons jugé plus utile de mettre les détails techniques dans ce chapitre et non pas dans les chapitres précédents dans un souci de clarté.

4.2 Les méthodes testées : Le rendu classique

Introduction

Par rendu classique, nous faisons référence au rendu utilisé dans les jeux vidéo et dans les applications 3D. Dans ce type de rendu la scène et ses objets sont définis par des modèles 3D texturés à base de sommets connectés selon des primitives (point, triangle, quad, etc.).

Comme nous avons vu dans les sections précédentes, les écrans autostéréoscopiques multivue nécessitent la génération de plusieurs images correspondant aux différents points de vue. Une méthode naïve consiste à effectuer des rendus successifs pour générer les images nécessaires aux multivue. Dans cette optique, nous avons testé deux méthodes suivant ce paradigme, à savoir : le rendu multipasse et le rendu utilisant le *Geometry Shader*.

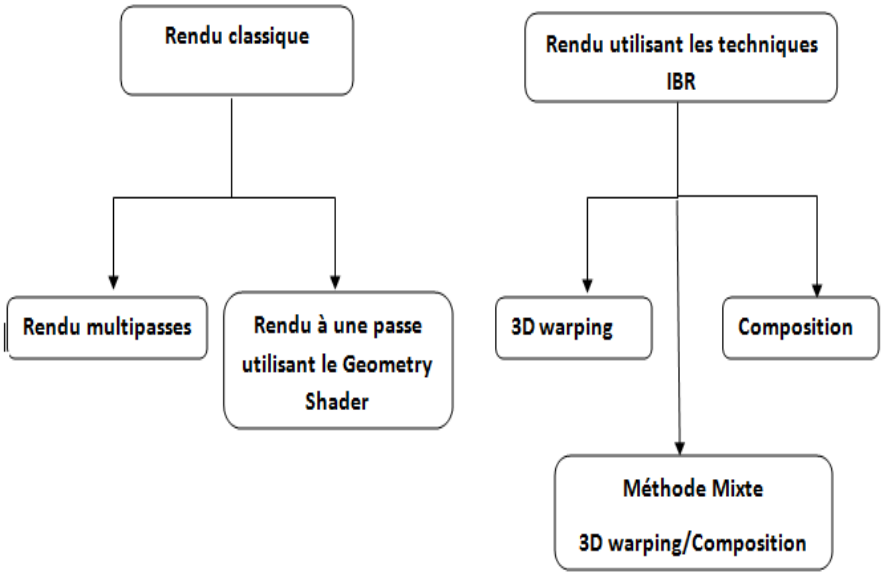

FIGURE 4.1.2: Les méthodes de génération de vues intermédiaires

4.2.1 Rendu multipasse

Cette méthode génère les vues par des rendus successifs à partir d'un modèle 3D de la scène. Autrement dit, nous faisons appel au pipeline graphique autant de fois qu'on a de vues à générer sans penser à aucune optimisation. Cette méthode a été intégrée par Sourimant Gaël dans une application qu'il a développée durant son stage post doctorat au sein de l'équipe TEMICS [7].

Pour le rendu multipasse, les vues sont générées selon le schéma de la figure 4.2.1. les caméras sont parallèles et positionnées sur le même axe horizontal (figure 4.2.1(a)). Les distances inter-caméra sont calculées selon la formule suivante (Figure 4.2.2, [7]) : $\Delta_{cam} = \alpha \frac{\Delta_{eye} d_f}{d_u}$ où :

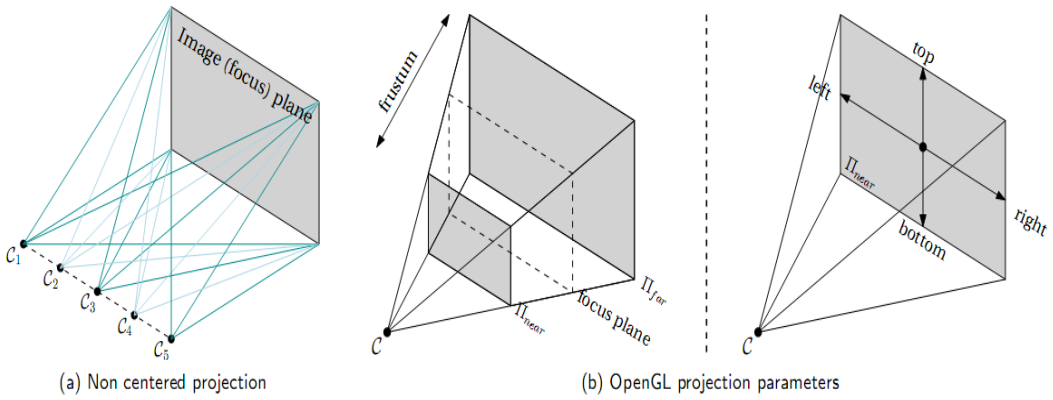

FIGURE 4.2.1: Paramétrage du multivue

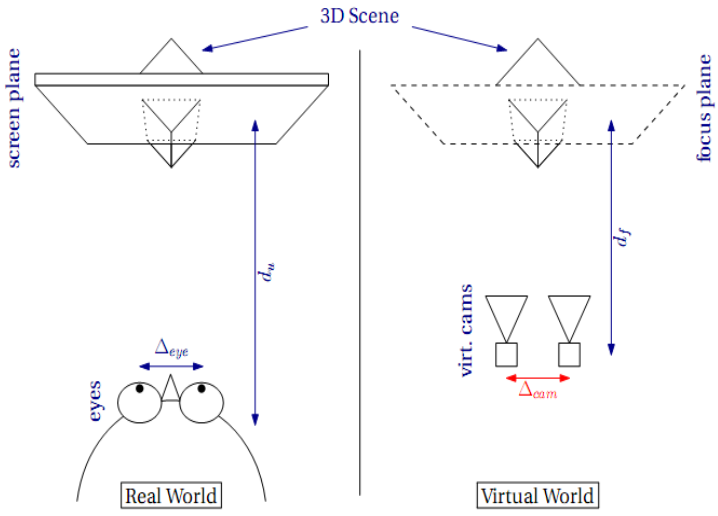

FIGURE 4.2.2: Correspondances caméras - yeux

Δ_{cam} est la distance inter-caméras, Δ_{eye} la distance inter-oculaires, d_f la distance focale , d_u la distance séparant l'utilisateur de l'écran et α est un paramètre qui contrôle l'effet autostéréoscopique.

Une fois les caméras configurées, nous devons rendre les différentes vues en mémoire interne (*offscreen*), ensuite les récupérer pour les entrelacer afin de produire l'image finale affichable sur notre écran autostéréoscopique. Pour le rendu *offscreen,* Gaël Sourimant a utilisé l'extension *Framebuffer Object* de OpenGL (FBO) que nous avons adoptée pour le reste du travail.

Le *Framebuffer Objec*t est une extension OpenGL qui permet d'effectuer un rendu en mémoire interne (*offscreen*) d'une manière intuitive et simple. Une fois activé, le *Framebuffer Object* capture tout ce qui doit normalement être affiché sur l'écran et le stocke dans une ou plusieurs textures qui y sont attachées. Ces dernières peuvent être utilisées ou modifiées plus tard (application de filtres, entrelacement, .etc.) [7, 16]. Dans la suite nous décrivons sommairement le rendu multipasse.

L'algorithme de rendu en mémoire interne (*offscreen*) des images intermédiaires

Cet algorithme se résume ainsi :

- Activer le *Framebuffer Object*
- Pour chaque point de vue :
  - Sélectionner la texture (attachée au *Framebuffer Object*) correspondant à la vue courante.
  - Configurer la caméra (Figure 4.2.1 a).
  - Rendre la scène (à partir du modèle 3D).
- Désactiver le Framebuffer Object.

Après cette étape, les images intermédiaires sont stockées dans des textures.

L'entrelacement et l'affichage de l'image finale (l'image entrelacée)

Une fois que les différentes vues sont rendues dans des textures (attachées au FBO), l'étape finale consistera à afficher l'image finale entrelacée. Le principe est le suivant :

- Attacher les textures aux unités de texture OpenGL ,

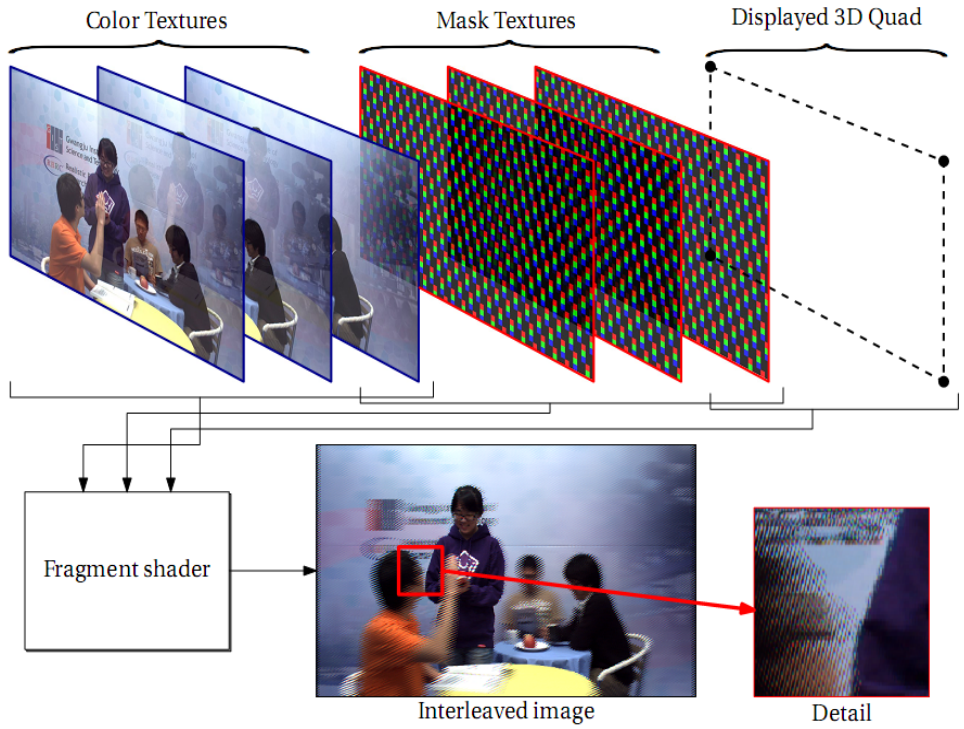

FIGURE 4.2.3: Entrelacement

- Attacher les filtres d'entrelacement ,
- Activer le shader d'entrelacement,
- Dessiner un quadrilatère qui couvre toute la zone d'affichage,
- Désactiver le shader.

Le *Program Shader* (plus exactement le *Fragment Shader*) va affecter à chaque pixel (fragment) du quadrilatère une couleur calculée selon la formule suivante :

Pour chaque pixel x, $couleur(x) = \sum_{i=1}^{n} texture_i(x) * mask_i(x)$, où n est le nombre de vues. (figure 4.2.3)

Cette méthode donne les meilleurs résultats en termes de qualité. Ceci est bien justifié car les images intermédiaires sont générées directement à partir du modèle 3D de la scène. Par contre les performances de cette méthode sont inversement proportionnelles à la complexité de la scène en termes de nombre de polygones, méthode de rendu, calcul de la luminance, etc.

La deuxième méthode, que nous avons testée, a pour objectif d'améliorer les performances de la méthode multipasse en exploitant au mieux le parallélisme offert par les cartes graphiques..

4.2.2 Le rendu à une passe utilsant le *Geometry Shader*

Les *Program Shaders* sont des programmes exécutés par la carte graphique (GPU). Ils servent à remplacer les fonctions figées implémentées dans la carte graphique (fonctions cablées) par d'autre fonctions écrites par le programmeur [11]. Ils interviennent à différents niveaux du pipeline graphique (Figure 4.2.4). Les Shaders les plus utilisés sont le Vertex et *Fragment Shaders*. Dans les cartes graphiques modernes, l'étape *Geometry Assembly* du pipeline est devenue programmable ce qui donne aux programmeurs plus de flexibilité pour gérer les sommets et leurs connexités (ajout ou suppression de sommets ou d'arêtes).

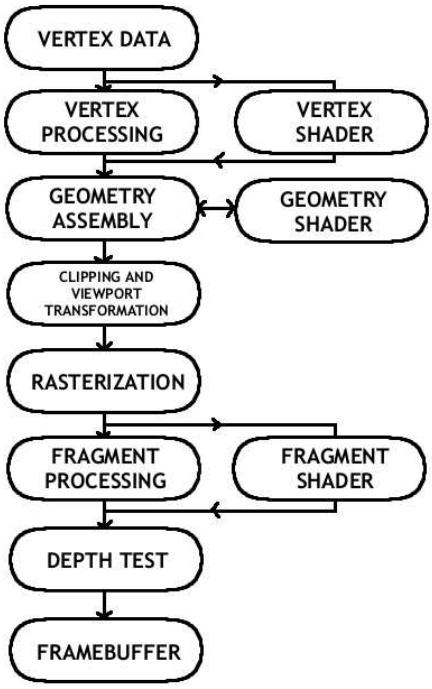

FIGURE 4.2.4: Le pipeline graphique programmable

Tout modèle 3D est défini par la donnée d'un ensemble de sommets connectés pour former des primitives : Points, Triangles, Quads,.etc. François et al. [4] ont proposé d'utiliser le Geometry Shader pour dupliquer la géométrie autant de fois qu'il y a de vues à générer. La figure 4.2.5 montre un aperçu de sa méthode.

Cette méthode repose sur deux techniques principales :

- Le *Geometry Shader* : pour dupliquer la scène autant de fois qu'on a de vues à générer.
- Le *Multi Rendering Target (MRT)* : pour avoir en sortie du pipeline graphique plusieurs images au lieu d'une seule image. En fait le but est de rendre chaque copie de la scène dans une zone mémoire différente (plus exactement : Buffer différent).

En fait il y a une grande similitude d'un point de vue logique entre la méthode multipasse et la méthode utilisant le *Geometry Shader*. Par exemple le nombre de sommets que traite le pipeline graphique est le même dans les deux méthodes. L'avantage de la deuxième méthode est que les traitements correspondant aux 8 vues sont effectués une seule fois. Par exemple pour un sommet donné, la couleur, la normale, la position dans la scène, etc. sont les mêmes quelle que soit la position de la vue choisie. François de Sorbier propose d'exploiter cette remarque dans l'étape de rendu *offscreen* décrite dans la section 4.2.1.

Les *Framebuffer Objects* offrent aussi la possibilité d'utiliser la technique du *Multi Render Target* (MRT), c'est-à-dire que la sortie du pipeline graphique est un ensemble de buffers de couleur au lieu d'un seul. Si nous activons le MRT, nous allons avoir exactement la même image rendue dans chaque buffer. L'idée de François de Sorbier est d'exploiter le MRT mais en faisant en sorte que le résultat final dans chaque buffer corresponde au rendu associé à chaque point de vue, comme le montre l'algorithme suivant :

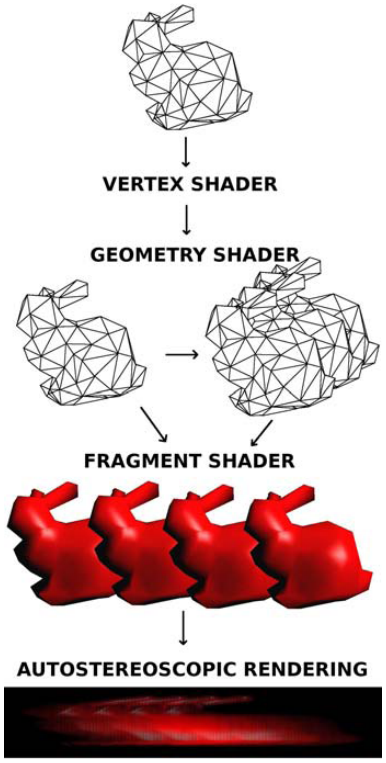

FIGURE 4.2.5: Apperçu sur la méthode utilisant le Geometry Shader

L'algorithme "Rendu multivue en une passe" :

- Activer le *Framebuffer Object*
- Activer le *Multi Rendering Target.*
- Activer le *Program Shader*.
- Passer les paramètres nécessaires au *Program Shader* : les matrices *Projection* et *Modelview* de chaque vue.
- Rendre la scène (à partir du modèle 3D) une seule fois.
- Désactiver le *Program Shader*.
- Désactiver le *Framebuffer Object*.

Le *Program Shader* est composé de trois programmes qui s'exécutent dans l'ordre de la figure 4.2.4

- ***Vertex Shader*** : traiter les sommets : couleur, normale, coordonnées de texture,...
- ***Geometry Shader*** : dupliquer les primitives selon le nombre de vues et appliquer pour chaque copie de la scène les matrices Modelview et Projection correspondantes.
- ***Fragment Shader*** : rendre chaque fragment (selon la copie de la scène à laquelle il appartient) dans le buffer correspondant (chaque copie sera rendue dans un Buffer différent).

Comme pour la méthode précédente, les différentes vues générées sont stockées et sont prêtes à être entrelacées selon le même principe exposé plus haut.

Le problème majeur de cette méthode est une limitation technique des FBOs. En effet chaque FBO contient un seul buffer de profondeur (*depth buffer*) pour sauvegarder le résultat du test de profondeur (*Depth Test*). De ce fait, il faut désactiver le test par défault de OpenGL et utiliser d'autres algorithmes comme l'algorithme du Peintre (*Painter*) mais c'est très couteux car il faut trier toutes les faces de la scène selon

leur profondeur par rapport à chaque caméra. Dans un article plus récent François [6] propose une méthode légèrement améliorée qui intègre un mécanisme de test de profondeur.

Les résultats obtenus en termes de performance sont décevants (voir la section 4.5). Nous pensons que le nombre de processeurs utilisés dans l'étape Geometry Assembly (les processeurs qui exécutent notre *Geometry Shader*) n'est pas suffisant pour dépasser les performances de la méthode multipasse.

Les deux méthodes déjà présentées reposent sur le principe de rendu multiple d'un modèle 3D. Dans le cas du rendu photoréaliste, à l'exemple de la méthode de Monte Carlo (*Monte Carlo Rendering*), il est possible d'atteindre des temps de traitement acceptables juste pour un rendu monovue et non pas pour un rendu multivue. Dans ces situations le calcul d'image à partir d'images (*Image based Rendering*) devient indispensable pour atteindre des performances répondant à l'exigence temps réel de plusieurs applications. Dans ce qui suit, nous expliquerons les méthodes que nous avons adaptées et testées.

4.3 Les méthodes adaptées : Le rendu basé images

Introduction

Dans la section précédente nous avons testé deux méthodes qui s'appuient entièrement sur les fonctionnalités du pipeline graphique. Ainsi nous avons montré leurs limitations pour des scènes très complexes ou pour des rendus photoréalistes. A celà s'ajoutent les exigences temps réel et de qualité qui sont considérables dans plusieurs applications.

Le rendu basé image (IBR, calcul d'image à partir d'images) peut améliorer les performances et même donner de bons résultats en termes de qualité s'il est bien utilisé. De ce fait nous avons adapté quelques méthodes de rendu basé image (qui ont été proposées dans des contextes plus généraux) au cas du rendu multivues pour écrans autostéréoscopiques.

Plusieurs approches étaient envisageables. Nous avons choisi deux méthodes : la méthode du *3D warping* et la méthode de composition basée aussi sur le *3D warping* mais utilisant plus d'une image de référence dans le calcul de nouvelles images pour réduire les effets de découvrement. Nous avons aussi proposé une méthode dite mixte qui combine les deux méthodes précédentes.

4.3.1 Le 3D warping

Introduction

Dans la section 3.3, nous avons développé le cadre théorique de la méthode de Berretty. En fait pour chaque pixel de l'image de référence, Berretty calcule les positions des pixels dans l'écran autoséréoscopique qui représente le même point 3D vu à travers un pixel de l'image de référence. En quelque sorte c'est un calcul implicite des images correspondant aux vues intermédiaires directement dans l'image finale (entrelacée). Berretty propose aussi un filtre pour gérer l'anti-aliasing. L'avantage de ce filtre est l'utilisation du *forward mapping* (voir section 3.4.5), c'est à dire qu'on n'a besoin de connaitre que la couleur du pixel et celles de ses voisins munis de certains poids pour trouver la couleur filtrée dans l'image finale (entrelacée). Néanmoins ce filtre n'est pas générique car il dépend directement de la grille de l'écran autostéréoscopique. Berretty intègre aussi un mécanisme de gestion d'occultation. Enfin, la méthode de Berretty est conçue pour une implémentation cablée dans un circuit intégré aux écrans autostéréoscopiques.

Dans notre cas, nous avons, en général, à notre disposition des cartes graphiques modernes qui sont puissantes et implémentant de bons filtres gérés par le matériel graphique. Dans cette optique, nous avons adapté et modifié la méthode de Berretty tout en gardant son principe de base à savoir effectuer un seul un rendu, celui de la vue centrale.

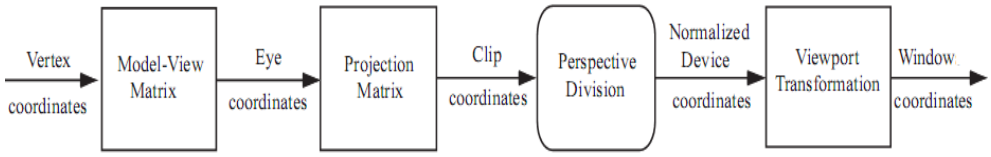

FIGURE 4.3.1: Pipeline OpenGL

Pour la génération de l'image finale entrelacée, nous avons décidé de générer les images intermédiaires correspondant aux différents points de vue d'une manière explicite, c'est à dire calculer les images intermédiaires ensuite les entrelacer. Ce choix est motivé par plusieurs raisons que nous citons un peu plus tard dans cette section. Tout d'abord nous allons décrire le 3D warping, méthode utilisée pour le calcul des images intermédiaires

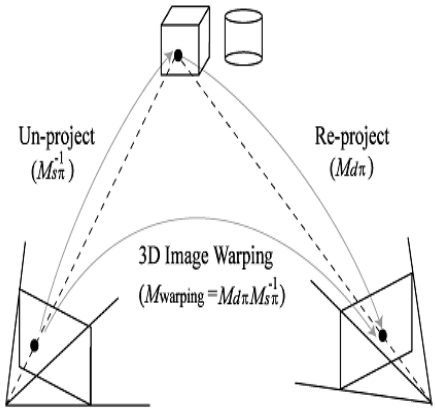

FIGURE 4.3.2: 3D warping

Détails techniques de la méthode

Le *3D warping* consiste à reprojeter l'image de référence, prise pour un certain point de vue, sur le plan image de la caméra associée à un nouveau point de vue. L'idée est de retrouver les coordonnées 3D du point de la scène à partir des coordonnées pixels augmentées de l'information de profondeur et ensuite de reprojeter le point 3D selon le nouveau point de vue (figure 4.3.2). Ceci est possible car nous connaissons les paramètres de transformation du point de vue de référence et celui du nouveau point de vue.

Les coordonnées de chaque pixel$(x_{écran}, y_{écran}, depth)$, $x_{écran}, y_{écran}$ étant les coordonnées du pixel dans l'image de référence et $depth$ sa profondeur, seront multipliées par la matrice de transformation inverse pour retouver le point 3D correspondant. Pour comprendre le principe prenant l'exemple de l'API OpenGL (API graphique utilisée dans ce stage) :

Dans OpenGL, deux matrices de transformation(Projection et Modelview), une division (normalisation) et un fenêtrage (viewport) constituent le pipeline graphique. La chaine de transformation de la figure 4.3.1 que subit un point 3D de la scène peut être décrite comme suit

- Modelview : $P_{eye} = M_{modelview} P_{object}$
- Projection : $P_{clip} = M_{projection} P_{eye}$
- La division : $P_{NDC} \stackrel{\div w}{=} P_{clip}$, w étant la quatrième composante du vecteur P_{clip}.
- Viewport : $P_{window} = M_{viewport} P_{NDC}$

Si nous rassemblons les trois matrices, nous obtenons la matrice de transformation :

$$\pi = M_{viewport} M_{projection} M_{modelview}.$$

Ainsi le *3D warping* se résume à une projection inverse de l'image de référence (*un-projection*) suivie d'une nouvelle projection (*reprojection*) pour obtenir l'image de destination (Figure 4.3.2). Ainsi le 3D warping s'exprime par l'équation suivante [12] :

$$P^{dst}_{écran}(x', y', depth') \stackrel{\div w}{=} \pi_{dst} * \pi^{-1}_{src} * P^{src}_{écran}(x, y, depth).$$

où :
π_{src} : matrice de projection utilisée pour générer l'image de référence ,
π_{dst} : matrice de projection utilisée pour générer l'image de distination.

L'API OpenGL propose une fonction de projection inverse appelée ***gluUnproject*** [10] qui calcule les coordonnées 3D $(x_{réel}, y_{réel}, z_{réel})$ d'un point 3D à partir d'un pixel (x,y, depth), et une fonction de *reprojection* ***gluProject*** qui reprojette ce point 3D dans la nouvelle image. Malheureusement ces deux fonctions s'exécutent sur le CPU et sont très couteuses.

Zhongding et al. [12] ont proposé une méthode pour effectuer le *3D warping* sur GPU. Elle consiste à utiliser un Program Shader pour transformer les sommets d'un maillage représentant les pixels de l'image de référence en utilisant les matrices de transformation décrites ci-dessus.

Nous nous sommes inspirés de cette méthode et nous l'avons adaptée à notre problème :

- Rendre l'image de référence, *offscreen*, pour récupérer une texture contenant la couleur et une autre texture contenant la profondeur.
- Construire un maillage dense où chaque pixel est représenté par un sommet (figure 4.3.3).
- Rendre le maillage en activant un *Vertex Shader* qui va effectuer le *3D warping* de chaque sommet du maillage, et un *Fragment Shader* pour appliquer la couleur finale contenue dans l'image de référence (la texture de couleur).

Quelques détails techniques sont nécessaires pour comprendre le principe. La figure 4.3.3 (issue de [7]) donne une idée générale sur le mécanisme utilisé. Les sommets du maillage sont positionnés aux centres des pixels, la troisième coordonnée z (depth) est la valeur de la texture de profondeur en cette position. Dans le *Vertex Shader*, nous calculons les nouvelles coordonnées 3D du sommet (*un-projection*), suivi d'une *reprojection* suivant le nouveau point de vue voulu (multiplication par la matrice de transformation *gl_ModelViewProjectionMatrix* du nouveau point de vue). Dans le *Fragment Shader*, une simple lecture dans la texture de couleur (image de référence) permet de trouver la couleur finale du fragment.

En pratique, nous chosissons la vue centrale (du milieu, figure 4.3.4) comme vue de référence alors que les autres vues sont calculées (*warped*). Les images calculées présentent quelques artefacts qui s'amplifient en s'éloignant de la vue de référence (milieu). Ces artefacts sont situés principalement autour des contours (voir section 4.5). Le problème des zones cachées n'apparait pas beaucoup parce que les points de vue sont proches les uns des autres.

Le fait de ne pas utiliser un filtrage spécifique à l'écran autostéréoscopique donne un caractère générique à la méthode à l'inverse de la méthode de Berretty qui doit intégrer un filtre spécifique à l'écran. Aussi le fait

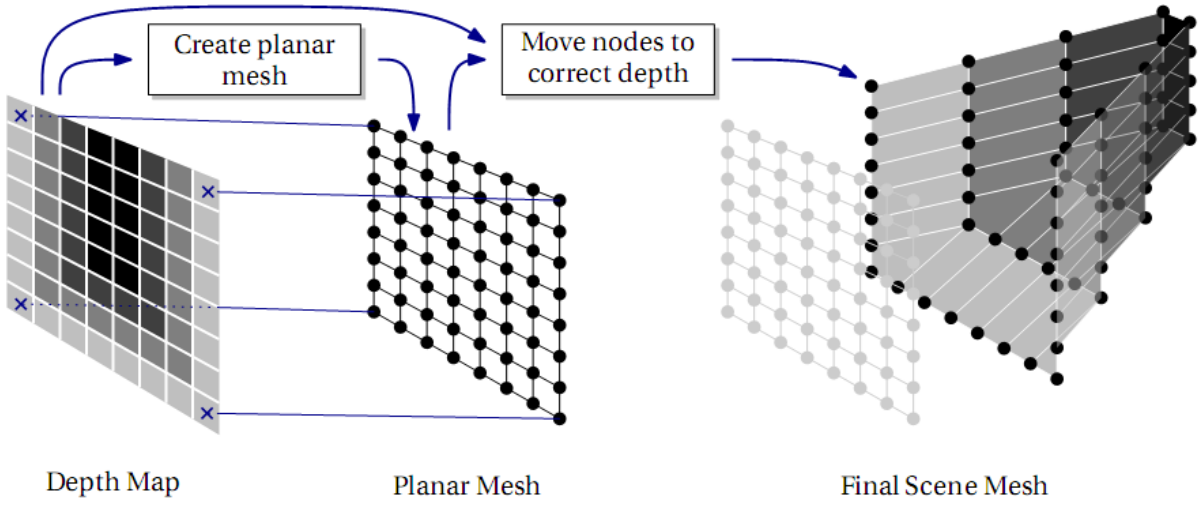

FIGURE 4.3.3: Le maillage dense

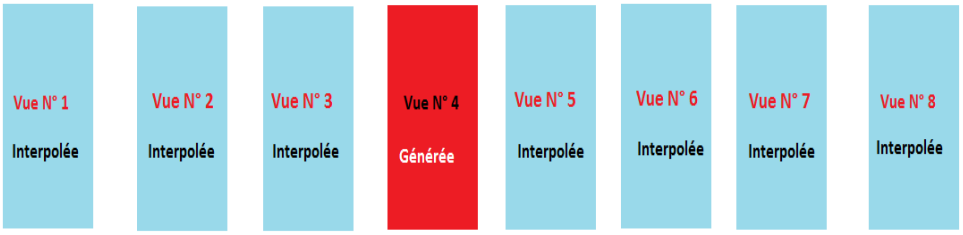

FIGURE 4.3.4: La configuration basée interpolation

de rendre un maillage dense permet de faire des interpolations de bonne qualité (zones loin des contours). Ceci est dû à la rastérisation très optimisée implémentée sur la carte graphique.

En conclusion, cette méthode est pratique dans le cas où la performance prime par rapport à la qualité. À noter que la dégradation en qualité peut être améliorée par un post-filtrage traitant les contours.

4.3.2 La méthode de composition

Dans la section 3.2, nous avons abordé l'approche développée par Mark [13, 15] pour générer des images intermédiaires à partir de deux images de référence. Cette méthode repose sur un traitement d'image 3D dans la mesure où elle prend en compte la profondeur de chaque pixel. Mark a aussi proposé une technique pour gérer les problèmes liés aux pixels appartenant aux contours des surfaces.

Néanmoins, toutes les techniques proposées par Mark sont à exécuter par le CPU. En outre, tous les traitements sont effectués pixel par pixel, ce qui rend l'exécution coûteuse dès que la résolution des images dépasse un certain seuil. Un autre inconvénient est que cette méthode n'exploite pas les fonctionnalités offertes par le matériel graphique, à savoir le parallélisme inhérent à ce type de matériel ainsi que les différents

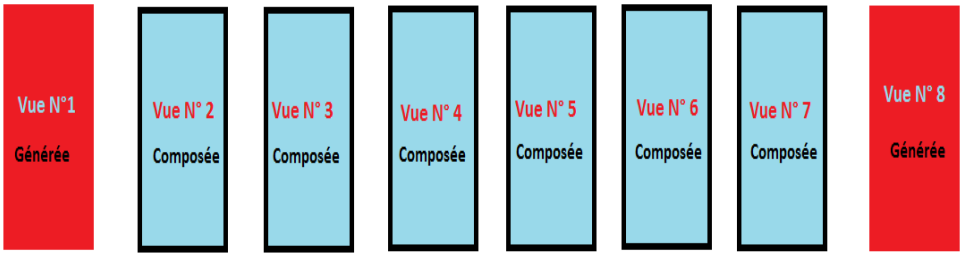

FIGURE 4.3.5: La configuration basée composition

modes de filtrage déjà câblés. C'est pourquoi notre approche consiste à adapter la méthode proposée par Mark pour une exécution sur GPU.

Dans notre cas, nous avons travaillé sur un écran autostéréoscopique à 8 vues. L'adaptation de la méthode de Mark consiste à générer deux vues utilisant le rendu classique, ensuite générer les autres vues par un rendu basé image. Le choix de la position des deux vues de référence est important pour faire face au problème de découvrement (l'apparition des surfaces cachées lors du passage d'un point de vue à un autre). Une première idée est de générer les deux vues extrêmes (Figure 4.3.5), ensuite calculer les autres images intermédiaires en utilisant la méthode de Mark (Figure 3.2.2) selon l'algorithme suivant :

- Générer, *offscreen*, les deux vues extrêmes et sauvegarder leurs couleur et profondeur dans des textures.
- Pour chaque vue intermédiaire :
  - Reprojeter les images (textures couleurs) des deux extrémités (3D warping) par rapport au nouveau point de vue, pour produire deux nouvelles textures de couleur et deux nouvelles textures de profondeur.
  - Composer les deux textures de couleur pour obtenir l'image finale selon le test suivant : pour chaque pixel de l'image composée prendre le pixel de l'image reprojetée ayant la plus petite profondeur (même principe que le Z-test classique).
- Entrelacer les 8 vues pour produire l'image finale affichable sur notre écran autostéréoscopique.

Pour l'étape de reprojection, nous avons utilisé la méthode basés GPU expliquée dans la section 4.3.1. La composition se fait par l'intermédiaire de l'affichage d'un simple quadrilatère qui couvre toute la zone d'affichage d'un FBO (la composition est offscreen). L'idée est l'utilisation de la fonctionalité multitexture de OpenGL. Pendant le rendu un Fragment Shader va accéder aux deux textures de profondeur pour effectuer un « pseudo Z-test » (choisir le pixel ayant la plus petite profondeur). Le résultat de ce test détermine quelle texture de couleur sera utilisée pour colorer le pixel courant. À la fin du rendu de ce quadrilatère, nous récupérons la texture contenant l'image finale composée, cette dernière sera utilisée plus tard pour l'entrelacement des vues afin de produire l'image finale affichable sur l'écran autostéréoscopique.

Cette méthode donne de meilleurs résultats en termes de qualité par rapport à la méthode utilisant seulement le *3D warping,* mais d'autres améliorations de la qualité des images produites sont possibles grâce à un choix judicieux de la position des images de référence.

Malgré l'utilisation de deux images de référence, plusieurs artefacts peuvent apparaître. Ceci est dû à l'absence d'un mécanisme de gestion des contours des objets dans l'image. Une solution a été proposée par Mark [15] pour la gestion des pixels des contours. Nous avons jugé sa solution comme étant coûteuse car elle

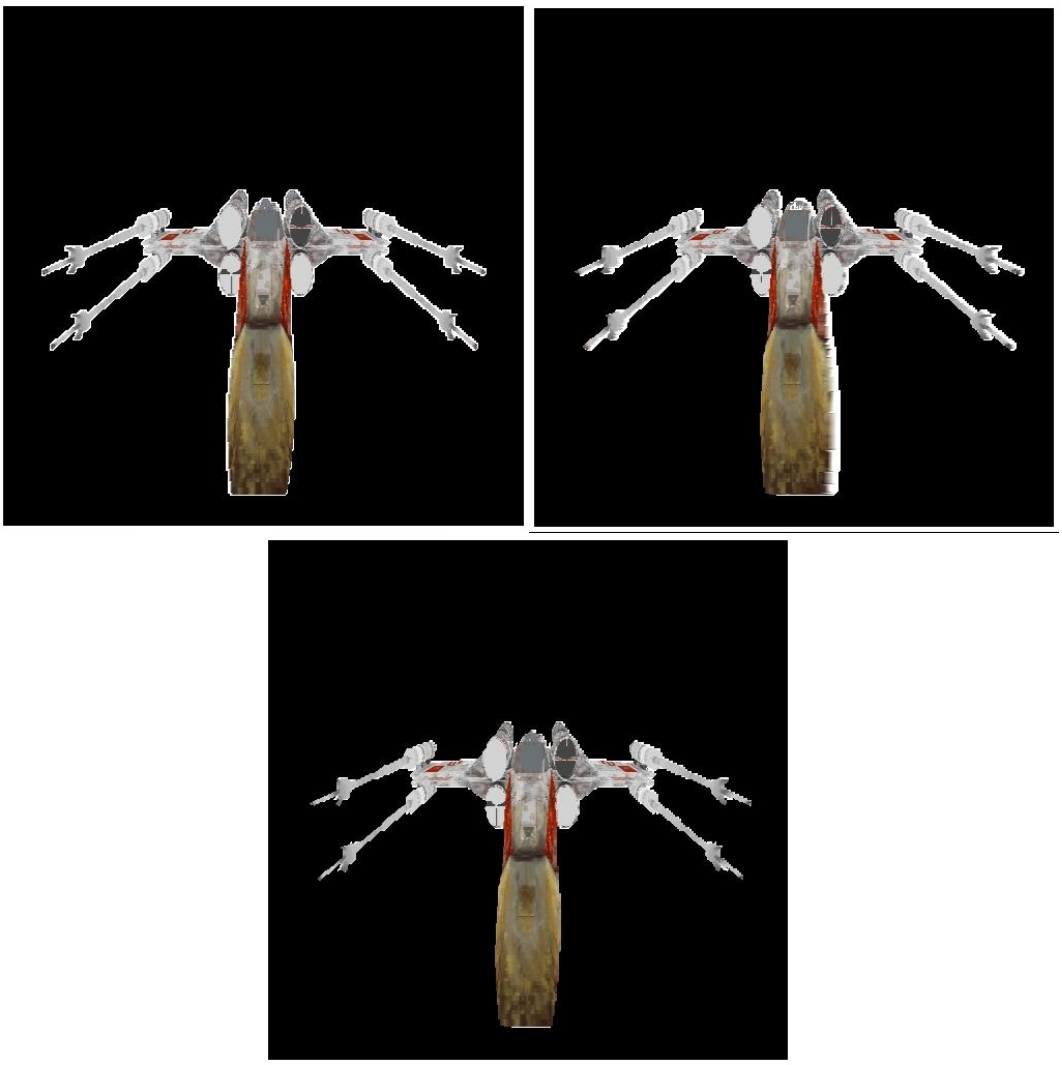

FIGURE 4.3.6: Composition de la vue 2 à partir des vues de références 1 et 8

en haut à gauche la vue de référence 1 transformée (warped),à droite la vue de référence 8 transformée, en bas la vue 2 composée

nécessite des comparaisons deux à deux des pixels pour déterminer les pixels de contour suivi d'un traitement spécial pour ces derniers. Malheureusement ce traitement, s'exécutant sur CPU, rend le rendu très coûteux surtout dans le cas d'images de haute résolution. De bons résultats peuvent être obtenus en augmentant la résolution des images de référence (suréchantillonnage).

D'autres artefacts ont pour origine l'opération de composition elle même. Prenons l'exemple de la vue 2 de la figure 4.3.5. La génération de cette vue consiste à composer les deux images résultant du *warping* des images de référence 1 et 8. L'image transformée (warped) correspondant à la vue de référence 8 contient plusieurs artefacts car les caméras 2 et 8 sont distantes l'une de l'autre (Figure 4.3.6). D'où la nécessité de proposer d'autres configurations qui tiennent compte de cette remarque.

4.3.3 La version CPU des deux méthodes : 3D warping et composition

Nous avons implémenté les deux méthodes décrites dans les sections 4.3.1 et 4.3.2 mais cette fois en utilisant un *3D warping* s'exécutant sur CPU. Autrement dit tous les traitements sont effectués pixel par pixel.

Pour chaque pixel de l'image de référence nous faisons appel à la fonction ***gluUnproject*** pour obtenir le point 3D correspondant, ensuite nous appelons la fonction ***gluProject*** pour reprojeter ce point selon le nouveau point de vue.

Nous avons obtenu des résutats très médiocres en termes de performance. En termes de qualité, l'abscence d'un mécanisme de filtrage laisse apparaitre quelques zones non traitées dues au *3D warping*. Le sur-échantillonage peut améliorer la qualité mais les performances chutent très rapidement dés que la résolution dépasse le seuil de 128x128.

4.4 La méthode proposée : Configuration mixte interpolation/composition

Dans la section précédente nous avons décrit une méthode consistant en une composisition des images intermédiaires à partir des images extrêmes prises comme images de référence. Cette méthode donne de bons résultats en termes de qualité mais nous avons remarqué l'apparition d'artefacts liés à l'interpolation des images correspondant aux points de vue éloignés des deux images de référence. Dans la figure 4.3.5, les vues 2 et 7 contiennent plusieurs artefacts car elles sont éloignées des vues de référence. Une solution possible pour diminuer ces effets désagréables consiste à adopter la configuration de la figure 4.5.8

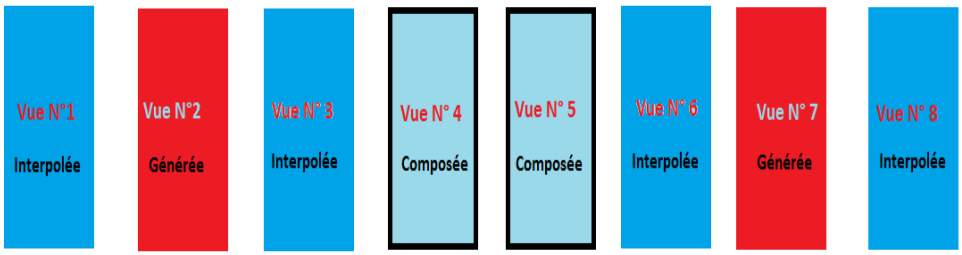

FIGURE 4.4.1: La configuration mixte

Dans cette configuration, nous avons choisi comme images de référence les deux images correspondant aux points de vue 2 et 7. Les deux images à gauche et à droite de chaque image de référence seront calculées par une simple interpolation (3D warping) décrite dans la section 4.3.1. Le choix de l'interpolation comme méthode de génération est justifié par le fait que leurs positions sont très proches de celle des images de référence. Les images restantes seront calculées par la méthode de composition à cause de leur position un peu éloignée par rapport à celles des images de référence.

Autres configurations

Pour diminuer les artefacts dus à l'interpolation et à la composition, et dans le cas où la contrainte qualité est exigée, nous pouvons adopter la configuration de la figure 4.4.2.

Dans cette configuration trois images de référence sont générées au lieu de deux comme dans les configurations précédentes. Nous générons par exemple les vues 1, 2 et 4 . Les vues restantes sont somposées à

partir des vues générées (Les vues 2 et 3 sont composée à partir des vues 1 et 4, les vues 5, 6 et 7 à partir des vues 4 et 8).

Cette configuration donne de très bons résultats en termes de qualité mais au détriment d'une baisse en performance due à la génération d'une troisième vue.

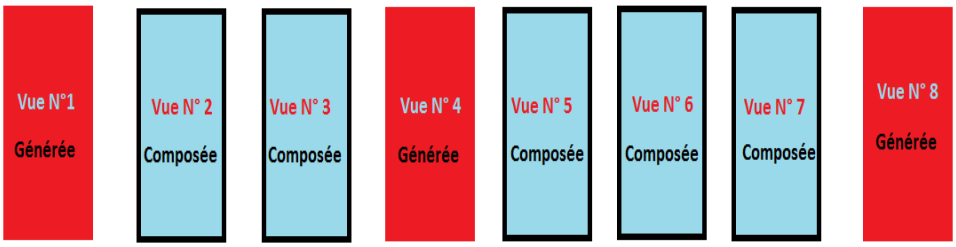

FIGURE 4.4.2: La configuration orientée qualité

D'autres configurations possibles sont à imaginer à l'instar de la figure 4.4.3. Dans cette configuration nous générons trois vues, les vues voisines sont ensuite simplement interpolées à partir de la vue de référence la plus proche. La vue 6 sera composée (car elle est un peu loin des deux vues de référence 4 et 8).

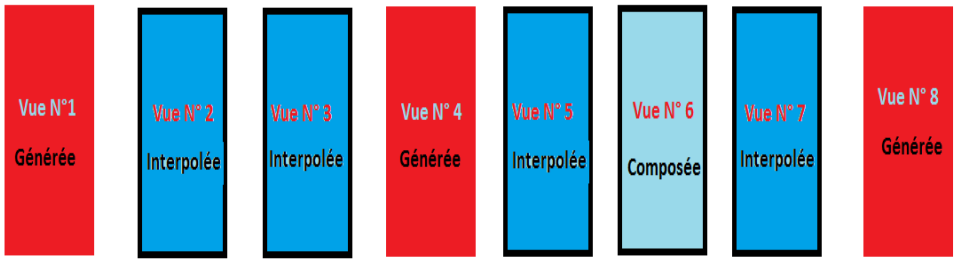

FIGURE 4.4.3: Autre configuration

4.5 Discussion et commentaires

Dans cette section nous allons montrer quelques résultats et comparaisons des différentes méthodes implémentées durant ce stage de Master. Nous terminerons cette section par un ensemble de recommandations pour le choix de la bonne méthode de rendu selon nos besoins en termes de qualité et de performance.

Pour comparer les méthodes, nous avons choisi le critère FPS (Frame Per Second) pour évaluer les performances. Pour évaluer la qualité des images produites, nous basons sur les images de différence. Notre scène test est la scène 3D de la figure 4.5.1 contenant 12564 sommets rassemblés en 20736 facettes triangulaires.

4.5.1 La qualité

L'évaluation de la qualité des méthodes est basé sur l'observation de l'image de différence, image représentant la différence (vectorielle) pixel par pixel entre l'image produite par les méthodes de rendu basé

FIGURE 4.5.1: La scène 3D de test

image et celle produite par la méthode de rendu classique (rendu direct à partir du modèle 3D).

Le rendu classique

La méthode multipasse utilisant le rendu classique, décrite dans la section 4.2.1, donne les meilleurs résultats en termes de qualité. Ceci était prévisible car dans cette méthode les images intermédiaires sont synthétisées directement à partir du modèle 3D de la scène.

La méthode utilisant le *Geometry Shader,* bien qu'elle utilise le rendu classique, présente des artefacts dus à l'absence d'un mécanisme de test de profondeur que nous avons jugé inutile de mettre en œuvre car les performances de cette méthode sont déjà faibles. L'implémentation d'un tel mécanisme chuterait encore plus ses performances.

Le *3D warping*

La figure 4.5.2 montre le résultat du calcul (3D warping) des deux vues voisines (la première à gauche et la première à droite) de la vue centrale générée par un rendu 3D classique. Nous avons changé le fond de l'image pour pouvoir détecter les artefacts dus à l'interpolation surtout au niveau des contours. Nous remarquons bien l'apparition d'une petite bande blanche autour du contour de l'objet dans les deux vues interpolées. Cette bande devient importante dans la figure 4.5.3 où les vues à interpoler sont aux extrémités gauche et droite.

La figure 4.5.4 représente l'image de différence entre la vue produite par le *3D warping* et la même vue produite par un rendu classique. Nous remarquons que les contours sont la source majeure des artefacts. Si

FIGURE 4.5.2: Calcul de deux vues proches de la vue centrale utilisant le 3D warping

FIGURE 4.5.3: Calcul des deux vues des extrimités utiliant la vue centrale utilisant le 3D warping

nous effectuons un zoom, d'autre erreurs apparaissent dans les zones qui sont loin des contours. Ceci est dû à l'utilisation d'une seule image de référence et aussi à la distance relativement importante entre les deux vues (la vue de référence au milieu et la vue extrême à calculer).

La composition

Dans la figure 4.5.5, nous avons obtenu de bons résultats en calculant l'image centrale par composition des images extrêmes générées par un rendu 3D classique. Dans d'autres situations, des artefacts peuvent apparaître dans des vues qui sont plus proches d'une extrémité que de l'autre (Figure 4.5.7). Ceci est dû à l'étape du 3D warping qui a lieu avant la composition (la composition concerne les images interpolées, voir la section 4.3.2 pour plus de détails).

Les images de différence (Figure 4.5.7 et 4.5.6) montrent clairement les artefacts autours des contours. Néanmoins les erreurs dans les zones intérieures sont minimes.

La méthode mixte

La méthode mixte, qui est un amélioration de la méthode de composition, réduit légèrement les artefacts autour des contours (Figure 4.5.8). Néanmoins un traitement plus approprié pour les contours demeure nécessaire.

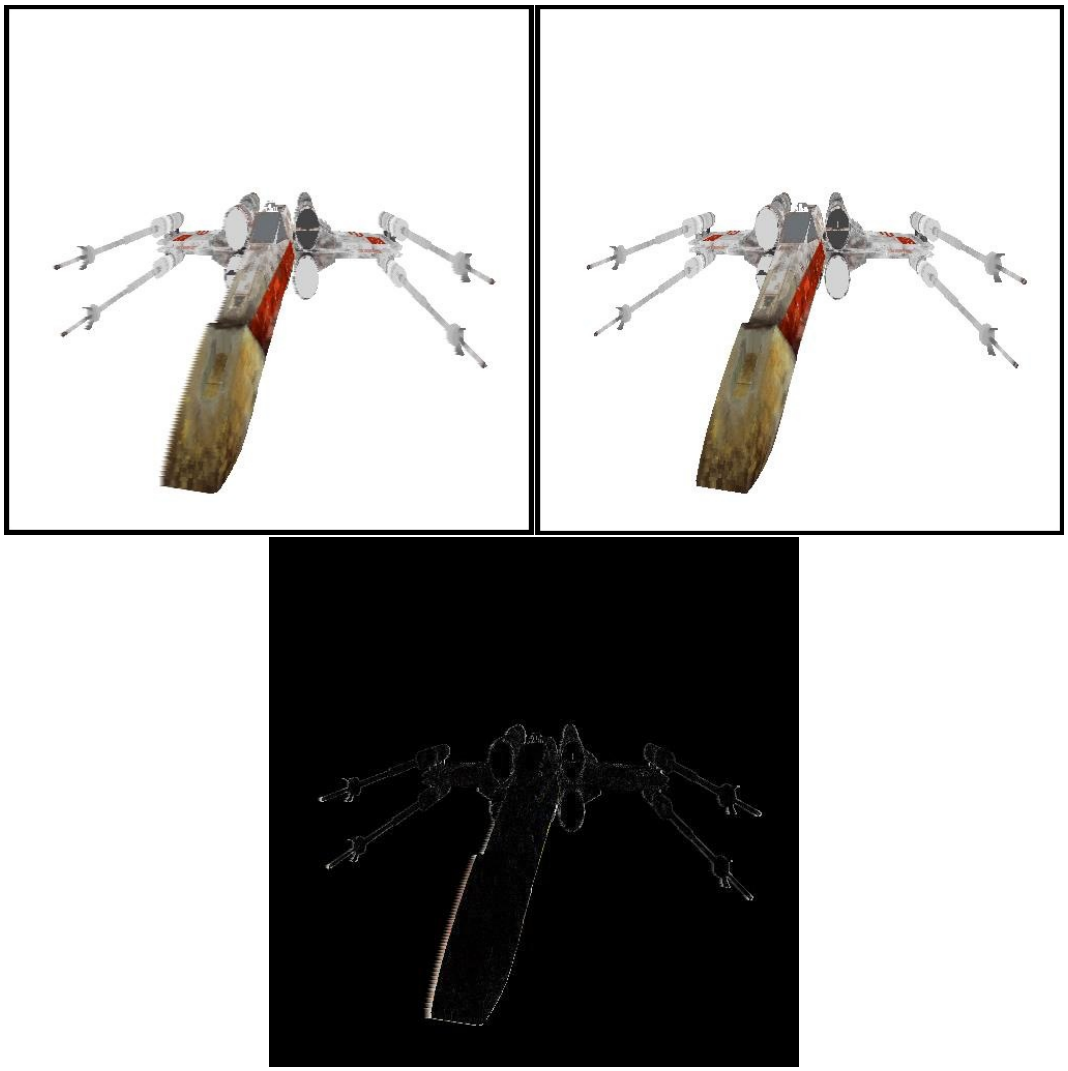

FIGURE 4.5.4: Image de différence des deux images repésentant la vue de l'extrême gauche

en haut à gauche la vue de l'extrême gauche produite par le 3D warping, en haut à droite la même vue produite par la méthode du rendu classique et en bas l'image de différence.

Le traitement de l'*aliasing*

Dans toutes les méthodes discutées ci-dessus, nous avons utilisé deux techniques de réduction d'*aliasing* :

- Le *multisampling* : technique offerte par la carte graphique qui diminue l'*aliasing* dans les images intermédiaires. Nous avons pu exploiter cette technique dans les méthodes de rendu basé image adaptées à une exécution sur GPU (*aliasing* dû au *3D warping*).
- Le sur-échantillonnage : nous avons généré des images intermédiaires ayant une résolution plus grande que celle nécessaire à l'entrelacement. (*aliasing* dû à l'entrelacement). Faute de temps, nous n'avons pas pu intégrer un filtre passe-bas pour éliminer les hautes fréquences sources principales d'*aliasing.*

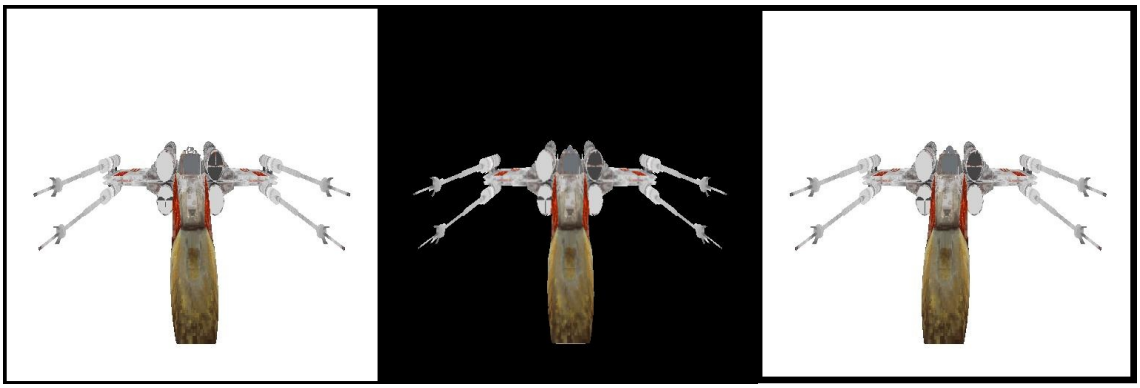

FIGURE 4.5.5: Composition de la vue centrale (vue 4) utilisant les vues extrêmes (vue 1 et vue 8)

Rendu	FPS
Rendu non stéréo	260
Stéréo 8 passes	99
Stéréo MRT (Geometry Shader)	11
3D warping GPU	62
Composition GPU	51
La méthode mixte	56
3D warping CPU	1
Composition CPU	1

TABLE 4.1: Tableau récapitulatif des performances

Nous pouvons encore réduire l'*aliasing* par l'utilisation de filtres :

- après la génération des images intermédiaires pour réduire encore l'*aliasing* dû au *3D warping*.
- après la génération de l'image finale pour réduire l'*aliasing* dû à l'entrelacement (filtre spécifique selon la grille de l'écran autostéréoscopique).

Faute de temps nous n'avons pas pu intégrer ces filtres. Néanmoins, les deux techniques que nous avons utilisées (le *multisampling* et le suréchantillonnage) donnent de bons résultats en termes de qualité.

4.5.2 Les performances

Le tableau 4.1 résume les résultats en termes de performance des différentes méthodes citées ci-dessus.

- La méthode multipasse présente les meilleures performances. Ceci est dû au fait que la scène 3D utilisée est plus ou moins simple et d'autre part parce que la méthode multipasse n'utilise pratiquement que des fonctions cablées très optimisées. Néanmoins, les performances de cette méthode sont inversement proportionnelles à la complexité de la scène.
- Le méthode utilisant le *Geometry Shader* présente des résultats peu satisfaisants. Nous pensons que notre carte graphique (NVIDIA Quadro FX 1800) ne dispose pas d'assez de processeurs pour l'exécution des *Geometry Shader.*
- La méthode du *3D warping* donne des résultats plus performants que ceux de la méthode de composition. Ceci était prévisible car la méthode de composition contient plus d'étapes de traitement que la

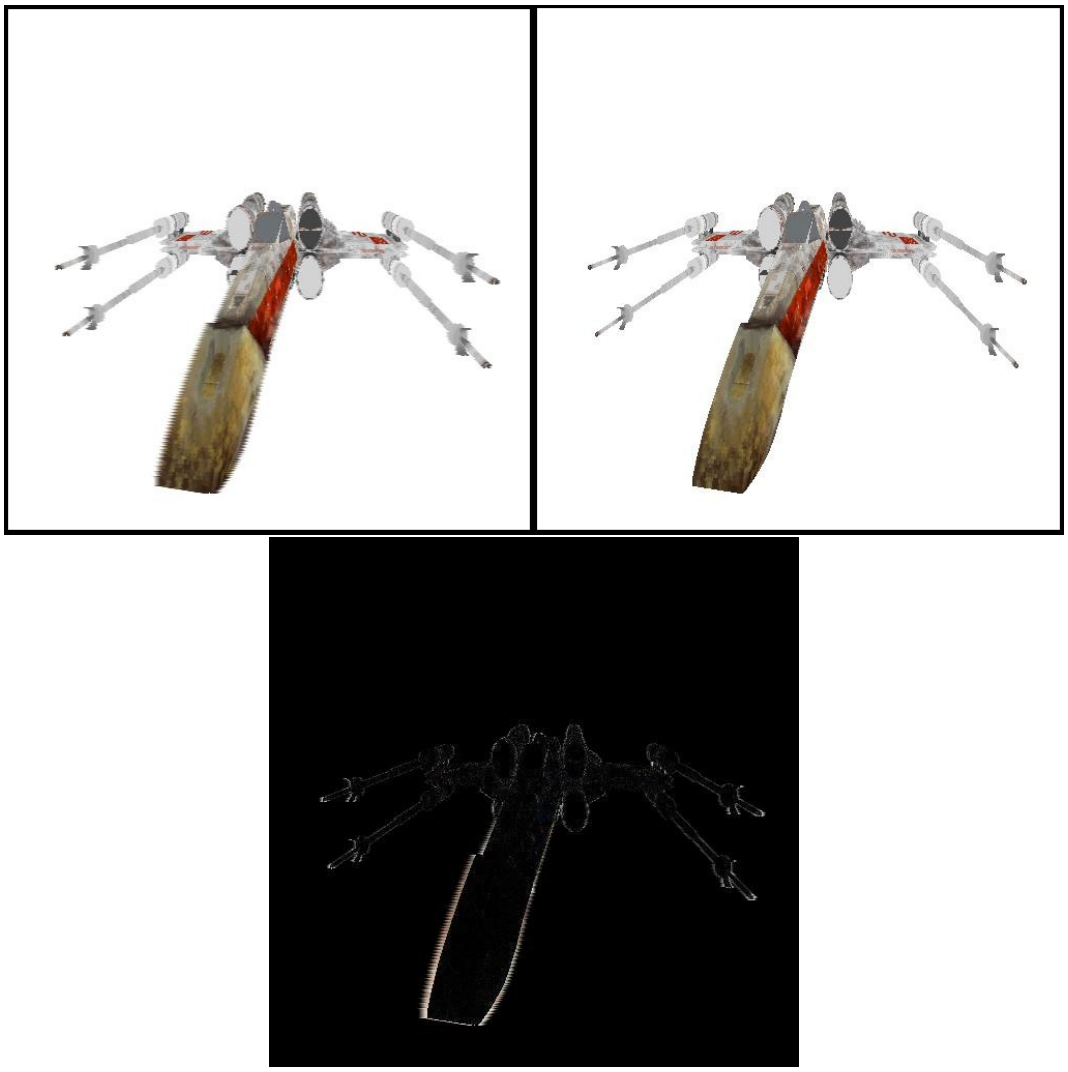

FIGURE 4.5.6: Image de différence des deux images repésentant la vue centrale

en haut à gauche la vue centrale (vue 4) produite par la composition des deux vues de référence 1 et 8 (vues extrêmes), en haut à droite la même vue produite par la méthode du rendu classique et en bas l'image de différence.

méthode du *3D warping*. Mais l'avantage de ces deux méthodes est qu'elles sont globalement indépendantes de la complexité de la scène à rendre .

- La méthode mixte (interpolation/composition) a donné des performances situées entre celles de la méthode *3D warping* et celles de la composition. Ceci était prévisible car elle combine les deux méthodes.
- L'implémentation sur CPU des deux méthodes (*3D warping* et la composition) est pratiquement inefficace car elle présente de très faibles performances pour les résolutions moyennes (512x512). Si nous diminuons la résolution, les performances s'améliorent un peu mais la qualité devient vite inacceptable (entre 4 et 8 FPS pour une résolution de 128x128).

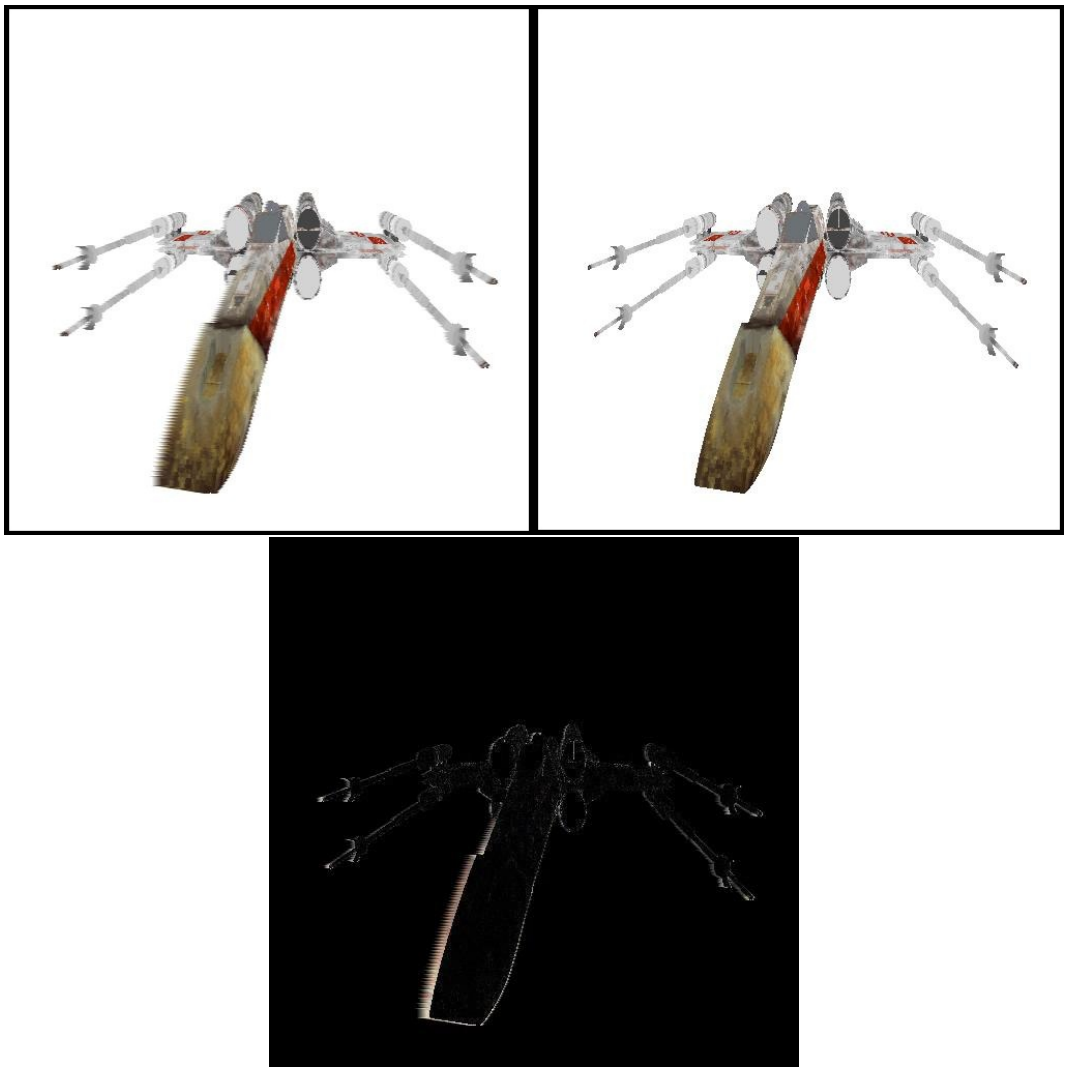

FIGURE 4.5.7: Image de différence des deux images repésentant la vue 2

en haut à gauche la vue 2 produite par la composition des deux vues de référence 1 et 8, en haut à droite la même vue produite par la méthode du rendu classique et en bas l'image de différence.

4.5.3 Recommandations

L'intérêt des méthodes basées sur un traitement d'image (*3D warping* et composition) est qu'en général elles ne dépendent pas de la complexité de la scène. En fait, seule l'étape de génération des images de référence nécessite un rendu 3D. La génération des images intermédiaires est entièrement indépendante de la complexité de la scène.

Ces méthodes sont idéales dans une configuration client-serveur où les images de référence sont générées au niveau du serveur (généralement assez puissant) et ensuite elles seront transmises au client pour calculer les images intermédiaires restantes. Cas idéal pour la télévision 3D, mais nécessitant des terminaux TV

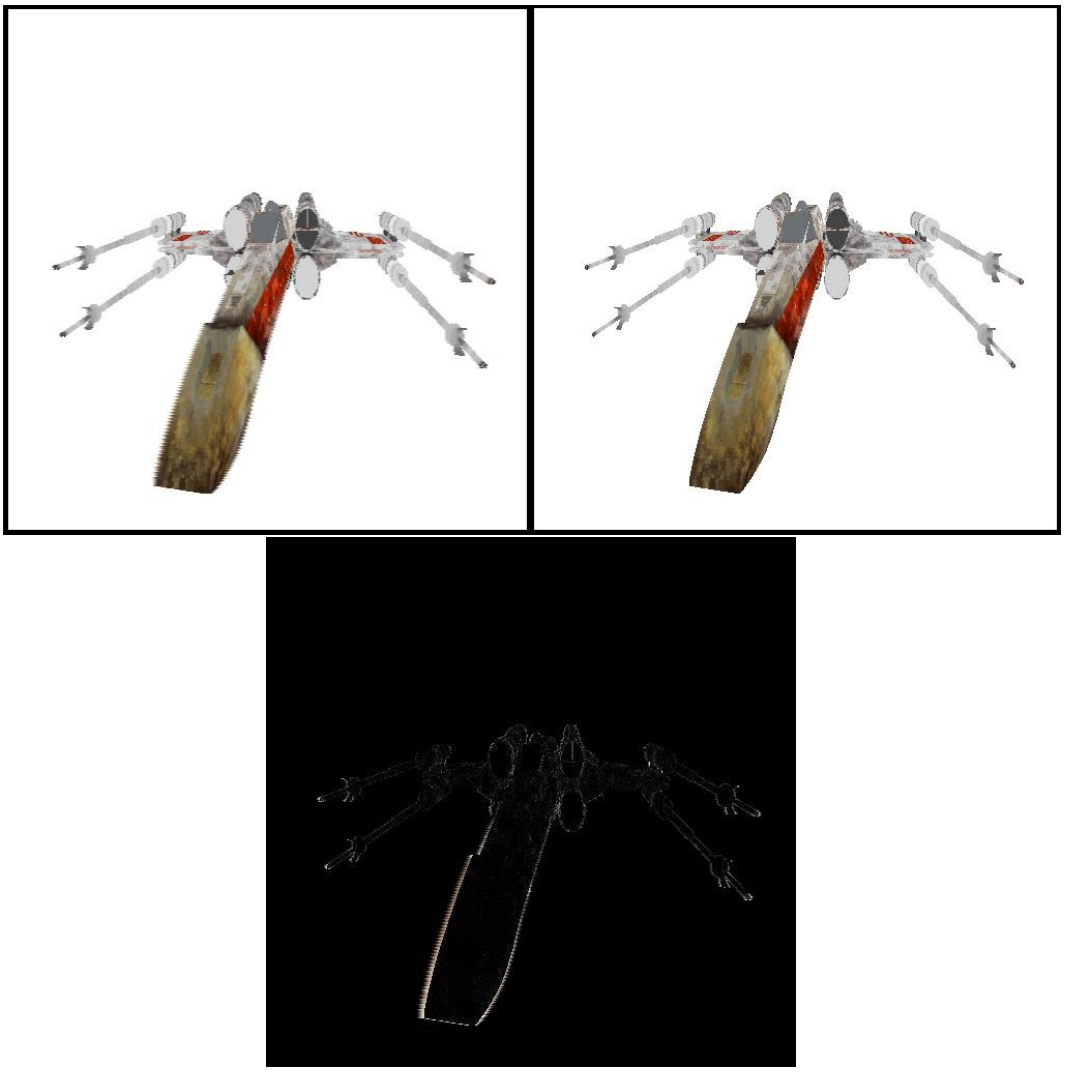

FIGURE 4.5.8: Image de différence des deux images repésentant la vue centrale

en haut à gauche la vue 4 produite par la méthode mixte, en haut à droite la même vue produite par la méthode du rendu classique et en bas l'image de différence.

munis de circuits dédiés implémentant les méthodes de rendu basé image décrites auparavant.

Dans le cas de scènes peu complexes, la méthode multipasse est plutôt à envisager surtout pour une exécution sur un PC muni d'une carte graphique puissante ou bien sur une console de jeu moderne (généralement très puissante).

La méthode mixte, à notre avis, est le meilleur compromis entre une exécution rapide et une très bonne qualité. Le problème des artefacts aux contours dû à la discontinuité dans la texture (la carte) de profondeur nécessite un traitement additionnel pour assurer une bonne qualité.

A noter que non seulement la complexité de la scène mais aussi la méthodes du rendu qui influent sur les performances de toutes approches : rendu photo-réaliste, ray-tracing, shaders procéduraux, etc.

Chapitre 5

Conclusion et perspectives

Dans ce rapport nous avons fait un tour d'horizon sur les différents concepts qui ont été utilisés dans mon stage, à savoir les écrans autostéréoscopiques, le rendu basé image adapté à ce type d'écran et enfin les différents problèmes pouvant affecter la qualité des images produites (les artefacts et l'aliasing). Les travaux de Jiang [12], de Berretty [2] et Mark [13, 15] ont constitué une base pour notre approche développée durant mon stage pour ce qui concerne le rendu basé image.

Nous avons proposé plusieurs méthodes pour améliorer la qualité des images produites et les performances du rendu par rapport aux autres méthodes proposées auparavant.

Malgré les bons résultats obtenus, plusieurs problèmes restent à résoudre surtout ceux d'artefacts qui apparaissent au niveau des contours. Ce qui ouvre des perspectives de recherche de bonnes solutions améliorant la qualité et assurant de bonnes performances.

Par manque de temps, un point important demeure non traité, à savoir l'intégration de filtres *antialiasing*. En effet, deux filtres *antialiasing* sont à appliquer. Un premier filtre doit être appliqué aux vues intermédiaires pendant ou après leur génération (réduire l'aliasing dû au *3D warping*). Un deuxième filtre doit être utilisé pendant où après la génération de l'image entrelacée (réduire aliasing dû à l'entrelacement des vues intermédiaire).

Un autre point non traité dans ce stage est la cohérence temporelle, concept important pour optimiser au mieux les performances dans le cas de la télévision 3D où les images sont générées et transmises à distance.

D'autre tests utilisant des scènes plus complexes (plusieurs objets, un fond avec plus de détails, de méthodes de rendu photoréalistes, etc.) sont à effectuer pour se rendre compte des limites réelles de nos méthodes.

Enfin, nous sommes convaincus que nous n'avons pas encore exploité toutes les fonctionnalités offertes par le matériel graphique ce qui laisse penser à d'autres possibilités pour améliorer encore la qualité et les performances du rendu autostéréoscopique.

Bibliographie

[1] B. Barenbrug, R-P. M. Berretty, and R. Klein Gunnewiek. Robust image, depth, and occlusion generation from uncalibrated stereo. volume 6803, page 68031J. SPIE, 2008.

[2] R.-P. M. Berretty, F. J. Peters, and G. T. G. Volleberg. Real-time rendering for multiview autostereoscopic displays. volume 6055. SPIE, 2006.

[3] Johan Compen. 3d graphics rendering for multiview displays. Master's thesis, TECHNISCHE UNIVERSITEIT EINDHOVEN, 2005.

[4] François de Sorbier, Vincent Nozik, and Biri venceslas. Gpu rendering for autostereoscopic displays. *Proceedings of 3DPVT'08- the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, June 18-20, 2008.

[5] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steve Worley. *Texturing and Modeling, Third Edition : A Procedural Approach (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 3 edition, December 2002.

[6] Hideo Saito François de sorbier, Vincent Nozick. Gpu-based multi-view rendering. page 7, april 2010.

[7] Sourimant Gaël. Depth maps estimation and use for 3dtv. Technical report, INRIA de Rennes, N° 0379, February 2010.

[8] R. Klein Gunnewiek, R.-P. M. Berretty, B. Barenbrug, and J. P. Magalh. Coherent spatial and temporal occlusion generation. volume 7237. SPIE, 2009.

[9] Paul S. Heckbert and C Fl Paul S. Heckbert. Fundamentals of texture mapping and image warping. Technical report, 1989.

[10] Silicon Graphics Inc. Opengl 2.1 reference pages.

[11] Randi J. Rost and Bill Licea-Kane. *OpenGL Shading Language Third Edition*. Addison-Wesley, 2009.

[12] Zhongding Jiang, Tien-Tsin Wong, and Hujun Bao. Three-dimensional image warping on programmable graphics hardware. Technical report, Zhejiang University, Hangzhou, 2003.

[13] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *I3D '97 : Proceedings of the 1997 symposium on Interactive 3D graphics*, New York, NY, USA, 1997. ACM.

[14] Koen Meinds and Bart Barenbrug. Resample hardware for 3d graphics. *HWWS '02 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 17–26, 2002.

[15] William R. Mark. *Post-Rendering 3D Image Warping : Visibility, Reconstruction and performance for Depth-Image Warping*. PhD thesis, University of North Carolina, 1999.

[16] Dave Shreiner. *OpenGL Programming Guide Seventh Edition*. Addison-Wesley, 2009.

[17] Ireneusz Tobor, Patrick Reuter, Laurent Grisoni, and Christophe Schlick. Visualisation par surfels. Technical report, LaBRI, Université de Bordeaux1, France, 2000.

[18] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.