Rapport final

PFA - De la 3D vers la 2D

Année scolaire 2014-2015

Client : BLANC Carole, DESBARATS Pascal

Encadrant : LOMBARDY Sylvain

Equipe : BOHER Anaïs - CABON Yohann - CHAUVAT Magali
LEVY Akané - MARCELIN Thomas
MAUPEU Xavier - PHILIPPI Alexandre

Table des matières

1 Introduction

Le projet PFA a pour objectif de permettre aux élèves la gestion d'un projet dans sa totalité, depuis la création du cahier des charges en réponse à une demande de clients jusqu'à l'implémentation du projet visé. Il aura ainsi permis de travailler sur le cahier des charges d'un projet, ce qui est un exercice constructif auquel nous n'avions jamais encore eu à faire, mais également de travailler dans une équipe conséquente de 7 personnes quand la plupart des projets réalisés dans le cadre de notre cursus se font par équipe de 3 à 4 personnes.

Le projet PFA effectué est un projet d'imagerie numérique. Son but est de travailler à l'aide d'un logiciel dans une scène virtuelle en trois dimensions, et d'obtenir grâce à cette dernière des images en deux dimensions permettant de recréer une illusion de trois dimensions. Les algorithmes qui auront notamment été étudiés au cours de ce projet permettent l'obtention d'anaglyphes, d'autostéréogrammes, ou encore de folioscopes. L'imagerie numérique n'est pas étudiée au cours des deux premières années d'Informatique à l'Enseirb-Matmeca. Ainsi, ce projet constituait une réelle ouverture d'esprit sur un nouveau domaine de l'informatique.

Dans ce rapport, nous présenterons dans un premier temps le domaine de la synthèse d'image, les rendus à implémenter dans notre logiciel et les objectifs donnés dans le cahier des charges. Nous parlerons ensuite de la réalisation technique de notre logiciel, depuis les choix effectés pour la programmation et les algorithmes jusqu'au déroulement du projet et aux difficultés rencontrées. Enfin, nous ferons un bilan de cet expérience de gestion de projet, depuis la phase de cahier des charges jusqu'au rendu.

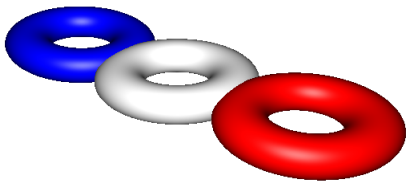

FIGURE 1.1 – Rendu classique du projet

2 Présentation du projet

Cette partie permettra de présenter le domaine de la synthèse d'image ainsi que le sujet du projet sur lequel nous avons eu à travailler. Nous présenterons également les trois rendus qui ont été implémentés pour le logiciel : l'anaglyphe, l'autostéréogramme et le folioscope.

2.1 Domaine

Ce projet s'inscrit dans le domaine de la synthèse d'images et de la visualisation de modèles en trois dimensions. Depuis le quinzième siècle, grâce à la peinture, la perspective apparaît sur des supports en deux dimensions. Aux XIXème et XXème siècles, l'utilisation de stéréoscopes, tel que le stéréoscope de Holmes, permettait la visualisation de relief à partir de deux images planes et d'un dispositif optique. Dans la deuxième moitié du XXème siècle, l'utilisation du numérique permet de modifier les images et d'obtenir une meilleure visualisation de la profondeur sur des supports en deux dimensions.

On peut ainsi créer des anaglyphes, des autostéréogrammes ou des flipbooks, qui sur papier ou sur écran permettent d'apercevoir la profondeur d'une scène grâce à des techniques adaptées. Ces différents rendus seront présentés plus tard dans ce cahier des charges. De nos jours, il existe également des logiciels, tels que Meshlab et Blender qui sont gratuits, open source et permettent d'ores et déjà la visualisation en trois dimensions sur un écran. L'utilisateur peut tourner autour d'un objet et le voir sous tous ses angles grâce à un ensemble de projections successives autour de l'objet.

On appelle synthèse d'image l'ensemble des techniques qui permettent de visualiser des objets en trois dimensions en perspective sur un écran d'ordinateur, en tenant compte de lumières et de textures appliquées à l'objet. Il existe un grand nombre de techniques et les résultats obtenus peuvent eux aussi varier (perspective isométrique, perspective conique...). Nous nous préoccupons par la suite de la perspective conique, dite aussi vue naturelle.

Bien souvent, la synthèse d'image utilise le principe de scène. Il s'agit d'un espace à trois dimensions dans lequel des objets peuvent être placés. Ces derniers sont décrits par un ensemble de points disposés dans l'espace.

Pour pouvoir observer la scène et les objets, il est nécessaire de demander à l'ordinateur de les modéliser, c'est-à-dire d'afficher un rendu qui correspondrait à une vision de cette scène si elle était réelle. Pour cela, la machine simule le point de vue de l'utilisateur à l'aide d'une « caméra ». A partir de cette scène en trois dimensions, la caméra peut réaliser des projections ou photographies permettant de créer des anaglyphes, autostéréogrammes ou flipbooks. Plusieurs méthodes de projection existent, mais seule celle par matrice de projection sera utilisée.

Ces matrices sont décrites à l'aide de coordonnées homogènes. Celles-ci ont été introduites afin que l'ensemble des transformations de type rotation, translation et homothétie puissent être écrites sous forme de matrice. Ainsi, le produit des matrices de transformation peut être calculé en amont pour pouvoir appliquer la matrice de la transformation résultante à l'ensemble des points de l'objet sans avoir à recalculer le produit pour chaque point.

Pour pouvoir visualiser un modèle 3D il faut prendre en considération la lumière et sa réflexion sur l'objet. Si une sphère rouge était représentée dans un espace avec uniquement une lumière ambiante, il n'en ressortirait qu'un disque rouge, sans relief. En effet, la lumière ambiante atteint l'objet de la même façon en tout point. On ne peut donc pas savoir depuis un plan fixe s'il s'agit d'un objet en deux ou en trois dimensions. Si maintenant une lumière est ajoutée dans l'espace où est situé l'objet, celle-ci ne va pas atteindre tous les points de l'objet de la même façon. Elle sera plus faible sur un point plus éloignée, voire inexistante sur un point caché. En tenant compte de cette lumière, on peut obtenir une image comme présentée sur la figure 2.1.

1. http ://linut.free.fr/omgspl0kuberwebloglolz0r/ ?2010/02/01/93-raytracer-que-la-lumiere-soit

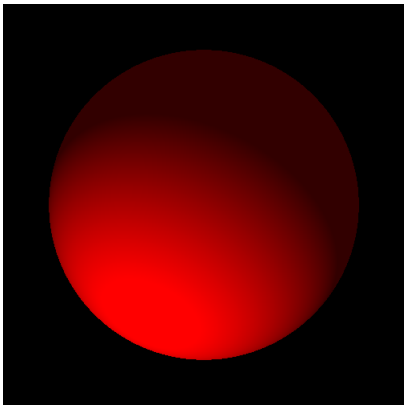

FIGURE 2.1 – Application d'une lumière diffuse à une sphère rouge [1]

Pour la création d'un anaglyphe, deux images espacées par une petite distance (qui correspond à la distance entre les deux yeux par exemple) sont générées. La composante rouge de l'une de ces images et la composante bleue de l'autre sont gardées et ensuite superposées dans une même image. Cette image est ensuite transformée en une image Rouge-Cyan, qui peut être visualisée à l'aide de lunettes Rouge-Bleue : l'image apparaît en trois dimensions.

Pour la création d'un autostéréogramme, une image permettant d'observer un objet en relief par vision parallèle est générée. Cette image est obtenue à partir d'une texture de base ou de points aléatoires pour l'image de fond.

Pour la création d'un flipbook, plusieurs images sont prises à intervalles réguliers par une caméra suivant un trajet prédéterminé dans ou autour de la scène. Le flipbook est visualisable en faisant rapidement défiler ces images tout en respectant l'ordre des prises de vue. Ce flipbook peut être transformé en GIF pour obtenir une visualisation animée des images.

2.2 Sujet du projet

Le projet concerne la réalisation d'un logiciel permettant d'obtenir des projections en deux dimensions, des anaglyphes, des autostéréogrammes ou encore des flipbooks à partir de scènes virtuelles en trois dimensions.

L'objectif premier est de permettre la visualisation, sur un support en deux dimensions tel qu'un écran d'ordinateur ou une feuille de papier, d'un espace en trois dimensions. A partir de la visualisation d'objets 3D dans une scène il faudra donc réaliser des photographies qui une fois traitées donneront lieu à des anaglyphes, autostéréogrammes ou des animations type flipbook.

Afin d'atteindre cet objectif, un logiciel s'appuyant sur le moteur 3D OpenGL devra être réalisé. Il permettra la création d'une scène où s'insèreront des objets dont la position, la taille et l'orientation seront paramétrables. Une caméra permettra de se déplacer dans la scène, de s'en rapprocher ou s'en éloigner.

Une fois la scène mise en place, il faudra pouvoir prendre des photographies de celle-ci sous différents angles afin d'obtenir, après application d'algorithmes de traitement d'images :

- des anaglyphes rouge-cyan, qui permettront une visualisation en trois dimensions grâce à des lunettes adaptées ;
- des stéréogrammes, qui sont des images dissimulant un contenu qui apparaît quand on fixe le dessin de façon spécifique ;

– des flipbooks ou images animées, correspondant à une succession d'images suivant une trajectoire qui permettent en les faisant défiler de donner une impression de mouvement.

2.3 Présentation des rendus souhaités

Nous présenterons dans cette partie les trois rendus que nos clients nous ont demandé d'implémenter dans notre logiciel : les autostéréogrammes, les folioscopes ou flipbooks, et les anaglyphes.

2.3.1 Les flipbooks

Le principe d'un flipbook, ou folioscope en français, est de créer une suite d'images successives d'une scène par rapport à une trajectoire. Il suffira ensuite de mettre toutes ces images dans l'ordre les unes derrière les autres, et de les faire défiler rapidement pour avoir l'impression d'un rendu en relief et en mouvement. C'est également le principe des fichiers d'extension .GIF, qui font défiler une liste d'images.

La création d'un flipbook est possible en créant une animation à l'aide d'un logiciel de manipulation d'objets 3D et en ne capturant que certaines images, par exemple avec Blender. Ainsi l'impression de ces images successives permet de réaliser un flipbook (cf. figure 2.2). En combinant par exemple avec le logiciel Gimp (outil d'édition et de retouche d'image) l'animation peut être obtenue en GIF.

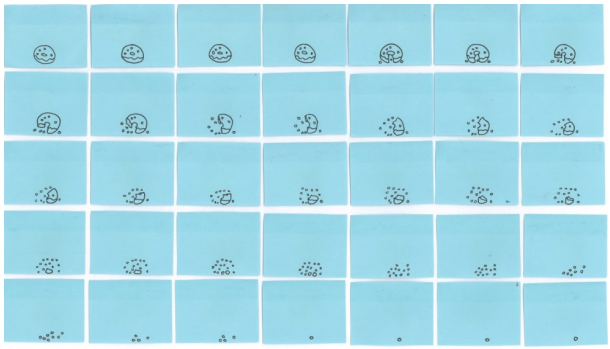

FIGURE 2.2 – Story-board d'un flipbook [2]

2.3.2 Les autostéréogrammes

Un autostéréogramme est une image qui cache une visualisation en trois dimensions d'un objet. Cette image est construite de sorte qu'à chaque point de l'objet en trois dimensions soient associés deux points de l'image. L'utilisateur doit observer chaque point d'un couple de points avec un seul œil, ce qui donne l'illusion au cerveau d'observer deux images différentes avec les deux yeux et donc l'incite à traiter cette information comme l'observation d'un objet en trois dimensions, comme illustré par la figure 2.3.

La visualisation en trois dimensions peut être difficile à obtenir, et demande une réelle gymnastique oculaire. Il faut pouvoir fixer le regard en avant ou en arrière de l'image, pour réussir à y voir l'objet caché. La plupart des autostéréogrammes sont observables en vision parallèle, c'est-à-dire qu'il faut faire le point au-delà de l'image pour pouvoir observer l'objet.

2. `http://tracieliu.blogspot.fr/2010/08/flipbook-storyboard.html`
3. `http://www.techmind.org/stereo/geometry.gif`

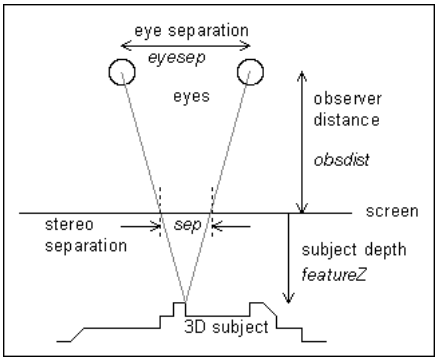

FIGURE 2.3 – Visualisation d'un autostéréogramme en vision parallèle [3]

Pour générer un autostéréogramme, il faut utiliser une carte des profondeurs, ou carte de disparité, de l'objet à dissimuler. Cette carte s'obtient grâce à deux visions d'une même scène prises à deux endroits différents, et permet de mettre en avant les informations sur la profondeur de l'objet. Par exemple, la carte des profondeurs de la figure 2.4 a permis d'obtenir l'autostéréogramme de la figure 2.5.

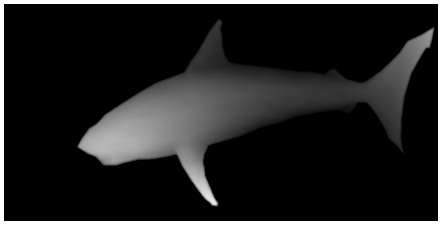

FIGURE 2.4 – Carte des profondeurs [4]

Il existe des algorithmes de génération d'autostéréogrammes très simples , comme celui proposé par Gary Beene [4]. Cependant un algorithme aussi peu optimisé produit des autostéréogrammes défecteux, comportant par exemple des échos (répétition d'une partie de l'objet 3D) ou des artefacts (défaut de l'objet 3D observé).

Des algorithmes plus poussés ont été développés, comme celui présenté par Harold W. Thimbleby, Stuart Inglis et Ian H. Witten [5], et celui proposé par W. A. Steer [1], associés à une réalisation en C. Ils permettent la génération d'autostéréogrammes SIRDS (Single Image Random Dots Stereogram) à partir d'une image 2D. Ces articles présentent également des méthodes de correction de problèmes tels que l'écho ou la gestion des faces cachées (faces de l'objet tridimensionnel visibles par un seul œil). W. A. Steer propose de plus une méthode de génération d'autostéréogrammes utilisant un motif bitmap comme image de base plutôt qu'un nuage de points aléatoires, ce qui pallie à certaines limites des SIRDS comme le manque de détails et la grossièreté des surfaces courbes.

2.3.3 Les anaglyphes

Un anaglyphe est une image sur laquelle on superpose deux vues, si possible différentes, d'une scène. Un exemple d'anaglyphe est donné dans la figure 2.6. La meilleure distance entre ces deux visions est la même que celle entre les deux yeux, afin que le cerveau puisse recréer la même vision en trois dimensions que dans la réalité.

4. http://en.wikipedia.org/wiki/Autostereogram
5. http://en.wikipedia.org/wiki/Autostereogram
6. http://zour.deviantart.com/art/Wernigerode-Boulevard-Dubois-Anaglyph-HDR-3D-276542278

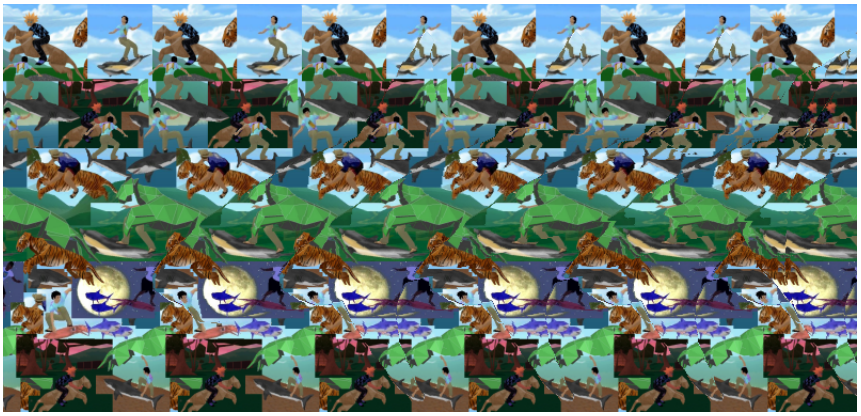

FIGURE 2.5 – Autostéréogramme obtenu [5]

FIGURE 2.6 – Anaglyphe [6]

Les anaglyphes les plus fréquents sont les anaglyphes dits rouge-cyan. Ils se nomment ainsi car ils sont constitués d'une image sur laquelle on passe un filtre magenta, et une autre avec un filtre cyan (cf. figure 2.7). Pour pouvoir visualiser le relief sur une telle image, on utilise une paire de lunettes rouge-cyan, dont chaque verre est un filtre pour l'une des deux couleurs de l'image. Le plus souvent, le filtre magenta est placé sur l'œil gauche, le cyan sur l'œil droit. En regardant l'image, l'œil gauche ne verra alors que la composante cyan, et inversement pour l'œil droit. Les deux images ayant un léger décalage, le cerveau va percevoir l'image comme si elle était en trois dimensions. Les anaglyphes rouge-cyan sont principalement intéressants sur des images en noir et blanc. En effet, quand il s'agit d'images en couleur, celles-ci sont souvent détériorées par l'usage des filtres, car les couleurs possèdent généralement de plusieurs composantes. A l'inverse, quand l'image est en nuances de gris, l'image n'est pas modifiée, juste mise en relief. Plusieurs types d'algorithmes existent pour créer des anaglyphes depuis des paires d'images. Ils se différencient par la qualité de l'anaglyphe en sortie en diminuant le nombre d'artefacts [8] ou en améliorant le rendu pour l'impression [6] par exemple.

8. `http://www.david-romeuf.fr/3D/Anaglyphes/TCAnaglypheLSDubois/TransformationCouleursPourAnaglyphe.html`

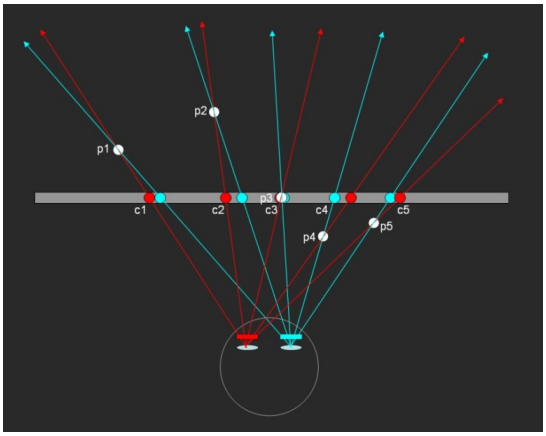

FIGURE 2.7 – Le décalage de la partie rouge et cyan permet à l'œil de percevoir l'image non plus dans un plan XY mais dans l'espace [8]

2.4 Résumé du cahier des charges

A partir de nos recherches sur le domaine et les algorithmes existants pour la réalisation de notre cahier des charges, nous avons rédigé un cahier des charges pour définir les besoins fonctionnels et non fonctionnels de notre projet. Ca cahier est un contrat passé entre les clients et les programmeurs, qui définit de façon exhaustive les engagements de ces derniers vis à vis du produit final.

Pour un tel logiciel, les besoins fonctionnels sont relativement simples à cerner, à savoir la création d'une scène, sa manipulation, et les prises de vue pour obtenir les rendus demandés dans le sujet.

Toutefois, les besoins non fonctionnels, même s'ils peuvent être clairement énoncés par le client, sont bien souvent implicites et doivent être déterminés en fonction du discours tenu. Pour un logiciel de synthèse d'images, la fluidité d'affichage est bien souvent un besoin essentiel, car l'utilisateur s'attend à une certaine rapidité du logiciel lors de la manipulation de la scène et de la génération de rendus.

Mais d'autres besoins, à savoir la portabilité du logiciel et sa maintenabilité, ont également été exprimés par les clients et ont été considérés comme essentiels pour ce projet. En effet, l'utilisaton de ce logiciel, au moins sous les systèmes d'exploitation Linux et Windows, était importante pour pouvoir travailler sur différentes machines dans leur travail.

De plus, un tel logiciel pouvait être amené à être amélioré ou réutilisé en partie dans de futurs projets, et c'est pourquoi il devait être facilement maintenable.

Ces besoins ont apportés des contraintes quand à la réalisation du logiciel, par exemples sur les langages et les bibliothèques utilisées. Pour s'assurer de la faisabilité des besoins fonctionnels et non fonctionnels du projet, deux prototypes ont été mis en place : un prototype de fichier XML pour réfléchir au format de sauvegarde et de chargement de la scène, et un prototype de scène en trois dimensions pour tester les fonctionnalités possibles avec le langage C++ et les bibliothèques que nous souhaitions utiliser : Qt et OpenGL pour Qt. Ce prototype aura également permis de s'assurer de la portabilité de ces bibliothèques, ainsi que de la fluidité potentielle du futur logiciel.

Le prototype XML aura permis de réfléchir aux informations importantes à stocker pour pouvoir recréer une scène à l'identique. Tout d'abord, les informations relatives aux objets doivent être stockées, telles que leur nom et leur fichier d'origine, mais également leur position, leur orientation et leur échelle. Les informations relatives à la caméra sont également essentielles pour retrouver l'angle d'observation de la scène, et les informations à stocker sont sa position, son orientation, ou encore son angle de vue. Le prototype d'origine du fichier XML est donné dans la partie Annexe

du Cahier des Charges, lui-même présent en Annexe de ce rapport. Un exemple de fichier XML obtenu à partir de ce prototype est également donné en Annexe.

Le prototype logiciel réalisé en C++ a permis de mettre en place une première version de scène contenant un objet obtenu grâce à une première version de parseur de fichiers d'extension PLY. La manipulation de cette scène a permis de tester quelques valeurs de fluidité pour pouvoir se rendre compter des capacités d'affichage des bibliothèques utilisées, qui semblaient satisfaire nos attentes.

De plus, le test du prototype sur des machines Windows et Linux ont montré la portabilité des bibliothèques, même si les versions de shaders utilisées auront posé problème par la suite comme il est détaillé dans la partie suivante de ce rapport. Un exemple d'affichage du prototype initial est donné dans la [FIGURE N°], et des exemples de l'affichage final seront donnés plus tard dans ce rapport.

Pour satisfaire le besoin d'extensibilité de notre projet, nous avons mis en place une architecture modulaire pour permettre à notre logiciel d'être modifié de façon simple. L'architecture principale est donné dans la Figure 3.3.

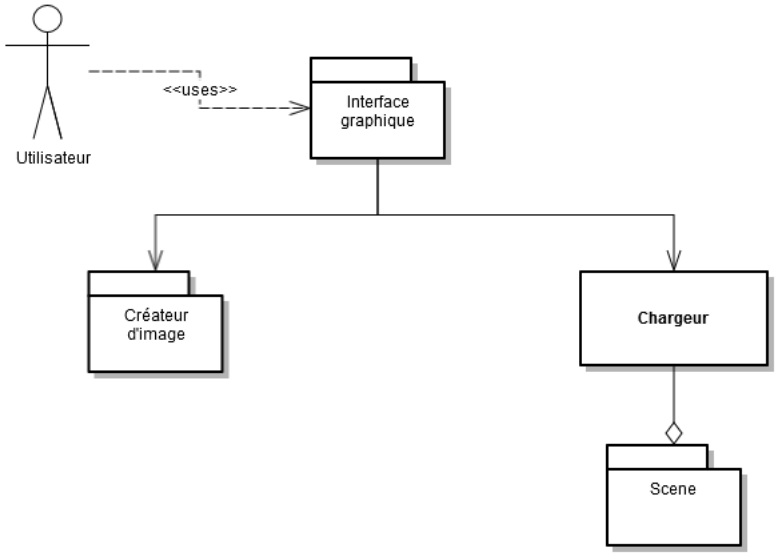

FIGURE 2.8 – Diagramme des paquetages, architecture globale du projet [9]

Le paquetage Creation a été implémenté dans l'intention d'utiliser au cours du projet des outils propres aux bibliothèques Qt et OpenGL pour Qt, comme le montre la figure 3.5. Les classes Point et Vecteur seront alors remplacés par les classes correspondantes de Qt.

Les besoins en modularité de notre projet se reflète particulièrement dans le paquetage Création de l'architecture, qui permet la création des différents rendus que le logiciel permet d'obtenir. L'architecture de ce paquetage est donné dans la Figure 3.4.

L'extensibilité de cette architecture se traduit principalement par l'utilisation d'interfaces à différents niveaux. Tout d'abord, le Creator, point d'entrée du paquetage, a pour attribut une première interface Creation qui peut devenir n'importe lequel des rendus souhaité par l'utilisateur. Il peut donc aussi bien devenir un anaglyphe qu'un autostéréogramme. Toutefois, pour permettre l'utilisation de divers algorithmes pour réaliser un même rendu, les classes des rendus sont

9. Réalisé grâce au logiciel Gliffy : `www.gliffy.com`
10. Réalisé grâce au logiciel Gliffy : `www.gliffy.com`
11. Réalisé grâce au logiciel Gliffy : `www.gliffy.com`

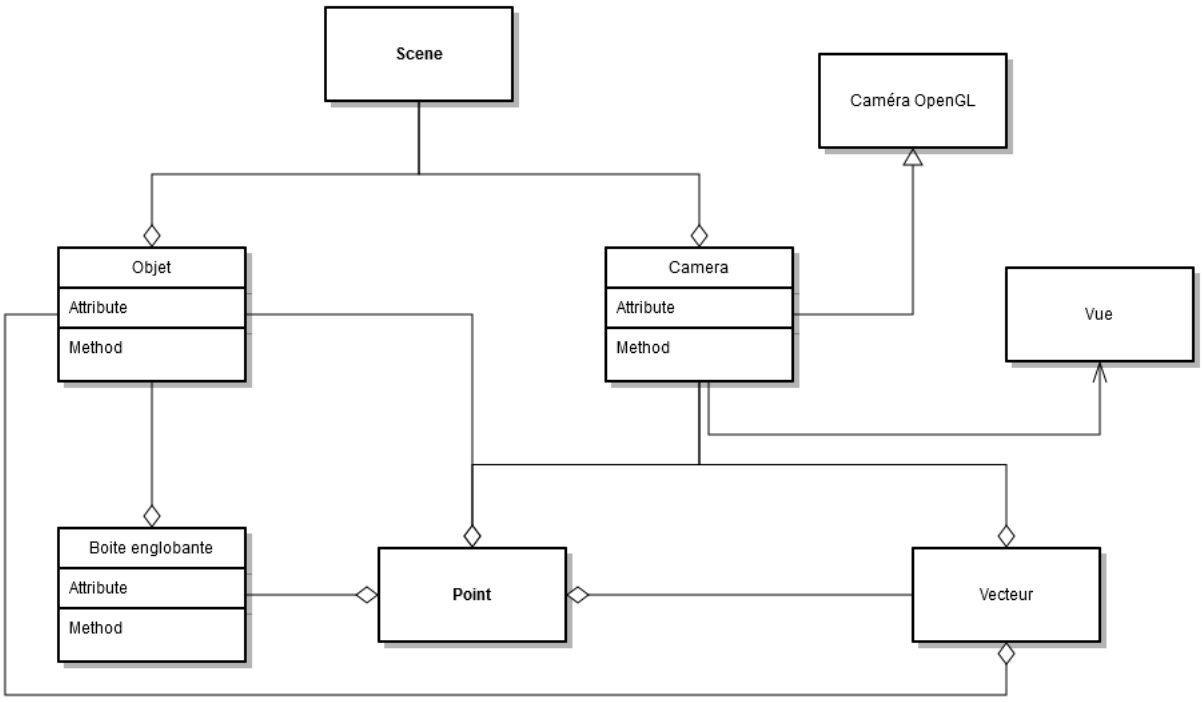

FIGURE 2.9 – Paquetage Scene en début du projet [10]

également des interfaces dont héritent les classes réellement instanciables qui correspondent chacune à un algorithme. On aura donc par exemple AnaglyphAlgorithm1 ou encore DepthMapAlgorithm2.

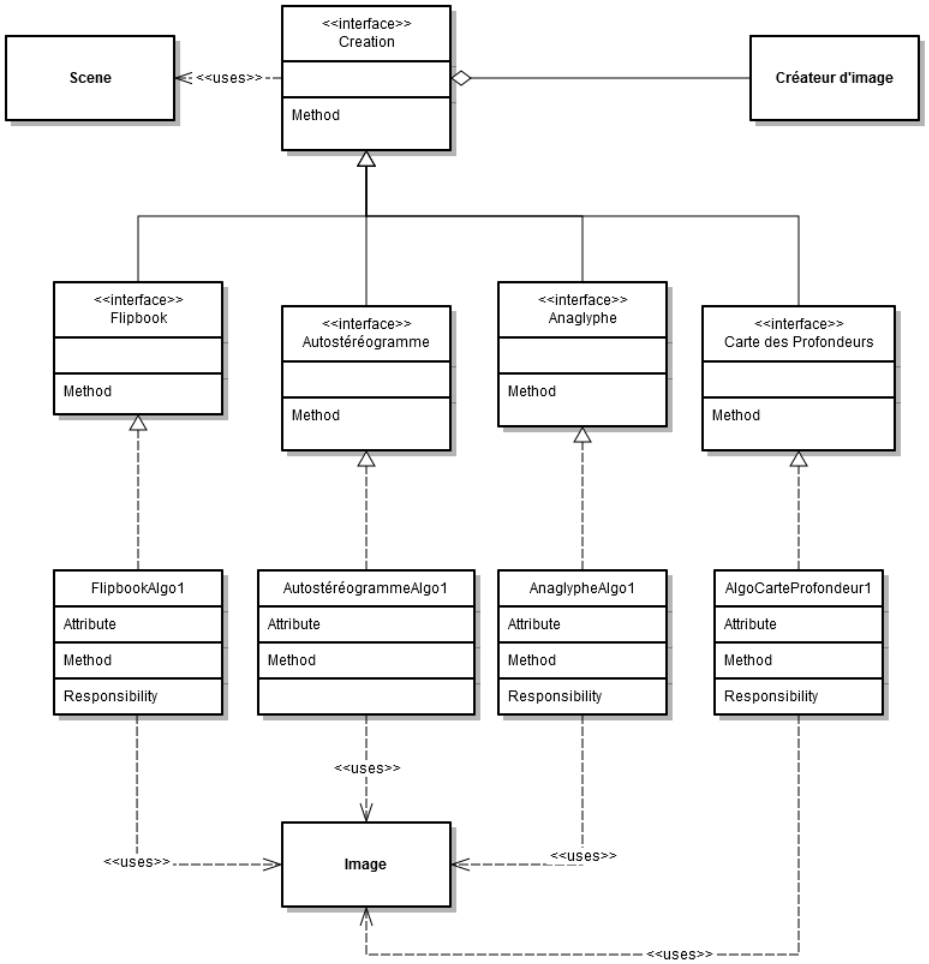

FIGURE 2.10 – Paquetage Création en début de projet [11]

3 Déroulement de la réalisation technique

Nous présenterons dans cette partie les choix faits pour la réalisation du logiciel concernant le langage, les bibliothèques et les algorithmes des rendus, et nous détaillerons les étapes de la réalisation.

Une notice d'installation et une notice d'utilisation de notre logiciel Project3Donuts sont fournies en Annexe de ce rapport.

3.1 Choix de programmation

Pour la réalisation du projet, nos clients nous proposaient des langages comme le C, le C++ ou encore le Python. Le C++ nous est apparu comme le choix le plus judicieux pour notre logiciel. En effet nous voulions un langage orienté objet, pour avoir du code lisible et une architecture claire, et un langage qui soit rapide pour pouvoir effectuer des rendus en un temps raisonnable.

L'utilisation du C++11 nous permet d'utiliser entre autres : les std : :unique_ptr pour limiter les risques de fuite de

mémoire ou les lambda expressions pour garder en mémoire les actions de l'utilisateur.

Comme nous l'avons exprimé plus haut, l'un des besoins non fonctionnels primordiaux de notre projet est la maintenabilité du code. Pour celà, il nous fallait choisir des outils et des bibliothèques qui étaient destinées à perdurer le plus longtemps possible.

L'utilisation de Qt et d'OpenGL est assez répandue pour des logiciels de visualisation et de manipulation en trois dimensions. De plus Qt est très complet et propose un vaste choix de modules qui permettent de traiter un ensemble très variable de tâches (par exemple l'analyse de fichiers XML ou la gestion d'évènement). Comme ces bibliothèques sont parfaitement portables, elles convenaient parfaitement à l'implémentation que nous souhaitions réaliser.

La version 5 de Qt est la plus récente actuellement, mais malgré sa jeunesse elle est devenu au fil des modifications suffisament stable pour pouvoir être utilisée sans problème, et elle dispose d'une communauté Internet active prête à aider en cas de difficultés de code. De plus, cette bibliothèque utilise la version ES 2.0 de OpenGL, qui est une version non seulement qui utilise des shaders, mais également qui peut être utilisée pour de la programmation d'applications mobile par exemple. C'est donc pour cette variété, cette communauté, cette maintenabilité et cette extensibilité que nous avons choisi d'utiliser ces bibliothèques et ces versions.

Pour la compilation du projet et des bibliothèques, nous avons choisi d'utiliser CMake, qui est répandu et portable. Nous aurions pu choisir également QMake, mais celui-ci dépandant trop fortement de QtCreator, nous avons préféré ne pas l'utiliser.

3.2 Choix des algorithmes d'anaglyphes

algo cherchés, trouvés, implémentés raisons des choix, difficultés rencontrées (dire pourquoi on a utilisé Dubois mais pas celui du cahier des charges

comment générer des anaglyphes

les problèmes (qui permettent d'en évaluer la qualité)

description brève des algos (comme dans l'article)

pour chaque algo : explication/description difficultés rencontrées les résultats comparés (pour chaque algo, les points positifs et négatifs avec l'image)

3.2.1 La génération des anaglyphes et ses défauts

L'anaglyphe est une image issue d'un traitement particulier sur deux prises de vue stéréoscopiques et qui nécessite des lunettes particulières composées d'un filtre rouge (ou magenta ou jaune) pour un oeil et cyan (respectivement vert ou bleu) pour l'autre.

La génération des anaglyphes passe par trois étapes essentiellement. Tous d'abord, une prise de vue stéréoscopique d'une scène 3D avec les deux points de vue éloignées d'une distance proche à celle entre les yeux, de manière à obtenir une vue pour chaque oeil, est nécessaire. Ensuite, un traitement d'image est opéré sur ces deux images obtenues, avec au minimum deux filtres rouge et cyan (ou deux autres couleurs correspondants à celles des lunettes)sont appliquées. Finalement, les deux images résultantes sont superposées pour n'obtenir qu'une seule image qui pourra être visionnée à travers les lunettes avec une impression de trois dimensions.

Les algorithmes de génération des anaglyphes proposent des traitements d'image différents (deuxième étape) ayant pour but d'améliorer la qualité des anaglyphes. Un critère important pour juger de la qualité des anaglyphes est l'absence des artefacts. Ces derniers sont des perceptions d'un phénomène de diaphonie qui est le résultat d'une mauvaise séparation des deux vues (un oeil perçoit un détail de la vue destinée à l'autre oeil), et dégradent l'impression de trois dimensions.

Cependant, aucun algorithme ne peut être parfait, comme il est expliqué dans l'article d' Andrew J. Woods et Chris R. Harris [9] : la qualité de l'anaglyphe dépend de l'image, des lunettes employées et du support affichant l'image (l'article traite principalement les écrans mais il en va de même pour des anaglyphes imprimés avec le choix du papier). Hormis les lunettes, les deux autres facteurs ont toujours été problématique pour le traitement d'image en général.

Pour obtenir des algorithmes très performants, il faudrait prendre en compte tous ces facteurs et adapter au cas par cas. En effet, étant donné que les différentes caractéristiques (luminosité, résolution,) varient pour chaque écran,

obtenir une même couleur sur deux écrans distincts (produits par des entreprises différentes) supposent un calibrage au préalable ou l'utilisation d'une palette de couleur limitée. Ainsi, les techniques employés dans le domaine de recherche de la génération des anaglyphes restent très souvent basées sur des connaissances empiriques.

Nous avons choisi d'implémenter trois méthodes qui permettent d'obtenir des images de qualité différentes, pour que l'utilisateur puisse choisir pour chaque donnée entrée l'algortihme qui rendra l'anaglyphe de meilleure qualité.

3.2.2 La méthode Photoshop

Le premier algorithme implémenté est le plus basique : c'est la méthode dite Photoshop qui se contente d'appliquer les deux filtres de couleur sans réaliser d'autres traitements d'image. Cette méthode est décrite dans l'article rédigé par W. Alkhadour, S. Ipson, J. Zraqou, R. Qahwaji et J. Haigh [8].

En pratique, ce ne sont pas des filtres de couleur qui sont appliqués, mais pour chaque image seules les composantes rouges pour l'une et les composantes bleues et vertes pour l'autre sont recupérées et restituées dans chaque image respectivement. Ces deux images sont ensuite "superposées", c'est-à-dire que celles-ci sont parcourues et pour chaque pixel les composantes en couleur (RGB) sont récupérés, additionnées deux à deux et réinsérées dans les canaux de couleurs du pixel correspondant dans l'image résultat.

insérer ICI l'image pour la méthode photoshop

Au niveau du résultat, on constate la présence de quelques d'artefacts et que l'impression de 3D est bien présente.

3.2.3 La méthode des moindres carrés avec correction gamma

Le deuxième algorithme implémenté est basée sur la méthode des moindres carrés présentée dans l'article [2] par Eric Dubois. Dans cet article, comme David Romeuf l'explique [7] l'approche consiste à prendre en compte le spectre d'absorption des filtres des lunettes, la densité spectrale des sources primaires des écrans d'ordinateur et de la sensibilité spectrale de l'œil humain, pour reproduire au mieux une image anaglyphe dont les couleurs sont proches de celles contenues dans l'image originale.

L'ensemble des couleurs visibles sur l'écran à travers les lunettes étant un sous ensemble de celui de l'écran, une transformation particulière est nécessaire et c'est pour cette raison que les moindres carrés interviennent dans la projection pour minimiser la distance entre les deux couleurs (celle obtenue et l'originale) dans le diagramme colorimétrique.

L'algorithme implémenté met en oeuvre une partie de la méthode décrite plus précisément dans cet article par Eric Dubois [3] : notre algorithme utilise les matrices issues du calcul par projection avec la méthode des moindres carrés, et une correction gamma mais ne modifie pas la saturation.

La correction gamma permet d'augmenter la luminosité, appliquée aux anaglyphes rouge-cyan le rouge est plus visible et l'impression de trois dimensions est renforcée. Cependant, l'augmentation de la luminosité implique une réduction des contrastes et peut résulter en une image avec les détails peu soignés.

INSERER IMAGE

On constate qu'il y a très peu d'artefact et la 3D est de meileure qualité que pour la méthode Photoshop. Toutefois, la qualité en couleur est légèrement perdue et les détails de l'image sont peu visibles.

3.2.4 La méthode des moindres carrés avec saturation

A REDIGER même algo que la partie précédente principalement, sauf qu'il n'y pas de correction gamma et qu'il y des modifications de la saturation.

image resultat -> l'algo dubois avec saturation -> la diminution de saturation, atténue trop les couleurs : moins bonne 3D mais moins fatiguant pour les yeux

3.2.5 Comparaison des différents algorithmes

En comparant les algorithmes, bien que la méthode Photoshop présente un résultat satisfaisant, il semblerait que la méthode de Dubois [2] soit celle qui produit un anaglyphe avec le moins d'artefact et qui donne la meilleure impression

3D.

Cependant, après plusieurs comparaisons des images (et ce par différentes personnes) la correction gamma employée dans la méthode de Dubois qui améliore l'impression 3D rendrait l'image plus difficile à percevoir la trois dimensions rapidement et fatiguerait plus vite les yeux que pour la méthode Photshop.

3.2.6 L'impression des anaglyphes

Dans ce projet, les anaglyphes générées par le logiciel étaient destinées à être visionnés sur écran mais aussi être imprimés sur papier. Le système de colorimétrie d'une imprimante est en CMJN (Cyan Magenta Jaune Noir) et son ensemble de couleur est plus réduit que celui des écrans.

INSERER Diagramme comparatif pour illustrer ?

Par conséquent, le problème pour l'impression réside tout d'abord en la conversion du système RVB au CMJN. Tout comme l'algortihme de Dubois avec la méthode des moindres carrés [2], il faudrait pouvoir convertir en CMJN en essayant d'obtenir une couleur qui sera la plus proche possible de celle en RVB.

ICI ??? Lorsque ce problème a été exposé en réunion avec les clients, il nous a été spécifié que nous n'avions pas besoin d'implémenter des méthodes pour améliorer l'impression.

Par ailleurs, la conversion n'est pas le seul problème puisque comme expliqué avec l'article Andrew J. Woods et Chris R. Harris [9], le support d'affichage est aussi important : dans ce cas, le type de papier rentre aussi en compte dans la qualité de l'anaglyphe.

3.3 Choix des algorithmes d'autostéréogrammes

algo cherchés, trouvés, implémentés raisons des choix, difficultés rencontrées

Les deux algorithmes implémentés ont un squelette très similaire ; il s'agit, à partir de la carte des profondeurs, de lier deux par deux les pixels de l'image générée puis de colorier chaque paire de la même couleur.

Les deux algorithmes sont dotés d'une technique anti-surfaces cachées (implémentée différemment dans les deux cas). D'autre part, ils supportent tous deux l'utilisation de textures

3.3.1 Algorithme de base (Witten, Inglis, Thimbleby)

Cet algorithme a été choisi pour sa simplicité qui le laisse tout de même fonctionnel ; il a été conçu à l'origine pour des autostéréogrammes aléatoires en noir et blanc (SIRDS) mais a été adapté ici à d'autres types de rendus.

Les résultats obtenus sont cependant assez peu satisfaisants pour la représentation d'objets complexes : l'image est assez plate, et l'objet en relief est fait de paliers superposés plutôt que d'avoir une surface lisse. De ce fait, représenter des détails ou des courbes est ardu. L'utilisation de textures peut permettre d'atténuer certains défauts visuels, mais pas de les effacer totalement.

3.3.2 Algorithme de W. A. Steer

L'algorithme de W. A. Steer a été choisi à cause des nombreuses améliorations qu'il apporte par rapport à l'algorithme précédent. En effet, il utilise une méthode de suréchantillonnage de l'image : dans une première étape, un autostéréogramme n fois plus large que l'image de base est construit puis les pixels sont fusionnés par groupes de n. Ces étapes permettent d'améliorer la résolution en profondeur de l'autostéréogramme.

Cet algorithme est plus lent que le précédent à cause du suréchantillonnage, mais les résultats obtenus sont nettement plus agréables à regarder car la résolution en profondeur est meilleure : l'image apparaît plus profonde. De plus, l'utilisation conjointe d'une texture et du suréchantillonage lisse l'image finale ; on n'y retrouve plus les "paliers" des résultats précédents. Ces paliers peuvent être visibles si le rendu choisi est de type aléatoire, mais ils sont beaucoup plus nombreux et rapprochés.

3.4 Réalisation du projet

3.4.1 Présentation générale de la réalisation

La partie Réalisation de ce projet se sera étalée sur les mois de Décembre, Janvier, Février et Mars. Grâce à la réflexion effectuée durant la phase de rédaction du cahier des charges, nous avons pu aborder notre projet avec une idée claire des priorités.

La partie Algorithmique était primordiale pour nos clients, c'est pourquoi nous avons dès le début de la réalisation mis en place deux équipes de deux personnes pour les deux algorithmes principaux : les anaglyphes et les autostéréogrammes. Les folioscopes quand à eux ne demandaient pas d'algorithme particulier de réalisation, si ce n'est la manipulation de la caméra vis à vis de la scène qui allait être mise en place grâce à la troisième équipe.

En effet, notre troisième équipe, constituée des trois derniers membres, a pu continuer de travailler sur le prototype généré lors de la phase du cahier des charges. Il a ainsi été amélioré pour mettre en place les parseurs de fichiers qui allaient être nécessaires pour la génération des objets de la scène, et améliorer la manipulation de la scène.

Malheureusement, l'implémenation des algorithmes s'est révélée plus longue que nous l'avions initialement prévue. En effet, cette partie étant l'une des plus importantes pour nos clients, nous avons eu besoin d'aller plus loin que ce que nous avions envisagé dans un premier temps. Il a également fallu que nous fassions davantage de recherches pour compléter celles qui avaient été effectuées pour la rédaction du cahier des charges. De plus amples explications seront données dans la partie Difficultés rencontrées de ce rapport.

Au final, l'implémentation des algorithmes Anaglyphe et Autostéréogramme aura duré environ deux mois et demi au lieu du mois initialement prévu, avant de pouvoir les intégrer au reste du projet.

En parallèle, la troisième équipe, appuyée parfois par l'un ou l'autre des autres membres en cas de besoin, a continué d'avancer sur la partie Scène du logiciel. L'implémentation de certaines manipulations de scène (déplacements de la caméra ou des objets notamment) auront parfois étaient plus rapides que prévu, mêmes si quelques retours en arrière ont été nécessaires au milieu du projet comme nous l'expliqueront dans la partie Difficultés rencontrées.

L'ordre chronologique initialement prévu dans le diagramme de Gantt n'aura finalement pas toujours été respecté. Par exemple, la différence de travail entre l'ajout d'un objet dans la scène et l'ajout de plusieurs objets n'étant pas énorme, ces deux parties auront été effectuées simultanément. Il en est de même pour la sauvegarde automatique qui aura été ajoutée en même temps que la sauvegarde classique de la scène, ou encore pour les manipulations de la scène et de l'objet dont l'implémentation était grandement facilitée par l'utilisation de la bibliothèque OpenGL pour Qt.

Au final, la totalité des tâches prévues jusqu'au début du mois de Mars auront pu être effectuées dans les temps, nous laissant le mois de Mars pour l'intégration des algorithmes au logiciel, l'amélioration de certaines fonctionnalités, le développement de l'interface et la rédaction de ce rapport.

3.4.2 Les algorithmes du logiciel

Pour pouvoir permettre à l'utilisateur de notre logiciel d'obtenir des anaglyphes, des autostéréogrammes, ou encore des folioscopes, il aura fallu à nos programmeurs un ensemble de recherches pour comprendre le fonctionnement de ces illusions d'optique, collecter un ensemble d'algorithmes permettant de les obtenir, puis sélectionner les meilleurs d'entre eux avant de les implémenter.

Cet exercice s'est révélé assez complexe car le travail de recherche qui s'y rapporte a été long, et il était parfois difficile de trouver les bonnes informations pour répondre à nos interrogations. Nos clients nous auront longuement accompagné dans notre travail de recherche, et nous aurons conseillé sur des outils pour trouver des articles pertinants. Cette partie Recherche de l'existant, qui s'est principalement déroulée durant la phase de rédaction du cahier des

charges, aura été formateur car il nous aura appris à utiliser les bons outils et les bons mots-clés pour rendre nos recherches fructueuses, plutôt que de devoir ré-inventer la roue.

Une fois les algorithmes sélectionnés, leur implémentation s'est déoulée en parallèle de l'édition du logiciel. Il a donc fallu que nous nous mettions d'accord sur les éléments à prendre en entrée et en sortie pour leurs fonctions, afin de satisfaire l'architecture du projet et de faciliter par la suite l'intégration des algorithmes dans le logiciel.

Cet exercice nous aura obligé à trouver des astuces pour pouvoir tester nos algorithmes sans pouvoir utiliser la scène qui était mise en place en parallèle. On considère alors nos algorithmes comme des boîtes noires, indépendantes du logiciel, ce qui montre bien la modularité du projet final.

L'utilisation de cartes de profondeurs et de paires d'images stéréoscopiques trouvées sur Internet auront permis de tester et de corriger les algorithmes pour qu'ils soient prêts pour leur intégration au logiciel.

Lors du dernier mois de notre projet, nous avons pu intégrer nos algorithmes au logiciel qui avait été implémenté aux parallèles. Malgré les tests qui avaient été effectuées sur le module des algorithmes, quelques problèmes sont apparus avec les anaglyphes : les rendus étaient faux car les couleurs bleu et rouge étaient parfois inversées.

Ce problème ne provenait finalement pas de l'algorithme en lui-même, mais de la scène utilisée. En effet, les tests effectués sur des images n'avaient pas de contours aussi nets que les objets de la scène, et le rendu obtenu était correct. Toutefois, les fichiers d'extension OBJ et PLY sont constitués de triangles, et leurs bords sont très marqués, ce qui cause parfois une erreur au moment de la génération de l'anaglyphe. Une solution possible pour régler ce problème aurait été de rendre les contors des objets moins abruptes, mais cette solution n'a pas été implémentée dans notre logiciel.

Au final, le travail sur les algorithmes aura été très enrichissant car il aura demandé de nombreuses recherches d'articles, un esprit critique pour comparer les algorithmes, puis un travail d'implémentation pour faire correspondre les algorithmes à notre logiciel. L'obtention d'un rendu à partir de la scène est montré dans la figure **??**.

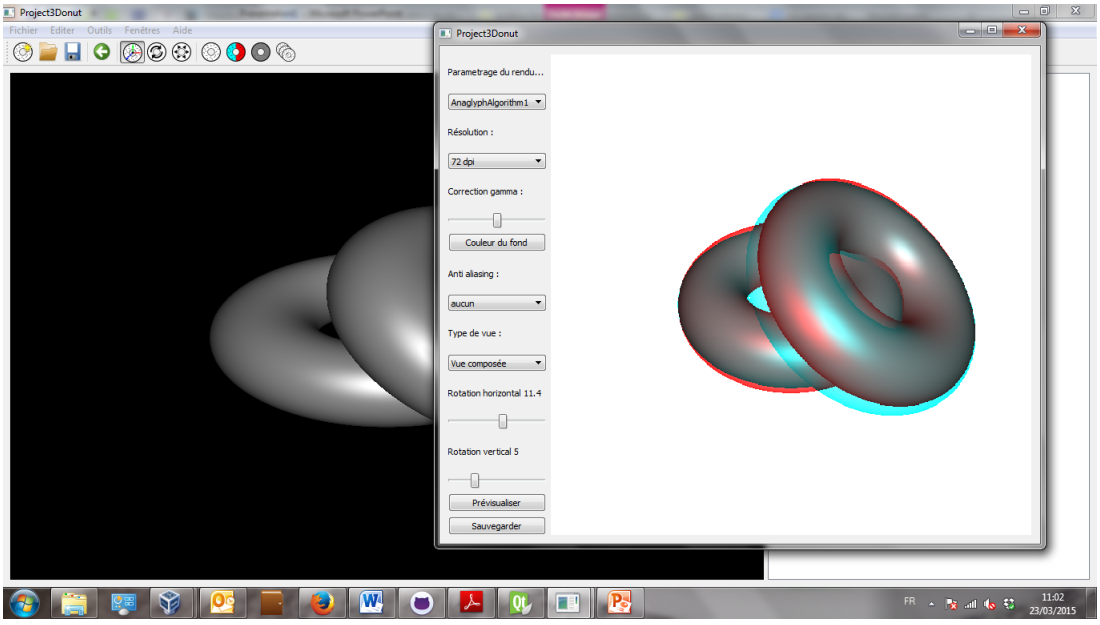

FIGURE 3.1 – Obtention d'un anaglyphe dans le logiciel

3.4.3 La manipulation de la scène

Grâce à la rédaction du cahier des charges, les notions de scène en trois dimensions, de caméra et d'objet avait été acquise par notre équipe avant la phase d'implémentation. Un premier prototype de la scène, présenté dans la partie Cahier des charges, aura permis une première manipulation de ces notions et des bibliothèques Qt et OpenGL pour Qt afin de créer une première scène constituée d'un première objet. Les manipulations premières de la scène, notamment la rotation de la caméra tout autour de la scène, auront ainsi été mise en place pour pouvoir juger des capacités des bibliothèques, et déterminer la faisabilité de nos objectifs.

Dès la fin du cahier des charges, la scène aura été grandement modifiée. Les parseurs auront été finalisés pour pouvoir charger l'ensemble des fichiers objets demandés par les clients, et le stockage des objets de la scène aura été implémenté de telle façon que la gestion de plusieurs objets dans la scène a directement été opérationnelle.

D'autres fonctionnalités, relatives à la scène et demandées par le cahier des charges, ont été implémentées peu à peu au cours de la phase de réalisation du projet. Au niveau de la scène, les rotations autour de la scène sont toujours possibles, ainsi qu'une fonction de zoom pour s'approcher ou se reculer de la scène. Au niveau des objets, la sélection d'un objet est possible soit en cliquant directement sur l'objet avec le bouton droit de la souris, soit en le sélectionnant dans la liste des objets donnée sur la fenêtre principale du logiciel. Une fois un objet choisi, on peut le déplacer, le faire tourner, modifier sa taille, sa couleur, ou encore le supprimer.

Grâce à l'ensemble des manipulations à mettre en place sur la scène, notre équipe a pu se familiariser et apprendre à utiliser les bibliothèques Qt et OpenGL pour Qt. De plus, la communauté Internet de la bibliothèque Qt nous aura été fort utile car elle aura permis de trouver des réponses à d'éventuels problèmes rencontrés, voire de poser des questions dans des cas particuliers.

La fenêtre principale du logiciel est montrée dans la figure 3.2.

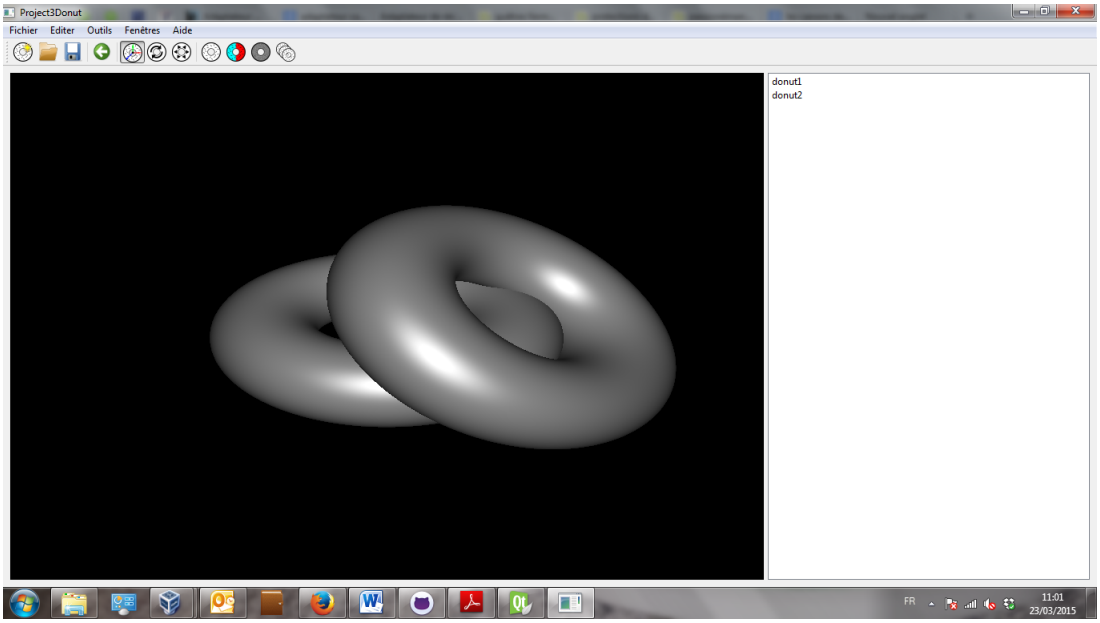

FIGURE 3.2 – Visualisation de la scène dans le logiciel

3.4.4 Les sauvegardes et chargement de la scène

Pour satisfaire la portabilité du logiciel, le nombre de bibliothèques utilisées devait être le plus faible possible. Fort heureusement, la bibliothèque Qt, portable sur la plupart des plateformes, propose une grande quantité de fonctionnalités, y compris le module QtXml qui permet la lecture et le découpage d'un fichier XML. Au cours de la phase de rédaction du Cahier des charges, nous avions mis au point un prototype de fichier XML pour permettre la sauvegarde et le chargement de la scène, que nous avons pu utiliser comme nous l'avions prévu grâce à Qt. Un exemple de fichier utilisé pour le chargement d'une scène effective dans notre logiciel, nommé 'MaScene.xml', est donné en Annexe.
[ANNEXE XML]

La sauvegarde d'une scène consiste principalement à écrire dans un fichier texte, en respactant le format souhaité par le format XML et en récupérant les bonnes informations des Objets et de la Caméra de la scène. Toutefois, deux types de sauvegarde ont été mises en place : une sauvegarde manuelle et une sauvegarde automatique.

La sauvegarde manuelle s'effectue de la même façon que dans la plupart des logiciels. A la demande de l'utilisateur, le logiciel va lancer une sauvegarde dans un fichier dont le nom est donné, et une fois que celle-ci sera achevée l'utilisateur pourra recommencer à travailler sur sa scène.

Pour la deuxième sauvegarde, nous avons dû utiliser nos connaissances acquises au cours des enseignements de Programmation Système et de Système d'Exploitation. Nous avons en effet vu d'une part comment utiliser des Threads, et d'autre part qu'un Thread en train de dormir ne demande pas d'intervention du processeur. Nous avons ainsi mis en place la création d'un Thread dès l'ajout d'un premier objet dans une scène, et ce Thread va dormir durant un certain temps avant d'effectuer une sauvegarde automatique puis de se rendormir. Cette attente passive permet ainsi de ne pas gaspiller de temps du processeur, et de sauvegarder les avancées de la scène en cas d'arrêt non souhaité du logiciel. De plus, le temps d'attente entre deux sauvegardes automatiques peut être choisie par l'utilisateur grâce aux paramètres du logiciel.

La difficulté principale du chargement de la scène est de s'assurer de la validité du fichier XML passé en paramètre. Ainsi, de nombreuses vérifications, au fur et à mesure de la lecture du fichier, sont nécessaires afin de pouvoir créer une scène fonctionnelle. La bibliothèque QtXml aura été assez simple d'utilisation, et le chargement aura donc pu être rapidement mis en place, tout d'abord en parallèle du projet pour s'assurer que seuls les fichiers XML valides sont acceptés, puis après intégration pour s'assurer de l'appel des constructeurs et de la génération de la scène souhaitée.

L'utilisation du module QtXMl aura toutefois posé problème avec l'utilisation de CMake. En effet, la plupart des informations données sur Internet permette l'utilisation de ces modules avec QMake, et l'intégration est alors relativement simple. Pour l'intégrer à CMake, il aura fallu comprendre précisément le fonctionnement de CMake et trouver les bons noms de paquetages à intégrer afin que le module puisse être utilisé dans le logiciel.

3.4.5 Architecture finale du projet

L'architecture prévisionnelle de notre projet n'aura que peu été modifiée au fil de notre projet. La structure principale du projet est resté la même, avec les différents paquetages initialement prévus, mais comme le montre la figure 3.3, le Chargeur n'est plus le point d'entrée du paquetage Scene.

Le paquetage Création tel que présenté dans la figure 3.4 n'a pas connu de modifications particulières. En effet, les dépendances entre les différentes classes étaient très importantes pour l'extensibilité du logiciel. Toutefois, de nouvelles classes ont été ajoutées, pour permettre la création des images et des folioscopes.

Le paquetage Interface contient les classes principales permettant la génération des fenêtres du logiciel. Le logiciel QtCreator aura été utilisé pour aider à la génération des ces fichiers sources.

Le paquetage Scene a quant à lui été modifié. Si nous avions initialement prévu que ce soit le Loader qui se charge de créer la scène et les objets qui s'y trouvent, c'est finalement la classe Scene en elle-même qui est devenue le point

12. Réalisé grâce au logiciel Gliffy : `www.gliffy.com`
13. Réalisé grâce au logiciel Gliffy : `www.gliffy.com`

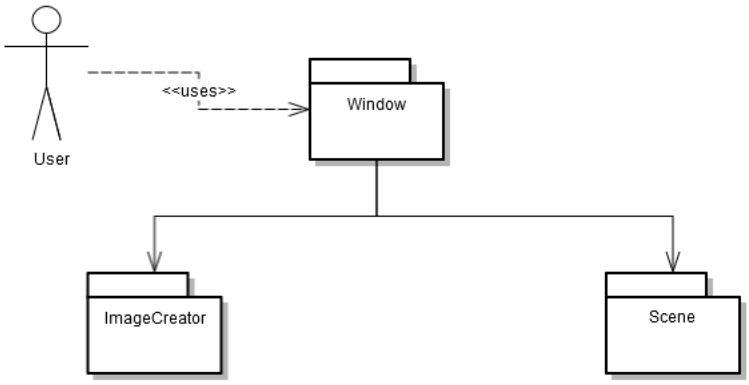

FIGURE 3.3 – Diagramme des paquetages en fin de projet [12]

d'entrée du paquetage Scène. Elle se charge ensuite de transférer les informations nécessaires au Loader pour qu'il se charge de charger les objets. Le paquetage final de la Scene est présenté dans la figure 3.5.

3.5 Test des fonctionnalités

Explication des tests réalisés pour valider les besoins fonctionnels/non fonctionnels du cahier des charges

Pour valider la réalisation des besoins fonctionnels et non fonctionnels présentés dans le cahier des charges, une série de tests a été effectuée sur le logiciel.

Concernant les besoins fonctionnels, nous avons manipulé le logiciel sous les différents systèmes d'exploitation envisagés, à savoir Windows et Linux. Les différentes rotations de la caméra autour de la scène ont bien été implémentées, permettant à l'utilisateur de voir les objets qui y sont placés sous différents angles, et de s'en rapprocher ou de s'en éloigner grâce au zoom. [TRANSLATION] Les objets sont issus de fichiers d'extension OBJ version 3.0, ASCII, ou PLY, versions 1.0, ASCII et binare. Ceux-ci peuvent ensuite être sélectionnés un par un, en cliquant sur la scène ou sur son nom dans la liste des objets. L'utilisateur peut alors les déplacer grâce à des translations et rotations selon trois axes, et modifier leur taille. Ils pourront également être supprimés de la scène. Enfin, les différents rendus prévus ont pu être obtenus à partir de la scène. Pour chacun d'entre eux, une nouvelle fenêtre est ouverte, proposant les paramètres modifiables pour la génération des photographies, anaglyphes, autostéréogrammes, et folioscopes. Pour chacun d'entre eux, on proposera une prévisualisation, et la possibilité d'enregistrer [LES IMAGES INTERMEDIAIRES ayant permis ce résultat, à savoir les vues gauche et droite pour les anaglyphes, la carte des profondeurs pour les autostéréogrammes, et chaque image du folioscope permettant d'obtenir un GIF animé.] [VERIFIER SI CEST OK] [SAUVEGARDES]

Les tests des besoins non fonctionnels ont permis de valider la portabilité et la fluidité du logiciel. L'extensibilité du logiciel aura été permise par l'architecture du logiciel, qui a été présentée dans la partie Réalisation du projet. Pour s'assurer de la portabilité du logiciel, nous avons manipulé le logiciel sous les différentes machines de notre équipe de programmeur, aussi bien sous Linux que sous Windows, avec ou sans carte graphique intégrée. L'ensemble des fonctionnalités précisées dans la paragraphe précédent ont été testées pour s'assurer de leur bon fonctionnement. La fluidité du logiciel a été testée grâce au modèle happy.ply qui avait servi de modèle lors de la réalisation du cahier des charges. [SCREENS] [RESULTATS WINDOWS/LINUX/CHIPSET] [LOGICIEL DE TEST]

14. Réalisé grâce au logiciel Gliffy : `www.gliffy.com`

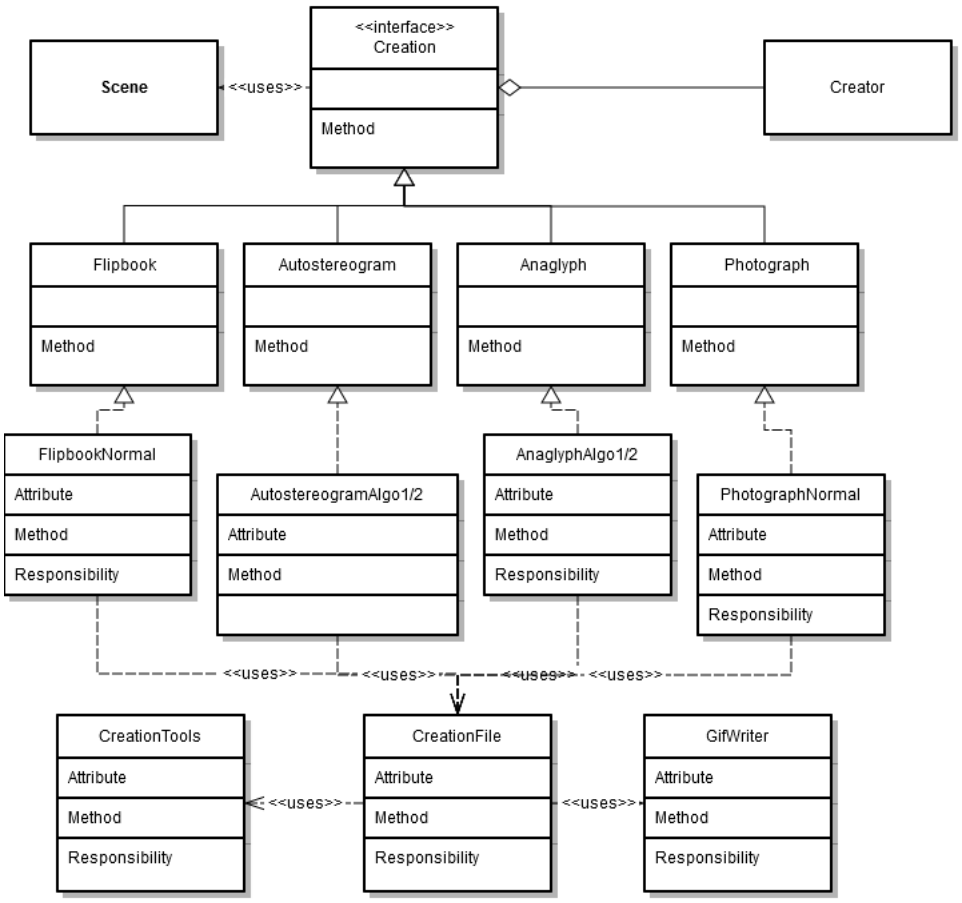

FIGURE 3.4 – Paquetage Creation en fin de projet [13]

L'ensemble de ces tests auront permis de s'assurer de la conformité de notre projet au Cahier des charges.

3.6 Difficultés rencontrées

Au cours de ce projet, certaines difficultés rencontrées auront parfois ralenti l'avancement initialement prévu.

3.6.1 Implémentation des algorithmes

L'implémentation des algorithmes d'anaglyphes et d'autostéréogrammes était le coeur de notre projet et du logiciel que nous avions à mettre en place. Il était donc important que nous lui accordions un temps conséquent durant notre projet, et c'est pour cela que nous avions initialement prévu de faire travailler deux personnes sur ces algorithmes durant un mois, en partant des recherches qui avaient été effectuées pour le cahier des charges. Toutefois, l'implémentation de deux algorithmes étant nécessaire pour chacun de ces rendus, ce mois a finalement été trop court pour nos équipes de programmeurs.

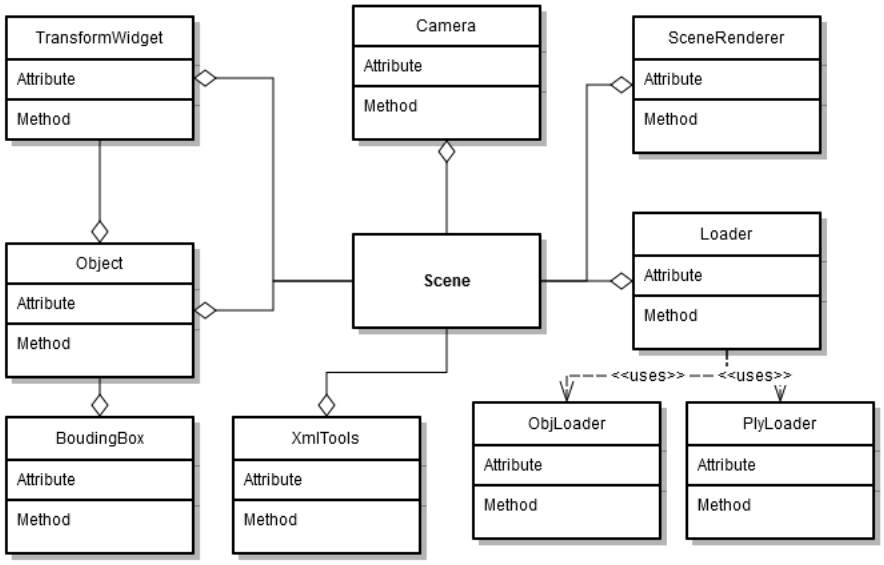

FIGURE 3.5 – Paquetage Scene en fin de projet [14]

Pour les autostéréogrammes, le rendu donné par le premier algorithme choisi était découpé en tranches et ne donnait pas suffisament l'impression de profondeur comme il aurait dû, comme nous pouvoir le voir sur la figure 3.6.

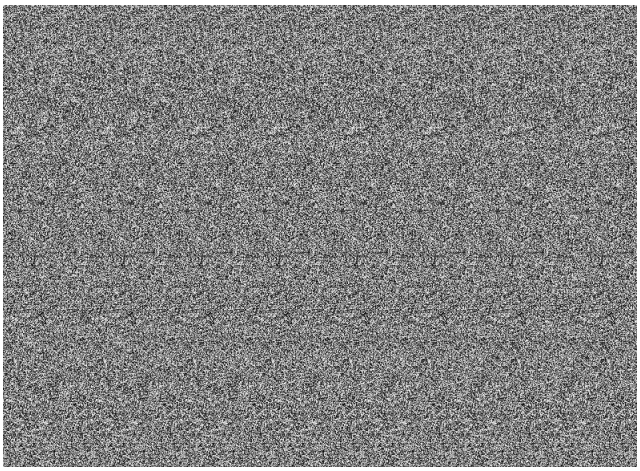

FIGURE 3.6 – Autostéréogramme d'une boule avec un effet en tranche

Cet effet avait en effet été souligné dans l'article de Witten, Inglis et Thimbleby, mais l'utilisation de points aléatoires pour le fond de l'image accentuait cet effet de tranches. Il a alors fallu mettre en place le second algorithme que nous avons présenté précédemment dans ce rapport, ce qui a allongé la période de réalisation. De plus, l'optimisation proposé pour cet algorithme par W.A.Steer posait dans un premier temps problème pour la génération de l'autostéréogramme. Il aura alors fallu procéder étape par étape pour obtenir un premier rendu sans l'optimisation de l'algorithme, qui aura été ajoutée par la suite.

Pour les anaglyphes, la première difficulté était pour la récupération de paires d'images sur lesquelles travailler. Notre équipe a tout d'abord essayé de travailler sur la scène avec OpenGL pour récupérer des images qu'ils pourraient utiliser. Toutefois, une autre difficulté, liée aux shaders comme nous le présenterons dans la partie qui suit, les a empêché de poursuivre dans cette voie. Ils ont donc choisi de travailler sur des images trouvées sur Internet afin d'avancer dans l'implémentation. Une autre difficulté de l'algorithme des anaglyphes est le travail sur la [saturation gamma ?]. Pour permettre un rendu agréable à visualiser, notre équipe a cherché à travailler sur cette saturation, mais l'effet obtenu n'était pas toujours celui qu'ils espéraient. [FINALEMENT ?] Enfin, nos clients souhaitant pouvoir imprimer les anaglyphes, il était très important que les couleurs lors de l'impression révèlent aussi bien l'anaglyphe. En effet, si un écran d'ordinateur possède les composantes rouge, verte et bleue pour son affichage, la plupart des imprimantes utilisent le pattern cyan, magenta, jaune et noir. Pour éviter d'éventuels artefacts qui auraient pu être causés par la transformation automatique des couleurs, il a fallu traiter celles-ci pour que l'impression soit le plus parfaite possible.

3.6.2 L'utilisation de shaders

On appelle shaders des programmes informatiques qui vont travailler directement sur la carte graphique ou le processeur graphique d'un ordinateur pour permettre un rendu meilleur et plus rapide. Ces shaders sont souvent utilisés en imagerie numérique, notamment par le logiciel Blender qui aura été un des logiciels-modèle pour la création du nôtre. L'utilisation de ces shaders nous paraissait importante pour permettre un rendu agréable dans notre logiciel, et pour atteindre nos objectifs de fluidité énoncés dans notre Cahier des charges.

Dans certains ordinateurs, et surtout dans les ordinateurs portables, il n'existe pas de carte graphique à proprement parlé. Un chipset graphique en général peu puissant le remplace. Toutefois, ces chipsets empêchent parfois l'utilisation de shaders trop récent.

Dans une première version de notre logiciel, nous avions souhaité utiliser la version 3.3 des shaders. Toutefois, celle-ci posait problèmes car elles n'étaient pas reconnue sous des machines Linux. La portabilité du logiciel étant primordiale, du moins sous les environnements Windows et Linux, nous avons dû effectuer une modification de l'ensemble du code déjà construit afin de passer à la version 2.0 des shaders qui est bien plus ancienne qui permet la portabilité du logiciel sur quasiment toute les machines.

3.6.3 Les outils utilisés pour la réalisation du projet

Pour permettre la portabilité et la maintenabilité du projet, nous avons choisi de l'implémenter en C++11, en utilisant Qt5 et la bibliothèque OpenGL pour Qt, et en compilant avec l'outil CMake.

La communauté Internet sur la bibliothèque Qt et son module OpenGL est très importante et de nombreux problèmes rencontrés ont pu être résolus facilement grâce à des recherches sur certains forums. L'outil souvent utilisé dans la réalisation de projet Qt est QMake qui permet une compilation simplifiée avec Qt. Mais nous avons choisi de prendre CMake comme nous savions déjà l'utiliser. CMake étant très peu utilisé pour la bibliothèque Qt, il est parfois difficile de trouver sur Internet de l'aide pour inclure des modules, par exemple le module QtXml. La plupart de la documentation relative à ce problème est très simplement géré dans QMake, mais il aura fallu beaucoup de recherches pour parvenir à trouver les bons noms de bibliothèques et de modules pour l'inclure avec CMake.

3.7 Bonus d'implémentation et possibilités d'évolution

Dans la dernière phase de notre implémentation, nous avons pu ajouter quelques fonctionnalités pour améliorer le logiciel et le rendre plus proches de certains logiciels plus professionnels.

Tout d'abord, à la demande de nos clients, nous avons ajouté la possibilité d'annuler la dernière action effectuée quand celle-ci agit sur l'état d'un objet, son ajout [OU SA SUPPRESSION]. Grâce à un stockage de lambda-expression,

permises par le C++11, chaque action est enregistrée. Lorsque l'utilisateur utilise le raccourci clavier Control+Z, la dernière fonction ajoutée au vecteur de lambda-expression est récupérée, et l'inverse de l'action est effectuée.

Nous avons également implémenté des fonctionnalités pour coller aux préférences de l'utilisateur. Par exemple, l'utilisateur peut choisir le durée qu'il souhaite imposer entre deux sauvegardes automatiques du logiciel, la couleur du fond de la scène, ou encore l'emplacement de la barre verticale dans laquelle sont listés les objets présents sur la scène. Il peut également modifier les raccourcis clavier pour les personnaliser.

La couleur des objets est également devenue un paramètre modifiable. Après sélection d'un objet, l'utilisateur peut, grâce au menu déroulant prévu à cet effet, en modifier la couleur. Cette modification sera effective dans la scène et lors des rendus, mais ne sera pas sauvegardée avec les autres caractéristiques de l'objet.

Enfin, nous avons mis en place un antialiasing pour la génération des rendus. [EXPLICATION ANTIALIASING]

Faute de temps, d'autres améliorations éventuelles du logiciel n'ont pas pu être mises en place, mais représentent d'éventuelles perspectives d'amélioration pour notre logiciel.

Ce projet a été conçu pour permettre son extensibilité, principalement au niveau des algorithmes de rendus et des possibilités de création de rendus divers. On pourrait alors réfléchir à implémenter de nouveaux algorithmes, existants ou à venir, pour tester et comparer leur qualité. On pourrait également imaginer d'autres rendus, comme un flipbook d'anaglyphes ou des autostéréogrammes dynamiques.

Bien que le logiciel permette à l'utilisateur de le personnaliser, certains raccourcis ou fonctionnalités n'ont pas été implémentés, comme la sélection multiple d'objets ou le clic droit sur un objet déjà sélectionné pour connaître d'éventuelles actions effectuables sur cet objet. Cette éventualité pourrait permettre de rendre le logiciel plus agréable à manipuler pour ses utilisateurs.

Un autre objectif potentiel serait le passage du logiciel sous un environnement MAC. Faute de machines de test, nous n'avons pas eu l'occasion d'implémenter de tester notre code avec ce système d'exploitation, mais nous pensons que la transition vers ce système d'exploitation pourrait s'avérer relativement simple.

Enfin, l'utilisation de la version 2.0 ES de la bibliothèque OpenGL offre la possibilité d'implémenter le logiciel sous d'autres plateformes, comme par exemple des tablettes graphiques ou des téléphones portables. On pourrait même éventuellement imaginer prendre des photos d'une scène environnante et la transformer grâce à d'autres algorithmes en des anaglyphes ou des autostéréogrammes.

Les possibilités d'évolution du logiciel Project3Donuts sont donc multiples, et nous espérons que celui-ci pourra être amélioré ou réutilisé par la suite dans de nouveaux projets.

4 Retour sur la gestion de projet

Dans cette partie, nous reviendrons plus en détails sur la partie gestion de projet de notre PFA, depuis sa préparation avec le cahier des charges jusqu'à sa réalisation.

4.1 Cahier des charges

Ce projet a été pour nous la première occasion de rédiger un cahier des charges formel. Dès notre premier rendez-vous, nos clients ont clairement exprimés leurs attentes vis à vis du contenu du cahier des charges. Ils nous ont également permis de comprendre que l'exhaustivité des besoins exprimés permet d'assurer le contrat entre l'équipe des programmeurs et les clients et de certifier à la fois un engagement pour un produit minimal et l'assurance que les demandes vis-à-vis de celui-ci ne pourront pas être revisitées.

Nos clients étant d'ores et déjà fixés sur le contenu qu'il souhaitait pour le logiciel, il a été assez facile pour nous de rédiger les besoins fonctionnels et non fonctionnels qui avaient été clairement énoncés. La principale difficulté que nous avons rencontré lors de la phase du cahier des charges a été la rédaction des parties Domaine et Etat de l'existant. La partie Domaine nous aura permis de nous familiariser avec le vocabulaire de la scène et des objets, et de revenir sur l'historique de la synthèse d'image. Toutefois, il nous a fallu faire le tri des informations importantes pour la bonne compréhension du sujet par la suite. Les recherches relatives à l'Existant étaient d'autant plus importantes qu'elles allaient permettre de choisir des algorithmes les plus performants pour l'obtention des rendus souhaités.

Les algorithmes relatifs aux anaglyphes concernent principalement le traitement des couleurs pour qu'aucun artefact n'apparaisse au moment de la visualisation finale. Pour les autostéréogrammes, le traitement d'une carte des profondeurs permet d'obtenir une image qui, lorsque l'on sait l'observer, fait apparaître un relief. Enfin, il n'existe pas d'algorithme particulier pour la génération de folioscopes. Seule une série de prises de vue d'une scène avec des angles d'observation proches peuvent permettre, si elles sont visualisées les unes à la suite des autres et suffisamment rapidement, de pouvoir imaginer un mouvement, et ainsi un relief.

La rédaction du cahier des charges aura été un exercice enrichissant car il nous aura appris l'importance d'un cahier des charges et de la recherche d'algorithmes existants pour éviter de devoir repartir de zéro à chaque nouveau projet. Ce document est réellement un contrat destiné à protéger à la fois les clients et les programmeurs, et l'exhaustivité des données qu'il contient est importante pour qu'aucun des deux parties ne puisse changer d'avis au moment de la partie Réalisation, du moins pas sans un accord commun.

4.2 Diagramme de Gantt prévisionnel

Une fois le cahier des charges mis en place et l'accord des clients donnés, la première étape du projet consistait à réfléchir au diagramme de Gantt prévisionnel pour les trois mois destinés à la réalisation du projet.

La priorité était donnée aux algorithmes de réalisation des différents rendus espérés grâce au logiciel, et plus particulièrement aux anaglyphes et aux autostéréogrammes. Il fallait revenir sur les algorithmes choisis et présentés dans le cahier des charges, approfondir les recherches pour être sûr de ne passer à côté d'aucun autre algorithme plus performant, et s'approprier le fonctionnement des algorithmes pour les comprendre puis les retranscrire.

En parallèle, d'autres personnes pouvaient travailler sur une partie plus proche du logiciel finale, à savoir la manipulation de la scène ou encore les parseurs de fichiers pour le chargement des objets.

Une fois les manipulations premières de la scène effectuées, les travaux prioritaires seraient le chargement et la sauvegarde de la scène grâce à des fichiers d'extension XML, la création de prises de vue simples ou de folioscopes à partir de la scène et leur sauvegarde, et enfin l'assemblage complet du logiciel avec une interface d'utilisation permettant d'utiliser les différents modules.

Un récapitulatif du diagramme de Gantt prévisionnel du projet est donné dans les figures 4.1 et 4.2.

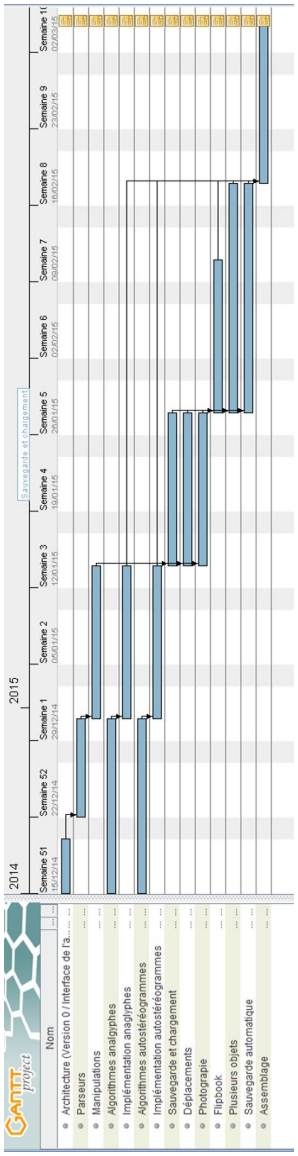

FIGURE 4.1 – Graphe temporel du déroulement du projet [15]

Pour déterminer les durées nécessaires à chaque partie de ce projet, nous avons dû réfléchir aux éventuelles recherches à réaliser au préalable, ainsi qu'à la complexité des tâches en les découpant en sous-parties. Nous avons également réfléchi aux éventuels examens qui auraient pu nous ralentir dans la réalisation en fonction des périodes.

Ainsi, pour une tâche telle que l'implémentation d'algorithme d'anaglyphes, il fallait dans un premier temps revenir et compléter les recherches faites sur l'état de l'existant. En effet, comme nous l'avons vu dans la partie présentation du projet, de nombreuses études ont été effectuées pour mettre en place des algorithmes plus ou moins performants

15. Réalisé grâce au logiciel GanttProject : `http://www.ganttproject.biz/`
16. Réalisé grâce au logiciel GanttProject : `http://www.ganttproject.biz/`

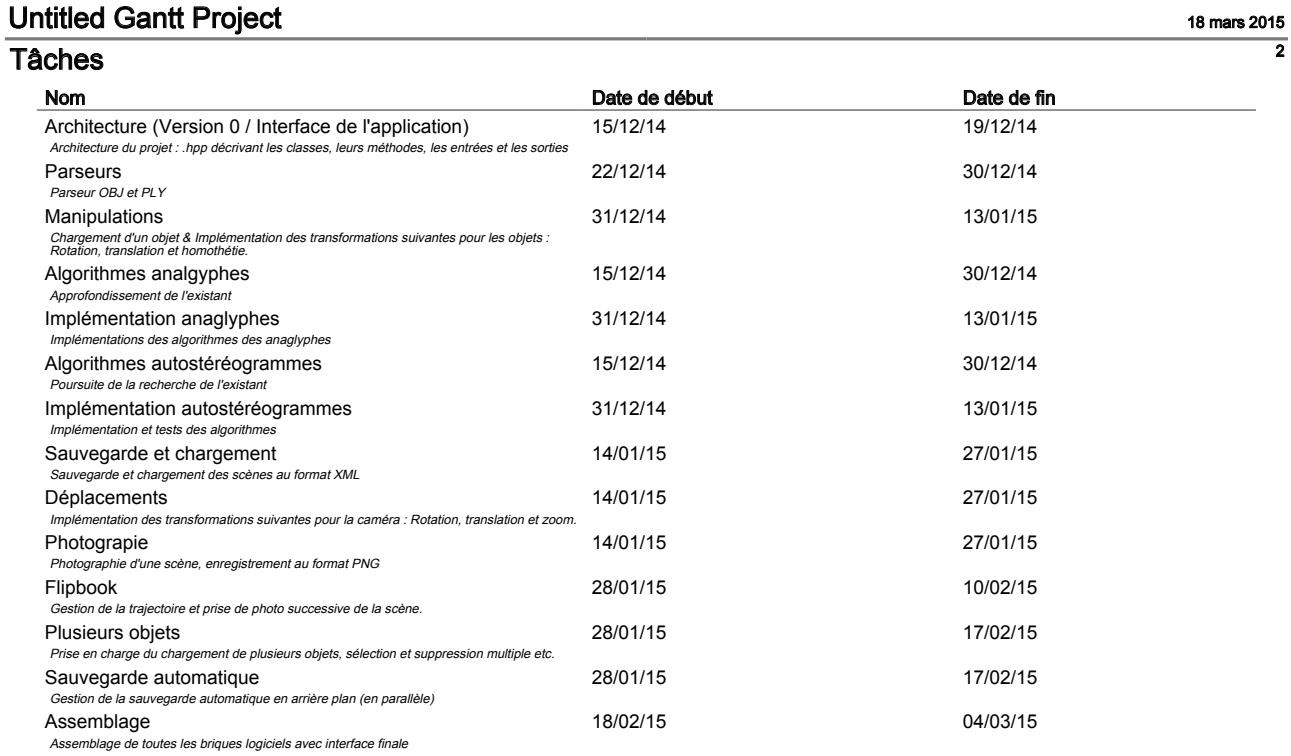

FIGURE 4.2 – Récapitulatif des tâches et des périodes par tâche [16]

capables de générer des anaglyphes. Une fois le choix des algorithmes effectués, l'implémentation de celui-ci en C++ pourrait être effectuée.

Au final, la période déterminée pour la recherche d'algorithmes et l'implémentation de ces algorithmes était d'environ un mois pour une équipe de deux personnes.

Un autre exemple de période déterminée est celle destinée à la sauvegarde et au chargement des scènes. Nous avions d'ores et déjà un fichier XML modèle sur lequel s'appuyer, ce qui a permis de passer rapidement à l'implémentation. La sauvegarde consistant simplement en une écriture dans un fichier, seul le chargement demandait des recherches pour déterminer la meilleure bibliothèque pour effectuer le découpage d'un fichier XML. La période déterminée pour cette tâche était de deux semaines pour une équipe de deux personnes.

4.3 Bilan final

Comme nous l'avons expliqué dans la partie Réalisation, le diagramme de Gantt a dû être remanié au fil de l'avancement du projet car certaines tâches, plus faciles que prévues, ont été anticipées alors que d'autres, posant plus de problèmes, ont été allongées. Le diagramme de Gantt final est donné page suivante et peut être comparé au diagramme

fourni dans les figures 4.3 et 4.4.

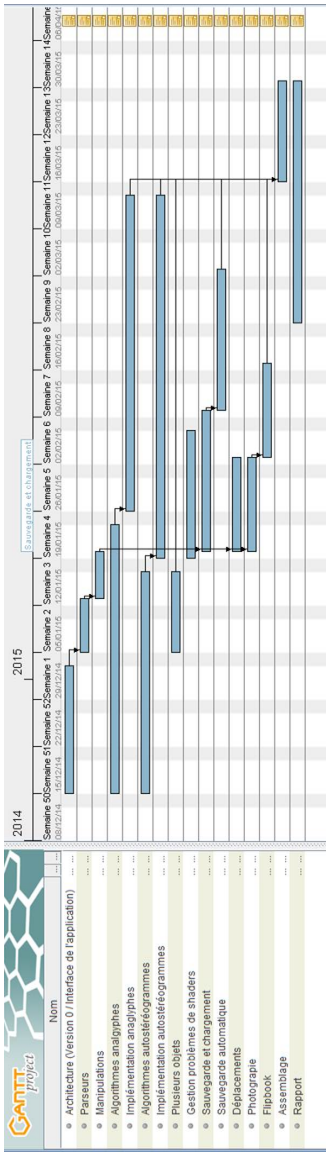

FIGURE 4.3 – Graphe temporel du déroulement du projet [17]

La principale modification consiste ainsi en l'allongement de la période destinée à la recherche d'algorithmes pour les rendus et à leur implémentation. Les raisons de cette différence ont été expliquées dans la partie Difficultés rencontrées ci-dessus. Les manipulations des objets et de la scène ont été effectuées plus rapidement que prévu, mais une période de retour en arrière à cause des shaders a été ajoutée, allongeant la période de travail sur le chargement et la sauvegarde de la scène qui a été momentanément interrompue le temps de régler ce problème.

17. Réalisé grâce au logiciel GanttProject : http ://www.ganttproject.biz/
18. Réalisé grâce au logiciel GanttProject : http ://www.ganttproject.biz/

Untitled Gantt Project 18 mars 2015

2

Tâches

Nom	Date de début	Date de fin
Architecture (Version 0 / Interface de l'application) *Architecture du projet : .hpp décrivant les classes, leurs méthodes, les entrées et les sorties*	15/12/14	02/01/15
Parseurs *Parseur OBJ et PLY*	05/01/15	12/01/15
Manipulations *Chargement d'un objet & Implémentation des transformations suivantes pour les objets : Rotation, translation et homothétie.*	13/01/15	19/01/15
Algorithmes analgyphes *Approfondissement de l'existant*	15/12/14	23/01/15
Implémentation anaglyphes *Implémentations des algorithmes des anaglyphes*	26/01/15	13/03/15
Algorithmes autostéréogrammes *Poursuite de la recherche de l'existant*	15/12/14	16/01/15
Implémentation autostéréogrammes *Implémentation et tests des algorithmes*	19/01/15	13/03/15
Plusieurs objets *Prise en charge du chargement de plusieurs objets, sélection et suppression multiple etc.*	05/01/15	16/01/15
Gestion problèmes de shaders *Problème de compatibilité des shaders sous Linux, recherche de solution et prise de décision*	19/01/15	06/02/15
Sauvegarde et chargement *Sauvegarde et chargement des scènes au format XML*	20/01/15	09/02/15
Sauvegarde automatique *Gestion de la sauvegarde automatique en arrière plan (en parallèle)*	10/02/15	02/03/15
Déplacements *Implémentation des transformations suivantes pour la caméra : Rotation, translation et zoom.*	20/01/15	02/02/15
Photograpie *Photographie d'une scène & Enregistrement au format PNG*	20/01/15	02/02/15
Flipbook *Gestion de la trajectoire et prise de photo successive de la scène.*	03/02/15	16/02/15
Assemblage *Assemblage de toutes les briques logiciels et interface finale*	16/03/15	30/03/15
Rapport *Rédaction du rapport final*	23/02/15	30/03/15

FIGURE 4.4 – Récapitulatif des tâches et des périodes par tâche [18]

Malgré ces débordements dans le temps, la réalisation du projet se sera achevée dans les temps, le dernier mois ayant permis d'intégrer les algorithmes au logiciel, de compléter et de parfaire des fonctionnalités.

Au final, l'ensemble des manipulations de la scène et de ses objets qui avaient été promises dans le cahier des charges ont été implémentées. De même, l'ensemble des rendus prévu est générable à partir de la scène, et deux algorithmes de génération sont proposés pour les autostéréogrammes et les anaglyphes alors qu'un seul avait été cité pour l'autostéréogramme dans le cahier des charges, et que seul le traitement de couleurs pour l'impression avait été initialement prévu pour m'anaglyphe (et pas la saturation).

Au niveau des besoins non fonctionnels, la portabilité aura été respectée malgré l'utilisation des shaders qui auraient pu poser problème sur certaines machines. Grâce aux nombreux modules proposés par la bibliothèque Qt, portable et très complète, des solutions auront été trouvées pour l'ensemble des modules et des cas d'utilisation sans avoir à sacrifier de fonctionnalité pour permettre l'utilisation du logiciel aussi bien sous Windows que sous Linux.

Bien que l'indication ne soit pas donnée dans le logiciel, l'utilisation de Project3Donuts en parallèle de l'utilisation du logiciel nous a permis de vérifier la fluidité du logiciel. Grâce notamment au modèle 'happy.ply' présenté dans le cahier des charges, les valeurs de frames par seconde données dans le cahier des charges ont été validées.

Enfin, la maintenabilité du logiciel sera également possible grâce au choix des outils et à l'architecture du projet. Tout d'abord, l'utilisation de Qt5, qui est actuellement la plus récente de Qt, laisse envisager que le logiciel pourra être utilisé longtemps sans avoir à migrer vers une nouvelle version de la bibliothèque. Ensuite, l'utilisation de OpenGL ES 2.0, qui est la version d'OpenGL de base avec Qt5, pourrait éventuellement permettre de générer une application mobile du logiciel. L'architecture a également été pensée pour permettre cette maintenabilité. En effet, comme le prouvent les deux algorithmes

Ce bilan positif montre que l'ensemble des besoins fonctionnels et non fonctionnels ciblés dans le cahier des charges ont été implémentés avec succès. Nous espérons que ce projet sera réutilisé et maintenu, puisqu'il a été conçu dans cet optique. Il sera éventuellement possible de générer une application mobile à partir du code existant, ou d'ajouter et de tester d'autres algorithmes ou d'autre rendus possible à partir d'une scène en trois dimensions. Nous espérons également que nos clients auront été satisfaits du travail réalisé et du logiciel final.

Malgré la réussite globale du projet, l'exercice du PFA aura présenté quelques difficultés dans la gestion de projet.

Dès le début de la période de réalisation, nous devions déterminer les durées et l'ordonnancement des tâches au cours du projet. Grâce à la réflexion apportée par le cahier des charges, il a été plus aisé de connaître les tâches priotaires et de les estimer. Bien que nous ayons essayé de planifier notre temps en fonction de nos autres obligations à l'école, certaines semaines ont été plus fructueuses que d'autres et nous aurons permis d'avancer plus vite que prévu. A l'inverse, des difficultés imprévues, des difficultés de compréhension ou encore des retours en arrière nous ont retardé à d'autres moments.

Travailler dans une équipe nombreuse aura été bénéfique pour notre apprentissage de la gestion de projet. En effet, habitués dans le cadre de nos études à réaliser des projets par équipe de trois ou quatre, la répartition des tâches et la place de chacun dans l'équipe est plus facile à trouver. Dans une équipe de sept personnes, chaque tâche est répartie entre plusieurs personnes, et chacun doit faire sa part dans chaque tâche. Chacun ayant des rythmes de travail différent, cette expérience nous a montré à quel point il est important que chacun ait son rôle, sa place, pour qu'il puisse avancer à son rythme.

5 Conclusion

apports du projet connaissances acquises compétences acquises pour le cahier des charges compétences acquises pour la gestion de projet avantage d'un client qui nous a suivi tout du long retour sur l'utilisation d'une forme de méthode agile (plus ou moins)

Le déroulement de ce projet nous aura permis étape par étape de prendre part à la réalisation d'un logiciel. Le cahier des charges étant un logiciel que nous n'avions jamais eu l'occasion de pratiquer, l'implication de nos clients dans notre avancement nous aura permis d'être rigoureux et de comprendre l'intérêt d'un tel contrat entre le client et les programmeurs. Au cours de la partie programmation, nous avons pu nous rendre compte de la difficulté à estimer la durée d'une tâche. L'importance du cahier des charges nous est là encore apparue, car cette phase de recherche permet de se renseigner sur le domaine et l'existant propre au sujet à traiter, et de choisir d'éventuels algorithmes pour le futur logiciel.

Le PFA aura également pour nous été l'occasion de prendre part à la réalisation complète d'un projet informatique, aussi bien sur la forme (interface graphique du logiciel) que sur le fond (scène en trois dimensions et algorithmes des rendus). L'ampleur de l'exercice nous aura permis de faire face à un exercice plus proche de ceux que nous aurons à réaliser en entreprise. Il nous aura appris à partir d'une demande, à bâtir le cahier des charges correspondant, puis à construire à partir de rien le logiciel souhaité en respectant ce cahier. Le travail complet de recherche pour les bibliothèques et les méthodes à utiliser aura été formateur car il nous aura appris l'auto-formation et la recherche de solutions.

Le suivi régulier du projet par nos clients, souhaité par notre équipe dans l'idée de l'utilisation d'une méthode de type agile, nous aura permis au fur et à mesure de présenter l'avancée de notre travail et de corriger les directions prises pour satisfaire les souhaits de nos clients, ou pour ajouter d'éventuels fonctionnalités comme par exemple l'annulation de la dernière action effectuée.

Le PFA a donc été une expérience très enrichissante puisqu'elle aura permis de travailler dans des conditions proches de celles que nous pourrions avoir en entreprise.

Références

[1] Steer W. A. Stereograms : Technical details. http://www.techmind.org/stereo/stech.html, Juin 2002.

[2] Eric Dubois. A projection method to generate anaglyph stereo images. *IEEE International Conference*, 3 :1661–1664, 2001.

[3] Eric Dubois. Conversion of a stereo pair to anaglyph with the least-squares projection method. http://www.site.uottawa.ca/~edubois/anaglyph/LeastSquaresHowToPhotoshop.pdf, March 2009.

[4] Beene G. Stereogram algorithms – basics. http://www.garybeene.com/stereo/rds-alg_basic.htm, 2006.

[5] Stuart Inglis, Harold W. Thimbleby, and Ian H. Witten. Displaying 3d images : Algorithms for single image random dot stereograms. *IEEE Computer*, 27 :38–48, 1994.

[6] Elena Patana, Ilia Safonov, and Michael Rychagov. Adaptive 3d color anaglyph generation for printing. *GraphiCon'2012 Conference*, Conference Proceedings :55–60, 2012.

[7] David Romeuf. Création d'un anaglyphe sans rivalité colorée transformation des couleurs originales du couple stéréophotographique vers des couleurs adéquates pour l'anaglyphe méthode de transformation de eric dubois. http://www.david-romeuf.fr/3D/Anaglyphes/TCAnaglypheLSDubois/TransformationCouleursPourAnaglyphe.html, Juillet 2008.

[8] Alkhadour W, Ipson S, Zraqou J, Qahwaji R, and Haigh J. Creating a color anaglyph from a pseudo-stereo pair of images. *4th ICIT*, pages –, 2009.

[9] Andrew J. Woods and Chris R. Harris. Comparing levels of crosstalk with red/cyan, blue/yellow, and green/magenta anaglyph 3d glasses. *Proceedings of SPIE Stereoscopic Displays and Applications XXI*, 7253 :0Q1–0Q12, 2010.