
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	4
2	Présentation du projet	5
2.1	Domaine	5
2.2	Sujet du projet	6
2.3	Présentation des rendus souhaités	7
2.3.1	Les flipbooks	7
2.3.2	Les autostéréogrammes	7
2.3.3	Les anaglyphes	8
2.4	Résumé du cahier des charges	10
3	Déroulement de la réalisation technique	12
3.1	Choix de programmation	12
3.2	Rendu 2D	13
3.3	Choix des algorithmes d'anaglyphes	14
3.3.1	La génération des anaglyphes et ses défauts	14
3.3.2	La méthode Photoshop	15
3.3.3	La méthode des moindres carrés avec correction gamma	16
3.3.4	La méthode des moindres carrés avec saturation	16
3.3.5	Comparaison des différents algorithmes	16
3.3.6	L'impression des anaglyphes	17
3.4	Choix des algorithmes d'autostéréogrammes	17
3.4.1	Algorithme de base (Witten, Inglis, Thimbleby)	17
3.4.2	Algorithme de W. A. Steer	18
3.5	Réalisation du projet	18
3.5.1	Présentation générale de la réalisation	18
3.5.2	Les algorithmes du logiciel	19
3.5.3	La manipulation de la scène	20
3.5.4	Les sauvegardes et chargement de la scène	21
3.5.5	Architecture finale du projet	22
3.6	Test des fonctionnalités	23
3.7	Difficultés rencontrées	24
3.7.1	Implémentation des algorithmes	24
3.7.2	L'utilisation de shaders	25
3.7.3	Les outils utilisés pour la réalisation du projet	26
3.8	Bonus d'implémentation et possibilités d'évolution	26
4	Retour sur la gestion de projet	28
4.1	Cahier des charges	28
4.2	Diagramme de Gantt prévisionnel	28
4.3	Bilan final	30
5	Conclusion	35
A	Notice d'installation du logiciel	36
B	Notice d'utilisation du logiciel	37
C	Conventions de codage	39

D Exemple de fichier de chargement	40
E Cahier des charges	42

1 Introduction

Le projet PFA a pour objectif de faire découvrir aux élèves la gestion d'un projet depuis la création du cahier des charges jusqu'à l'implémentation en ayant affaire à des clients réels. C'est le premier projet du cursus se réalisant avec un groupe de taille conséquente (en moyenne 7 personnes) et dans lequel les élèves établissent les spécifications du rendu en échangeant avec le client. Le travail se décompose en deux phases : une première phase durant le premier semestre pour établir les besoins, et les présenter de manière formelle dans le cahier des charges, puis une deuxième phase au cours du second semestre pour concrétiser ces besoins en réalisant ce qui est spécifié.

Le projet PFA effectué : De la 3D à la 2D, est un projet d'imagerie numérique. L'objectif est de réaliser une scène 3D constituée de un ou plusieurs modèles 3D pour ensuite générer des vues en deux dimensions de celle-ci, comme présentée dans la figure 1.1. Ces dernières seront utilisées pour obtenir une illusion en trois dimensions en se basant sur des principes connus. Les algorithmes étudiés au cours de ce projet permettent de générer des anaglyphes, autostéréogrammes et folioscopes, également appelés flipbooks, qui seront présentés plus en détails dans la suite du rapport.

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 effectués 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.

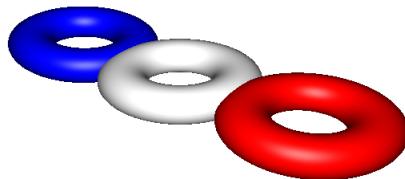


FIGURE 1.1 – Rendu classique du projet

2 Présentation du projet

Cette partie présente le domaine de la synthèse d'image ainsi que le sujet du projet. Nous y présenterons également les besoins du futur logiciel qui ont été déterminés à partir de ces recherches.

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 Meshlab¹ et Blender² 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 inexiste sur un point caché. En tenant compte de cette lumière, on peut obtenir une image comme présentée sur la figure 2.1.

1. <http://meshlab.sourceforge.net/>

2. <http://www.blender.org/>

3. <http://linut.free.fr/omgsp10kuberwebloglolz0r/?2010/02/01/93-raytracer-que-la-lumiere-soit>

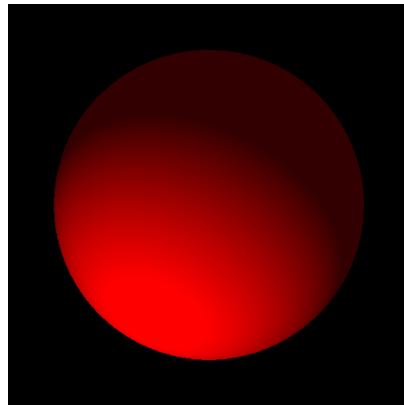


FIGURE 2.1 – Application d'une lumière diffuse à une sphère rouge³

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'inséreront 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

Cette partie présente les trois rendus qui ont été implémentés pour le logiciel : l'anaglyphe, l'autostéréogramme et le folioscope.

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.

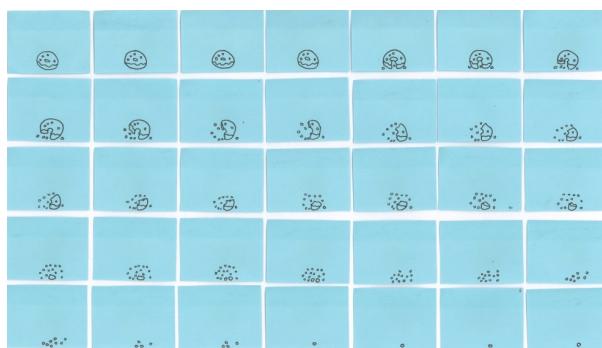


FIGURE 2.2 – Story-board d'un flipbook⁴

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.

4. <http://tracie.liu.blogspot.fr/2010/08/flipbook-storyboard.html>
 5. <http://www.techmind.org/stereo/geometry.gif>

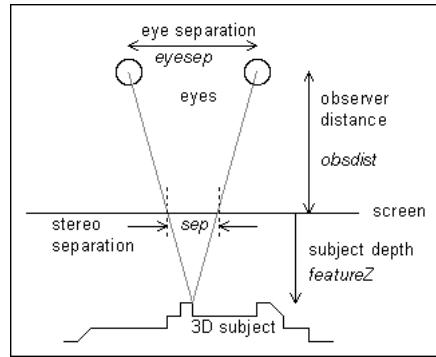


FIGURE 2.3 – Visualisation d'un autostéréogramme en vision parallèle⁵

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.

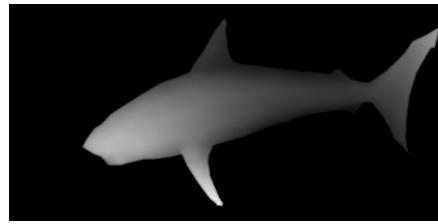


FIGURE 2.4 – Carte des profondeurs⁶

Il existe des algorithmes de génération d'autostéréogrammes très simples, comme celui proposé par Gary Beene [?]. Cependant un algorithme aussi peu optimisé produit des autostéréogrammes défectueux, 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 [?], et celui proposé par W. A. Steer [?], 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.

6. <http://en.wikipedia.org/wiki/Autostereogram>

7. <http://en.wikipedia.org/wiki/Autostereogram>

8. <http://zour.deviantart.com/art/Wernigerode-Boulevard-Dubois-Anaglyph-HDR-3D-276542278>



FIGURE 2.5 – Autostéréogramme obtenu⁷



FIGURE 2.6 – Anaglyphe⁸

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 diffèrent par la qualité de l'anaglyphe en sortie en diminuant le nombre d'artefacts [?] ou en améliorant le rendu pour l'impression [?] par exemple.

10. <http://www.david-romeuf.fr/3D/Anaglyphes/TCAAnaglypheLSDubois/TransformationCouleursPourAnaglyphe.html>

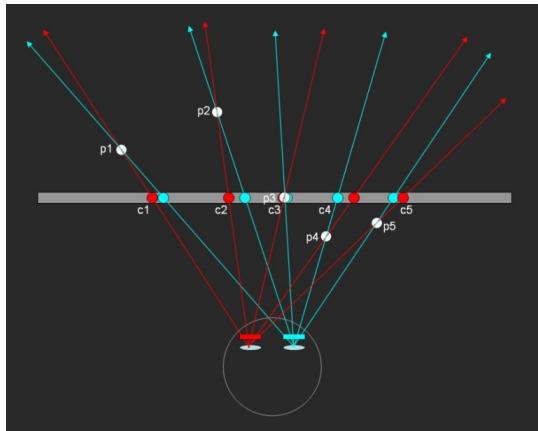


FIGURE 2.7 – Le décalage de la partie rouge et cyan permet à l’œil de percevoir l’image non plus dans un plan XY mais dans l’espace¹⁰

2.4 Résumé du cahier des charges

A partir de nos recherches sur le domaine et les algorithmes existants, nous avons rédigé un cahier des charges pour définir les besoins fonctionnels et non fonctionnels de notre projet. Ce 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’utilisation 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 apporté des contraintes quand à la réalisation du logiciel, par exemple 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 compte 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 le Cahier des Charges, 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 donnée dans la Figure 2.8.

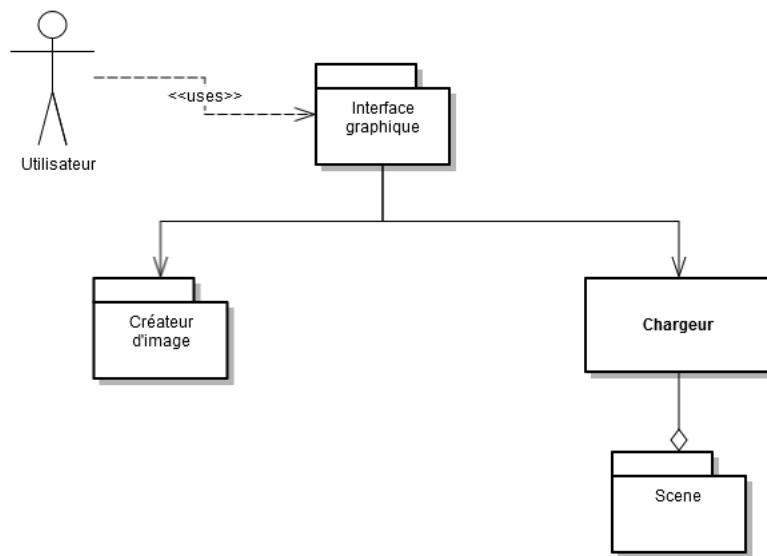


FIGURE 2.8 – Diagramme des paquetages, architecture globale du projet¹¹

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 2.9. 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 Crédit de l'architecture, qui permet la création des différents rendus que le logiciel permet d'obtenir. L'architecture de ce paquetage est donnée dans la Figure 2.10.

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

11. Réalisé grâce au logiciel Gliffy : www.gliffy.com

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

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

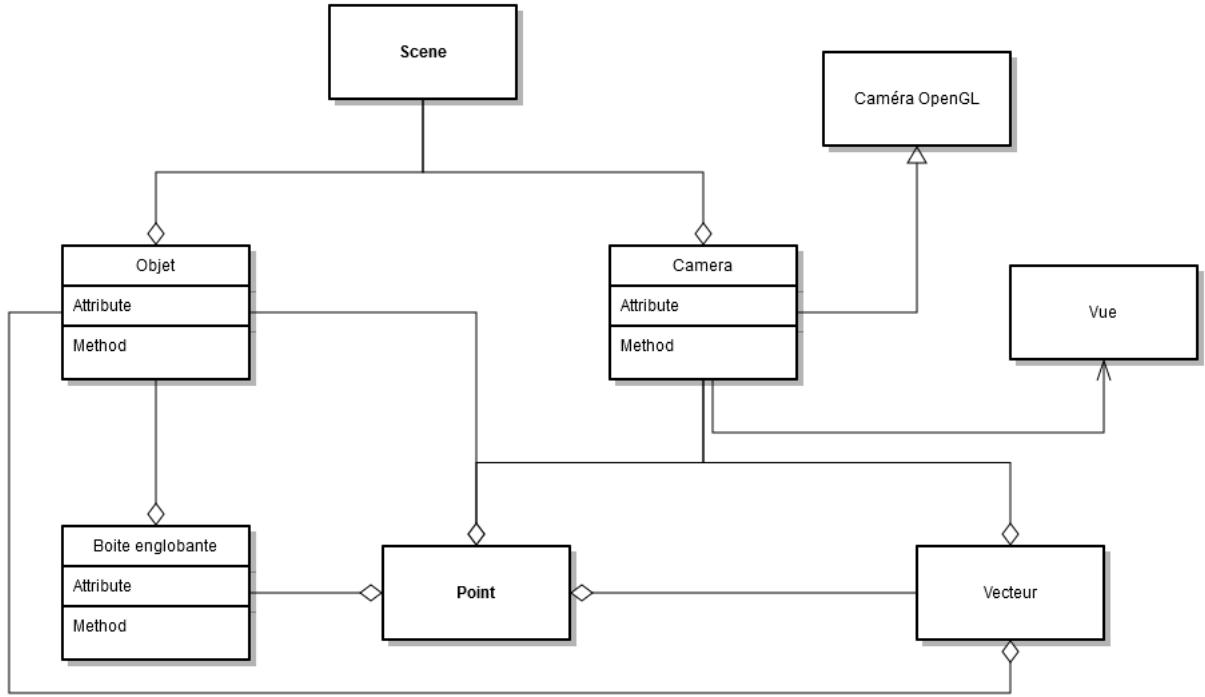


FIGURE 2.9 – Paquetage Scene en début du projet¹²

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, ainsi que les conventions de codage qui ont été utilisées.

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. 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 portables, elles conviennent parfaitement à l'implémentation que nous souhaitons réaliser.

La version 5 de Qt est la plus récente actuellement, mais malgré sa jeunesse elle est devenue au fil des modifications suffisamment stable pour pouvoir être utilisée sans problème, et elle dispose d'une communauté Internet active prête à

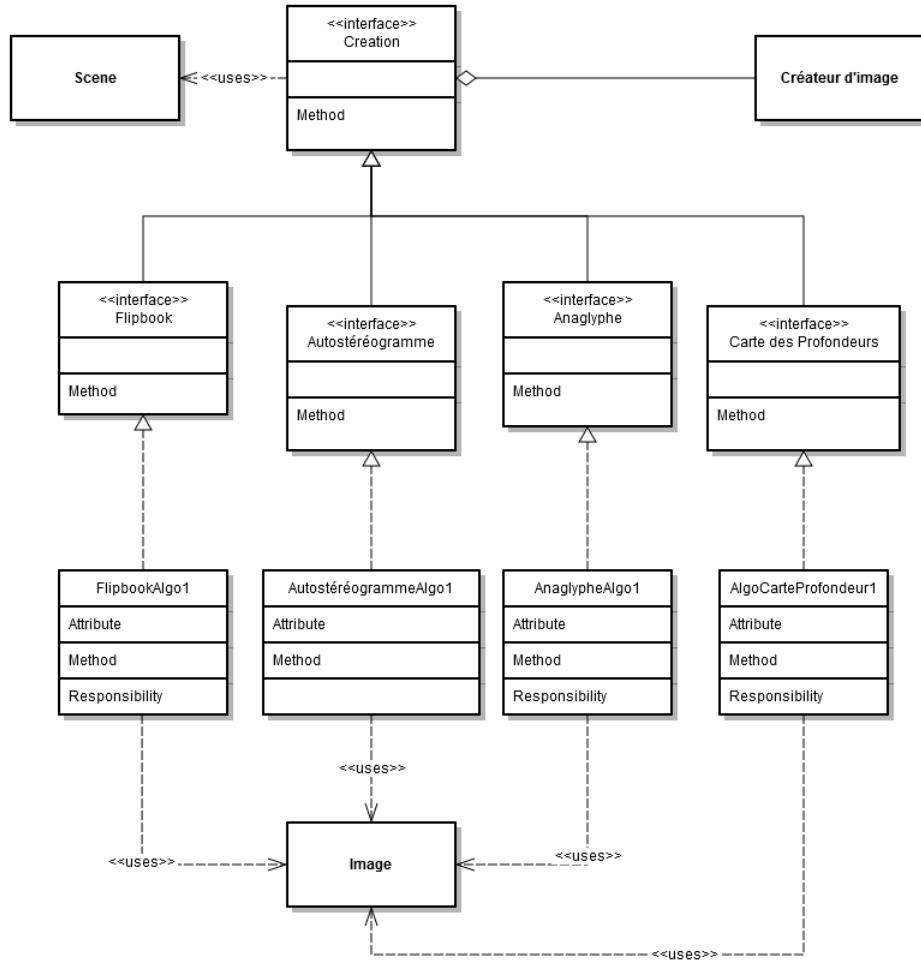


FIGURE 2.10 – Paquetage Crédit en début de projet¹³

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 dépendant trop fortement de QtCreator, nous avons préféré ne pas l'utiliser.

3.2 Rendu 2D

3.2.1 Rendu avec OpenGL

L'ensemble des rendus est réalisé avec OpenGL et des shaders écrits en GLSL. Lorsque un objet est chargé dans la scène, le chargeur d'objet remplit des tableaux tels que la position des sommets, les couleurs et les normales associées

aux sommets, et les indices associés aux sommets de chaque faces.

Le problème est que certains fichiers d'objet, notamment les fichiers avec le format .ply, ne possèdent pas d'information sur les couleurs et les normales des sommets. Or c'est information sont indispensable pour avoir un beau rendu 2D. Ainsi si ces informations sont manquantes, la couleur de l'objet est mise à grise par défaut et les normales sont recalculées à partir de l'orientation des faces.

3.2.2 Le VAO

Toutes les informations sur l'objet sont placées dans un VAO (Vertex Array Object) et transmises une seul fois à la carte graphique lors de l'initialisation. Ensuite ce VAO est traité pour chaque rendu par des shaders.

Cette technique présente l'avantage d'envoyer le VAO une seul fois à la carte graphique alors qu'avec la méthode du rendu sans shader, les données de l'objet sont envoyées à chaque rendu, ce qui le ralenti considérablement.

3.2.3 Les shaders

Sans shaders les rendus n'était pas esthétiques (figure 3.1), l'ombrage n'était pas doux et à la frontière entre chaque face était clairement visible.

Les shaders ont pu remédier à cela grâce à la technique du *per-pixel lighting* : l'ombrage n'est plus calculé face par face mais pixel par pixel, ce qui offre une plus grande précision et ce qui permet de lisser l'ombre.

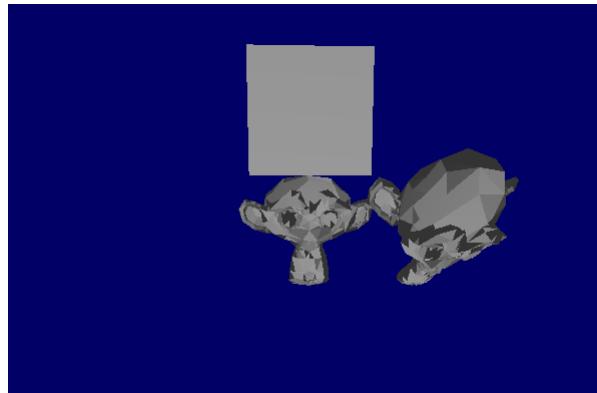


FIGURE 3.1 – Rendu sans shaders qui avait été effectué au début du développement du logiciel. L'ombrage est très chaotique.

3.2.4 La spécularité

Pour améliorer le rendu, il a été choisi de rajouter une spécularité à l'aide du shader. Cette effet permet de simuler les réflexions de lumière sur l'objet (figure 3.2). Le shader calcul pour cela la reflection de la lumière sur la surface.

3.2.5 L'anti-aliasing

3.3 Choix des algorithmes d'anaglyphes

Cette partie présentera les différents algorithmes qui ont été choisis pour générer des anaglyphes dans le logiciel.

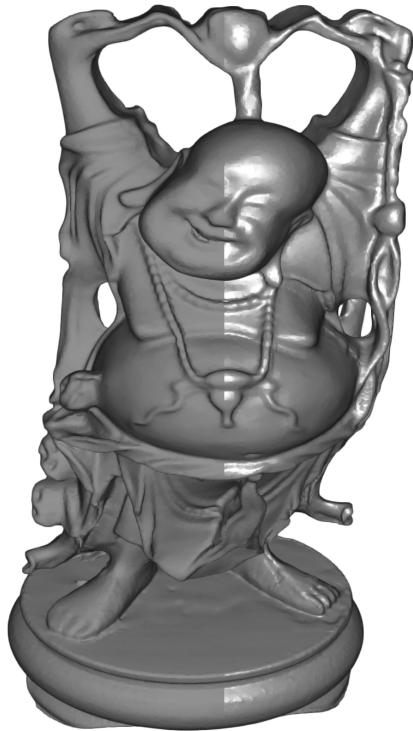


FIGURE 3.2 – Rendu avec l'ombrage de base à gauche et avec la spécularité à droite.

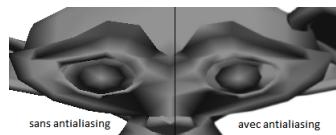


FIGURE 3.3 – Rendu de base à gauche et avec anti-aliasing à droite.

3.3.1 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 de 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 l'application de deux filtres rouge et cyan (ou deux autres couleurs correspondants à celles des lunettes). 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 [?] : 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 problématiques 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'algorithme qui rendra l'anaglyphe de meilleure qualité.

3.3.2 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 [?]. Un exemple de rendu obtenu avec cette méthode est présenté dans la Figure 3.3.

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 récupérées 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 récupérées, additionnées deux à deux et réinsérées dans les canaux de couleurs du pixel correspondant dans l'image résultat.

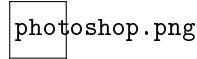


FIGURE 3.4 – Anaglyphe obtenu avec la méthode Photoshop

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

3.3.3 La méthode des moindres carrés avec correction gamma

Le deuxième algorithme implémenté est basé sur la méthode des moindres carrés présentée dans l'article [?] par Eric Dubois. Dans cet article, comme David Romeuf l'explique [?] 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. Un exemple de rendu obtenu avec cette méthode est présenté dans la Figure ??.

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 [?] : 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.

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

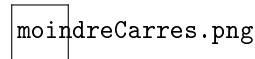


FIGURE 3.5 – Anaglyphe obtenu avec la méthode des moindres carrés

3.3.4 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 résultat -> l'algo d'origine avec saturation -> la diminution de saturation, atténue trop les couleurs : moins bonne 3D mais moins fatigant pour les yeux

3.3.5 Comparaison des différents algorithmes

En comparant les algorithmes, bien que la méthode Photoshop présente un résultat satisfaisant, il semblerait que la méthode de Dubois [?] 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 Photoshop.

3.3.6 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'algorithme de Dubois avec la méthode des moindres carrés [?], 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 [?], le support d'affichage est aussi important : dans ce cas, le type de papier rentre aussi en compte dans la qualité de l'anaglyphe.

3.4 Choix des algorithmes d'autostéréogrammes

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

3.4.1 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, comme le montre la figure 3.5 : 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.

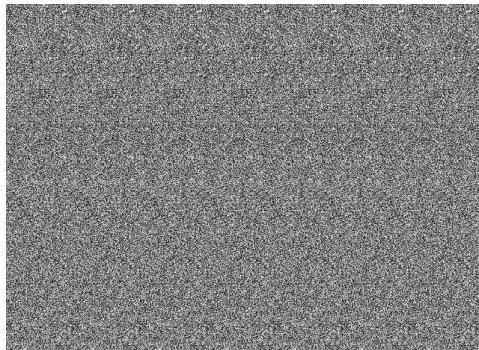


FIGURE 3.6 – Application d'une lumière diffuse à une sphère rouge

3.4.2 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 comme le montre la figure 3.6. De plus, l'utilisation conjointe d'une texture et du suréchantillonnage lissoit 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.

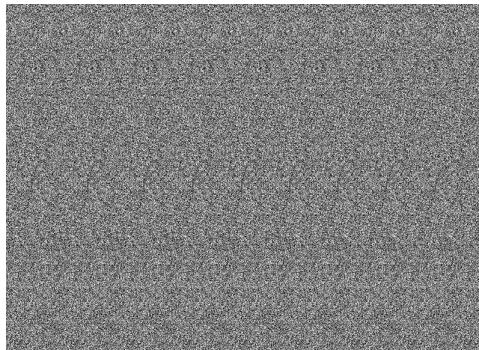


FIGURE 3.7 – Application d'une lumière diffuse à une sphère rouge

3.5 Réalisation du projet

Cette partie détaillera la réalisation technique du logiciel Project3Donuts.

3.5.1 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.

L'implémentation des algorithmes de rendus étant centrale au projet, nous avons dès le début de la réalisation mis en place deux équipes de deux personnes pour les deux rendus principaux : les anaglyphes et les autostéréogrammes. Les folioscopes quand à eux ne demandaient pas d'algorithmes particulier, 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.

Le travail sur la scène s'est basé sur le prototype généré lors de la phase du cahier des charges. Il a ainsi été amélioré pour intégrer les parseurs de fichiers objets qui allaient être nécessaires pour la génération des éléments de la scène, et rendre plus agréable la manipulation des objets et les déplacements dans la scène.

Malheureusement, l'implémentation des algorithmes s'est révélée plus longue que nous l'avions initialement prévue. Le mois prévisionnel n'étant pas uniquement dédié au projet PFA, les différents imprévus et un jugé peut-être trop léger de la difficulté que représentait cette étape ont engendré un retard par rapport au prévision. De plus amples explications seront données dans la partie "Difficultés rencontrées" de ce rapport. Au final, les recherches concernant les autostéréogrammes et anaglyphes auront pris deux mois et demi au lieu d'un seul. Deux mois ont servi à la réalisation de tests sur des prototypes pour évaluer la qualité des rendus, le demi mois concerne l'intégration au reste du logiciel avec d'autres tests pour trouver les réglages optimaux des algorithmes.

En parallèle, la troisième équipe, appuyée parfois par des membres des autres groupes 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, même 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.

3.5.2 Les algorithmes du logiciel

Pour pouvoir permettre à l'utilisateur de notre logiciel d'obtenir des anaglyphes ou des autostéréogrammes il a fallu prospecter sur internet afin de trouver suffisamment d'articles scientifiques et de site traitant le sujet pour établir des comparaisons et choisir parmi toutes les solutions proposées celles qui offraient le meilleur résultat.

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 formatrice car elle nous aura appris à utiliser les bons outils (comme google scholar) et les bons mots-clés pour rendre nos recherches fructueuses.

Une fois les algorithmes choisis, l'utilisation de cartes de profondeur et de paires d'images stéréoscopiques trouvées sur Internet ont permis de faire des tests sur des prototypes indépendants du logiciel final. Leur implémentation s'est faite durant le dernier du projet. Il a donc fallu définir les entrées et les sorties pour chaque méthodes, afin de satisfaire l'architecture du projet et faciliter par la suite l'intégration d'autres algorithmes dans le logiciel. La partie gérant la scène

est considérée comme une boîte noire avec laquelle les algorithmes de rendus interagissent. Seul les entrées et les sorties sont connues ce qui permet de garantir la modularité de la partie rendue.

Malgré les tests de pré-intégration fait sur les prototypes, des problèmes ont été rencontrés au moment de l'implémentation. Par exemple, pour les anaglyphes, le passage à l'utilisation de prises de vues issues de scène non-réelle donnait des résultats de moins bonnes qualités. Trouver le bon angle pour les couples d'images stéréoscopiques n'était pas non plus une chose aisée. 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 contours des objets moins abruptes, mais cette solution n'a pas été implémentée dans notre logiciel.

Au final, le travail sur les algorithmes fut très enrichissant car bien plus théorique que les autres parties du projet. La recherche et la compréhension des articles est une part plus importante du travail que leur intégration au projet. D'autant plus que tous les articles ne présentaient pas dans le détail leurs solutions, ce qui obligeait à faire des recherches supplémentaires pour trouver comment s'y prendre. L'obtention d'un rendu à partir de la scène est montré dans la figure ??.

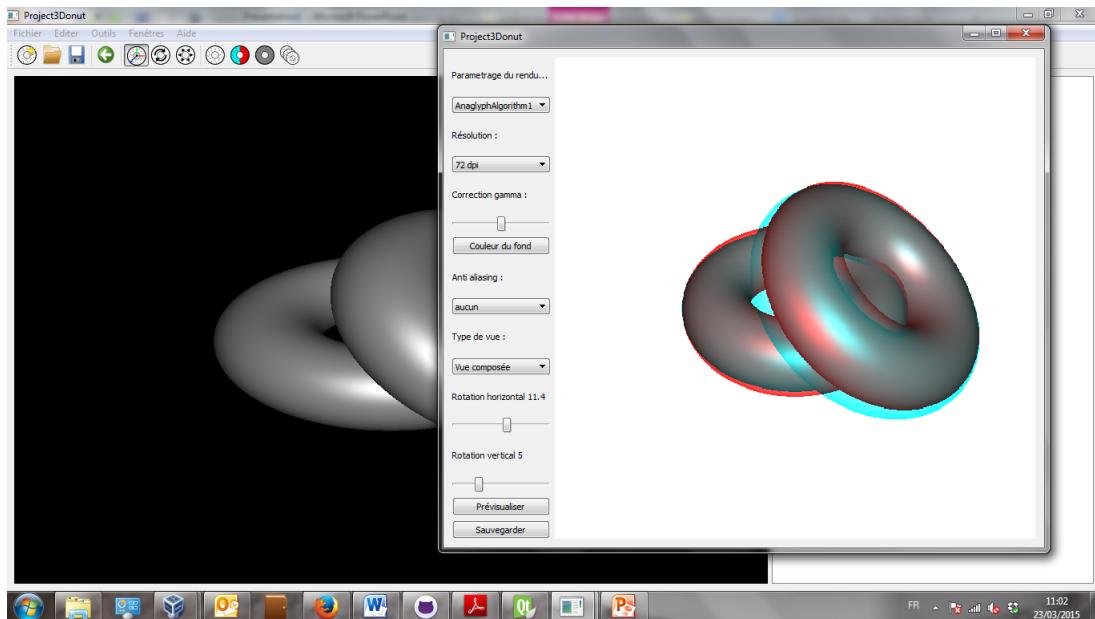


FIGURE 3.8 – Obtention d'un anaglyphe dans le logiciel

3.5.3 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 avaient 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 implementées 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 3.8.

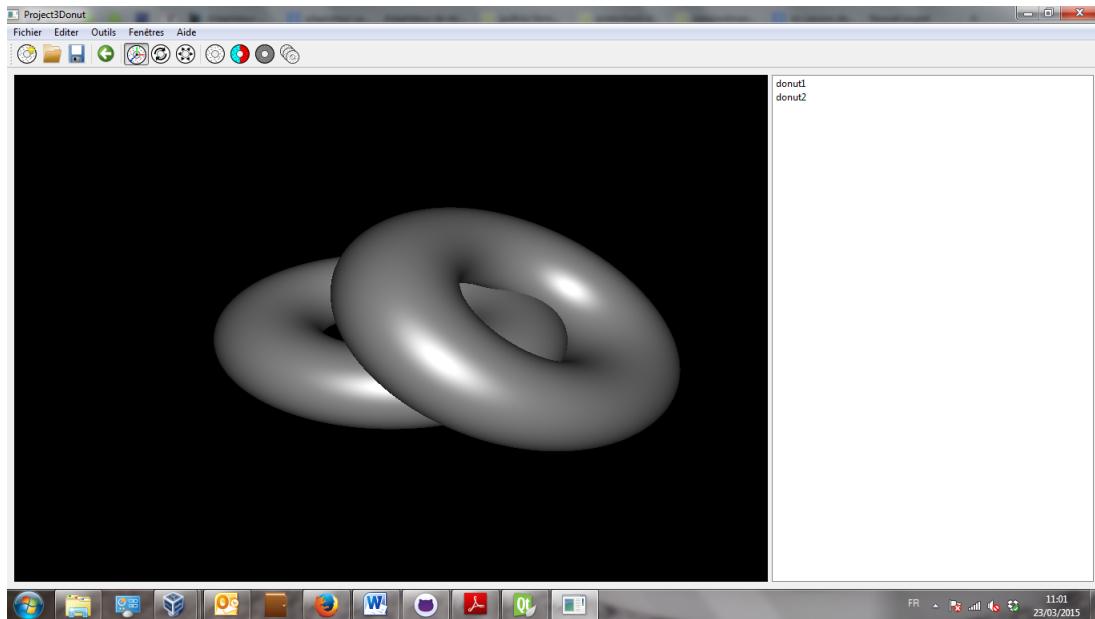


FIGURE 3.9 – Visualisation de la scène dans le logiciel

3.5.4 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 QDom 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 respectant 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 permettent 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.

3.5.5 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 3.9, le Chargeur n'est plus le point d'entrée du paquetage Scene.

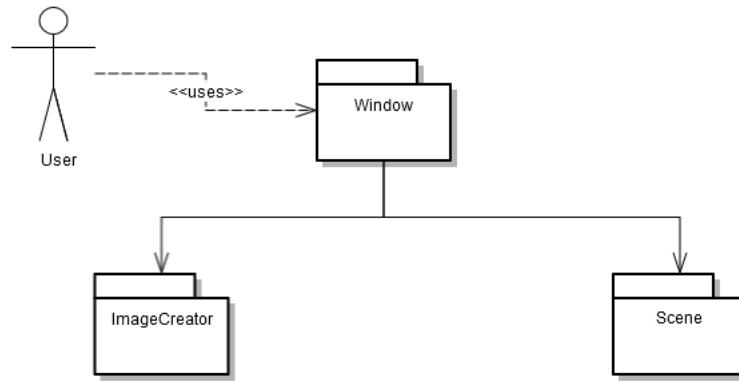


FIGURE 3.10 – Diagramme des paquetages en fin de projet¹⁴

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

Le paquetage Crédit tel que présenté dans la figure 3.10 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.

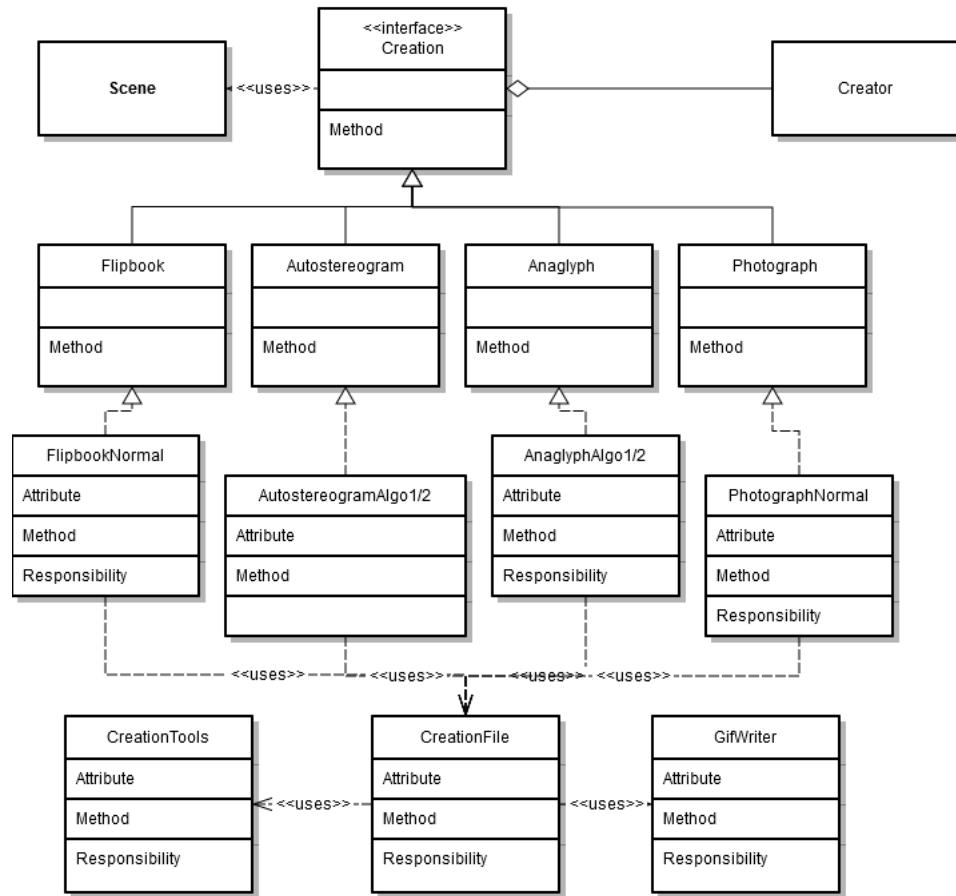


FIGURE 3.11 – Paquetage Creation en fin de projet¹⁵

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 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 3.11.

3.6 Test des fonctionnalités

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

15. Réalisé grâce au logiciel Gliffy : www.gliffy.com

16. Réalisé grâce au logiciel Gliffy : www.gliffy.com

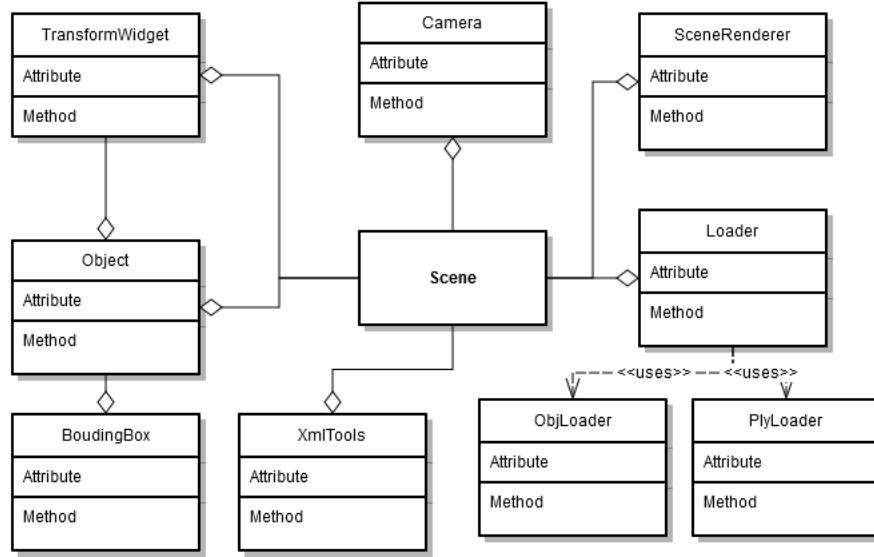


FIGURE 3.12 – Paquetage Scene en fin de projet¹⁶

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]

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

3.7 Difficultés rencontrées

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

3.7.1 Implémentation des algorithmes

L'implémentation des algorithmes d'anaglyphes et d'autostéréogrammes était le cœur 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.

Pour les autostéréogrammes, le rendu donné par le premier algorithme choisi était découpé en tranches et ne donnait pas suffisamment l'impression de profondeur comme il aurait dû, comme nous pouvons le voir sur la figure 3.12.

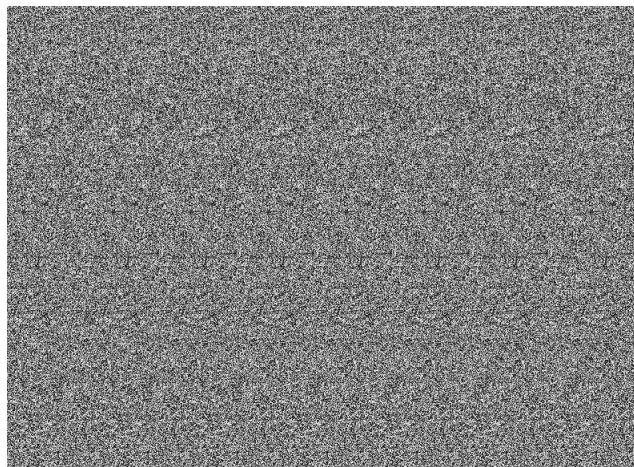


FIGURE 3.13 – Autostéréogramme d'une boule avec un effet en tranche

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 proposée 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 empêchés 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.

3.7.2 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.

3.7.3 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 QDom. 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.

3.8 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, notamment les anaglyphes et les autostéréogrammes. Il fallait revenir sur les articles trouvés et présentés dans le cahier des charges pour s'approprier les méthodes présentées afin de pouvoir les retranscrire et approfondir les recherches pour peut-être trouver d'autres algorithmes plus performant. 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 étaient 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.

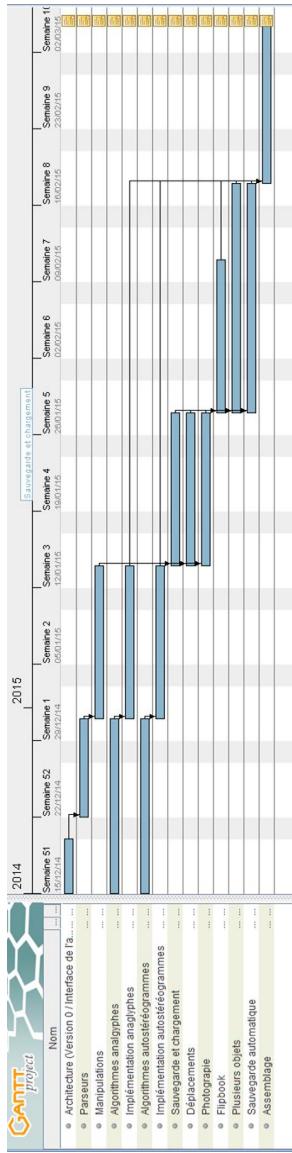


FIGURE 4.1 – Graphe temporel du déroulement du projet¹⁷

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 découplant 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

17. Réalisé grâce au logiciel GanttProject : <http://www.ganttp project.biz/>

18. Réalisé grâce au logiciel GanttProject : <http://www.ganttp project.biz/>

Untitled Gantt Project

18 mars 2015

2

Tâches

Nom	Date de début	Date de fin
Architecture (Version 0 / Interface de l'application) <i>Architecture du projet : .hpp décrivant les classes, leurs méthodes, les entrées et les sorties</i>	15/12/14	19/12/14
Parseurs <i>Parseur OBJ et PLY</i>	22/12/14	30/12/14
Manipulations <i>Changement d'un objet & Implémentation des transformations suivantes pour les objets : Rotation, translation et homothétie.</i>	31/12/14	13/01/15
Algorithmes anaglyphes <i>Aprofondissement de l'existant</i>	15/12/14	30/12/14
Implémentation anaglyphes <i>Implémentations des algorithmes des anaglyphes</i>	31/12/14	13/01/15
Algorithmes autostéréogrammes <i>Poursuite de la recherche de l'existant</i>	15/12/14	30/12/14
Implémentation autostéréogrammes <i>Implémentation et tests des algorithmes</i>	31/12/14	13/01/15
Sauvegarde et chargement <i>Sauvegarde et chargement des scènes au format XML</i>	14/01/15	27/01/15
Déplacements <i>Implémentation des transformations suivantes pour la caméra : Rotation, translation et zoom.</i>	14/01/15	27/01/15
Photographie <i>Photographie d'une scène, enregistrement au format PNG</i>	14/01/15	27/01/15
Flipbook <i>Gestion de la trajectoire et prise de photo successive de la scène.</i>	28/01/15	10/02/15
Plusieurs objets <i>Prise en charge du chargement de plusieurs objets, sélection et suppression multiple etc.</i>	28/01/15	17/02/15
Sauvegarde automatique <i>Gestion de la sauvegarde automatique en arrière plan (en parallèle)</i>	28/01/15	17/02/15
Assemblage <i>Assemblage de toutes les briques logiciels avec interface finale</i>	18/02/15	04/03/15

FIGURE 4.2 – Récapitulatif des tâches et des périodes par tâche¹⁸

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.

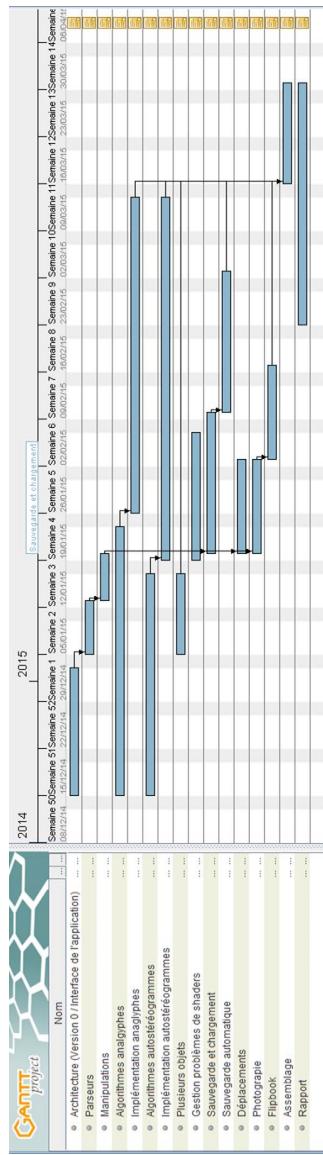


FIGURE 4.3 – Graphe temporel du déroulement du projet¹⁹

La principale modification consiste ainsi en l'allongement de la période destinée à la recherche d'algorithme 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.

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

FIGURE 4.4 – Récapitulatif des tâches et des périodes par tâche²⁰

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 aisément de connaître les tâches prioritaires 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

Annexes

A Notice d'installation du logiciel

B Notice d'utilisation du logiciel

NOTICE D'UTILISATION DU LOGICIEL PROJECT3DONUTS

Le logiciel Project3Donuts est un logiciel de synthèse d'images permettant la visualisation d'une scène en trois dimensions et la génération de photographies, d'anaglyphes, d'autostéréogrammes et de folioscopes à partir d'elle.

Menus et raccourcis clavier

===== Sauvegarde de la scène =====

Nouvelle scène : Raccourci Cntrl+N ou Icône Nouveau ou Menu déroulant Fichier > Nouveau

Ouvrir une scène pré-enregistrée : Raccourci Cntrl+O ou Icône Ouvrir ou Menu déroulant Fichier > Ouvrir Fichier de chargement acceptés : fichiers OBJ

Enregistrer la scène en cours : Raccourci Cntrl+S ou Icône Enregistrer ou Menu déroulant Fichier > Enregistrer (Pour enregistrer la scène ouverte sous un nouveau nom : Menu déroulant Fichier > Enregistrer Sous)

===== Déplacements autour de la scène =====

Rotation autour de la scène : Clic gauche continu sur la scène et déplacement de la souris

Translation de la caméra : Clic continu du bouton du milieu et déplacement de la souris

Zoom avant/arrière : Déplacement de molette de la souris

===== Gestion des objets de la scène =====

Ajouter un objet dans la scène : Menu déroulant Fichier > Importer (depuis la bibliothèque : bibliothèque de fichiers utilisables proposée par le logiciel depuis le disque dur : fichier objets de l'utilisateur) Fichier objets acceptés : fichier PLY versions 1.0 ASCII et binaire fichier OBJ version 3.0 ASCII

Sélection d'un objet : Clic droit sur un objet ou Clic gauche sur le nom de l'objet dans la colonne verticale

Modifier la position d'un objet : APRES SELECTION DE L'OBJET Raccourci clavier T ou Icône Translation PUIS Clic gauche continu sur les axes et déplacement de la souris

Modifier la rotation d'un objet : APRES SELECTION DE L'OBJET Raccourci clavier R ou Icône Rotation PUIS Clic gauche continu sur les axes et déplacement de la souris

Modifier la taille d'un objet : APRES SELECTION DE L'OBJET Raccourci clavier S ou Icône Scale PUIS Clic gauche continu sur les axes et déplacement de la souris

Annuler les modifications : Raccourci Cntrl+Z ou Icône Annuler

Modifier la couleur d'un objet : APRES SELECTION DE L'OBJET Menu déroulant Editer > Couleur de l'objet

===== Paramètres du logiciel =====

Modifier la couleur du fond de scène : Menu déroulant Editer > Préférences

Modifier la durée entre deux sauvegardes automatiques : Menu déroulant Editer > Préférences

Changement de côté de la barre verticale : Menu déroulant Fenêtres > Mettre la fenêtre de visualisation à gauche (option temporaire) ou Menu déroulant Editer > Préférences (option durable)

Modifier les raccourcis : Menu déroulant Editer > Préférences

Autres options : Menu déroulant Editer > Préférences

Quitter le logiciel : Raccourci Escap ou Icône Quitter classique du système d'exploitation utilisé ou Menu déroulant Fichier > Quitter

===== Obtention des rendus =====

Photographie : Raccourci P ou Icône Photographie ou Menu déroulant Outils > Effectuer un rendu

Anaglyphe : Raccourci N ou Icône Anaglyphe ou Menu déroulant Outils > Anaglyphes

Autostéréogramme : Raccourci U ou Icône Autostéréogramme ou Menu déroulant Outils > Autostéréogrammes

Flipbook : Raccourci F ou Icône Flipbook ou Menu déroulant Outils > Flipbook

C Conventions de codage

D Exemple de fichier de chargement

```
<?xml version="1.0" encoding="UTF-8" ?>
<scene>
    <object name="cube1" src="resources/example/cubeout.ply">
        <scale>
            <coordinates>
                <x>1</x>
                <y>1</y>
                <z>1</z>
            </coordinates>
        </scale>
        <translation>
            <coordinates>
                <x>0</x>
                <y>2</y>
                <z>0</z>
            </coordinates>
        </translation>
        <rotation>
            <coordinates>
                <x>0</x>
                <y>0</y>
                <z>0</z>
            </coordinates>
        </rotation>
    </object>
    <object name="monkey1" src="resources/example/monkey.obj">
        <scale>
            <coordinates>
                <x>1</x>
                <y>2</y>
                <z>1</z>
            </coordinates>
        </scale>
        <translation>
            <coordinates>
                <x>2</x>
                <y>0</y>
                <z>0</z>
            </coordinates>
        </translation>
        <rotation>
            <coordinates>
                <x>1</x>
                <y>0</y>
                <z>0</z>
            </coordinates>
```

```

        </rotation>
    </object>
<object name="monkey2" src="resources/example/monkey.obj">
    <scale>
        <coordinates>
            <x>1</x>
            <y>1</y>
            <z>1</z>
        </coordinates>
    </scale>
    <translation>
        <coordinates>
            <x>0</x>
            <y>0</y>
            <z>0</z>
        </coordinates>
    </translation>
    <rotation>
        <coordinates>
            <x>0</x>
            <y>0</y>
            <z>0</z>
        </coordinates>
    </rotation>
</object>
<camera angle="0.000000">
    <translation>
        <coordinates>
            <x>0</x>
            <y>0</y>
            <z>7</z>
        </coordinates>
    </translation>
    <rotation>
        <coordinates>
            <x>0</x>
            <y>0</y>
            <z>0</z>
        </coordinates>
    </rotation>
</camera>
</scene>

```

E Cahier des charges

Cahier des charges

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 Domaine	3
2 État de l'art et existant	4
2.1 Logiciels existants	4
2.2 Projection vers deux dimensions	4
2.3 Les flipbooks	5
2.4 Les autostéréogrammes	6
2.5 Les anaglyphes	7
3 Sujet	9
4 Cahier des besoins	9
4.1 Besoins fonctionnels	9
4.1.1 La scène	9
4.1.2 Les objets	10
4.1.3 La caméra	10
4.1.4 Le passage en deux dimensions	11
4.2 Besoins non fonctionnels	12
4.2.1 La fluidité d'affichage	12
4.2.2 Fluidité d'obtention des rendus	12
4.2.3 Portabilité du logiciel	12
4.2.4 Extensibilité du logiciel	12
4.3 Contraintes	13
4.3.1 Superposition des objets dans la scène	13
4.3.2 Utilisation des bibliothèques	13
4.3.3 Langage de programmation	13
4.3.4 Open Source	13
5 Maquette	13
5.1 Fenêtre principale	13
5.2 Fenêtre de chargement d'un objet	14
5.3 Interface de modification d'un objet	15
5.4 Fenêtre de choix des options du rendu	16
5.5 Fenêtre d'affichage du rendu	17
6 Prototype test	17
6.1 Format de sauvegarde d'une scène	17
6.2 Test de fluidité	18
6.3 Test de portabilité	18
6.4 Réduction du nombre de polygones d'un modèle 3D	18
7 Architecture du projet	19
7.1 Diagramme de séquences	20
7.1.1 Création d'un objet	20
7.1.2 Déplacement dans la scène	21
7.1.3 Création d'un autostéréogramme	21
7.1.4 Création d'un flipbook	22
7.2 Diagrammes de classes	23
7.2.1 Paquetage Crédit	23

7.2.2 Paquetage Scène	24
7.2.3 Paquetage Interface	25
A Modèles utilisés pour les prototypes	26
B Résultats des tests de fluidité	26
C Schéma XML	27

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 inexiste sur un point caché. En tenant compte de cette lumière, on peut obtenir une image comme présentée sur la figure 1.1.

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.

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

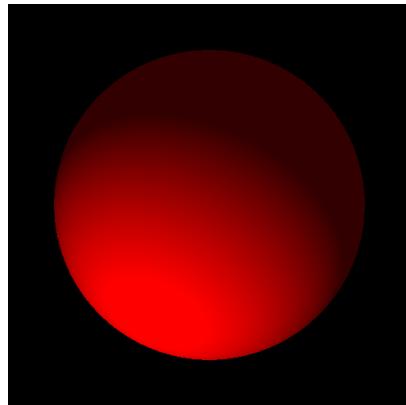


FIGURE 1.1 – Application d'une lumière diffuse à une sphère rouge¹

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 pré-déterminé 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 État de l'art et existant

2.1 Logiciels existants

StereoPhoto Maker² et Anaglyph Maker 3D³ sont des freewares permettant, entre autre, depuis deux photos d'obtenir l'anaglyphe rouge / cyan correspondant. La qualité de l'anaglyphe obtenu dépend du décalage entre les deux photos et de la qualité de la prise de vue.

Il existe également de nombreux logiciels de création d'autostéréogrammes, comme Stereographic Suite⁴ (@IndaSoftware) et 3DMiracles⁵ (@Urry Software Lab) qui permettent de créer des autostéréogrammes à partir d'une carte de profondeurs (image en deux dimensions en niveaux de gris).

2.2 Projection vers deux dimensions

Le passage d'un objet en trois dimensions à une image en deux dimensions se fait par projection. Une caméra présente dans la scène est déterminée par sa position et le vecteur direction qui indique vers quel endroit elle regarde dans la scène. La vue elle-même est obtenue par projection de la scène dans le repère de la caméra comme sur la figure 2.1.

2. StereoPhoto Maker : <http://stereo.jpn.org/eng/stphmkr/>
3. Anaglyph Maker 3D : http://www.stereoeye.jp/software/index_e.html
4. Stereographic Suite : <http://indasoftware.com/stereo/>
5. 3DMiracles : <http://www.ixtlan.ru/>

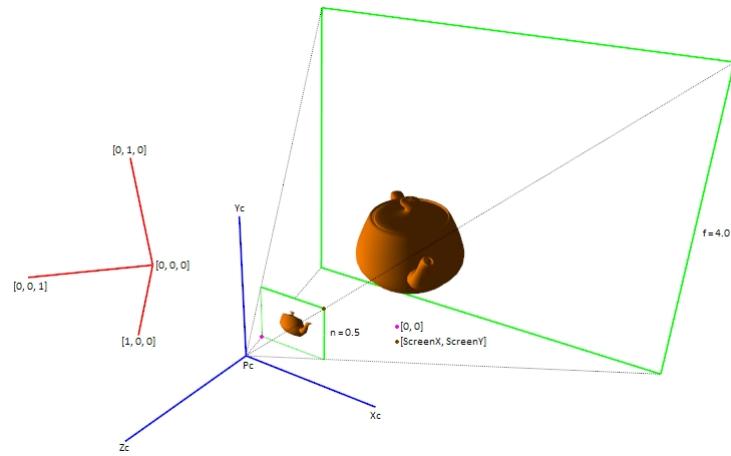


FIGURE 2.1 – Projection d'un objet en trois dimensions sur un support en deux dimensions⁶

2.3 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.

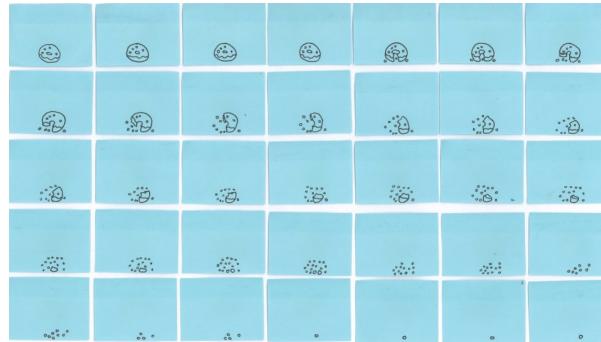


FIGURE 2.2 – Story-board d'un flipbook⁷

6. OpenGL perspective projection : <http://www.3dcpptutorials.sk/index.php?id=2>

7. <http://tracie.liu.blogspot.fr/2010/08/flipbook-storyboard.html>

2.4 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.

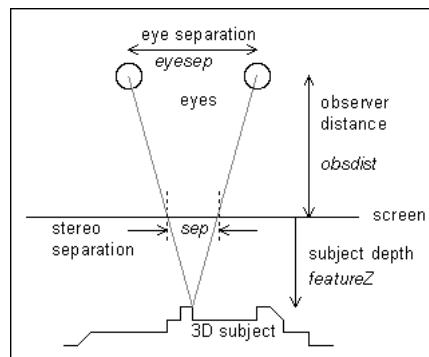


FIGURE 2.3 – Visualisation d'un autostéréogramme en vision parallèle⁸

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.

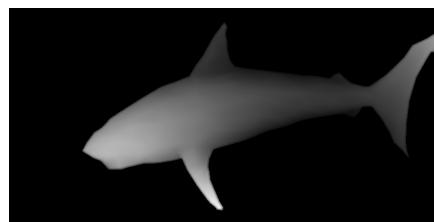


FIGURE 2.4 – Carte des profondeurs⁹

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

8. <http://www.techmind.org/stereo/geometry.gif>

9. <http://en.wikipedia.org/wiki/Autostereogram>

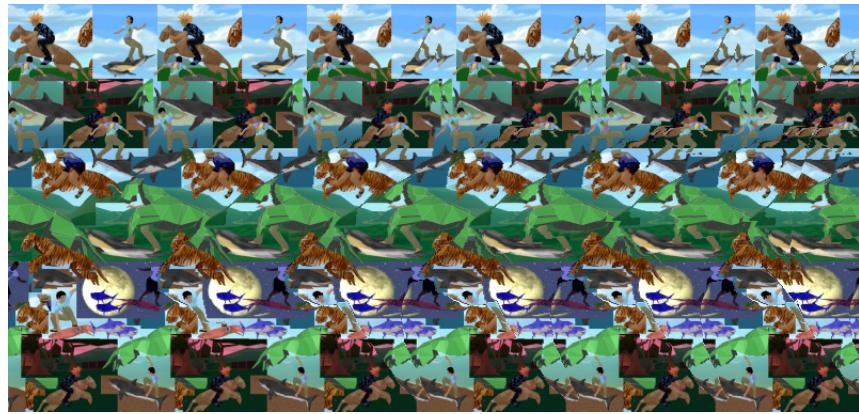


FIGURE 2.5 – Autostéréogramme obtenu¹⁰

Des algorithmes plus poussés ont été développés, comme celui présenté par Harold W. Thimbleby, Stuart Inglis et Ian H. Witten [?], et celui proposé par W. A. Steer [?], 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.

2.5 Les anaglyphes

Un anaglyphe est une image sur laquelle on superpose deux vues, si possible différentes, d'une scène. 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.

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 2.6). 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 diffèrentent par la qualité de l'anaglyphe en sortie en diminuant le nombre d'artefacts [?] ou en améliorant le rendu pour l'impression [?] par exemple.

Le paragraphe précédent concernent la création d'un anaglyphe depuis deux prises de vue d'une même scène avec un léger décalage. Mais il est également possible de le faire à partir d'une image en deux dimensions grâce à l'algorithme de Dubois [?], qui définit une vision matricielle de l'anaglyphe (cf. figure 2.7). Dès lors, une matrice de transformation

10. <http://en.wikipedia.org/wiki/Autostereogram>

12. <http://www.david-romeuf.fr/3D/Anaglyphes/TCAnglypheLSDubois/TransformationCouleursPourAnaglyphe.html>

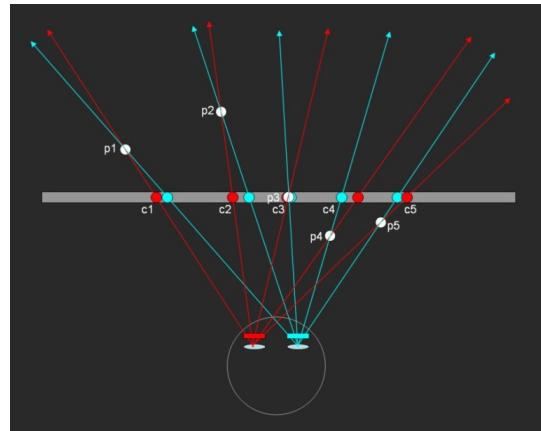


FIGURE 2.6 – Le décalage de la partie rouge et cyan permet à l’œil de percevoir l’image non plus dans un plan XY mais dans l’espace¹²

est introduite, qui permet de passer du taux RGB d’un pixel de l’image originale, aux deux taux RGB des anaglyphes gauche/droite. Les coefficients de cette matrice sont calculés pour satisfaire une bonne restitution de l’image originale, tout en supprimant les rivalités colorées induites par des lunettes bicolores. Ainsi, pour visualiser une image rouge pure, il faudra changer le taux RGB de ce pixel pour que les deux yeux puissent le voir après les filtres. Sinon l’information ne circulera que dans un œil, et l’on perdra la vision stéréoscopique, et donc la notion de profondeur.



FIGURE 2.7 – Rendu final d’une image par l’algorithme de Dubois¹³

13. <http://zour.deviantart.com/art/Wernigerode-Boulevard-Dubois-Anaglyph-HDR-3D-276542278>

3 Sujet

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'inséreront 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.

4 Cahier des besoins

4.1 Besoins fonctionnels

4.1.1 La scène

Cette partie montre l'ensemble des cas d'utilisation relatifs à la scène.

Création de la scène :

Au démarrage du logiciel, il sera possible de créer une nouvelle scène ou d'en charger une préexistante. Celle-ci permettra d'afficher et disposer plusieurs objets 3D avant d'être sauvegardée pour une utilisation ultérieure.

Sauvegarde de la scène :

On distinguera deux types de sauvegardes indépendantes entre elles : la sauvegarde automatique et la sauvegarde manuelle. La première s'effectuera après chaque modification apportée avec pour but d'éviter la perte de données en cas de crash du logiciel, elle se fera de manière discrète (c'est à dire en arrière-plan sans figer le logiciel le temps de la sauvegarde). La seconde se fera au format XML (dont le prototype est détaillé ultérieurement) sur demande de l'utilisateur. Il sera possible de choisir entre une importation des objets utilisés dans le répertoire courant de la scène ou d'une simple sauvegarde des chemins d'accès aux modèles 3D. Par ailleurs, les deux types de sauvegarde étant indépendants, une sauvegarde manuelle n'écrasera pas une sauvegarde automatique.

Chargement de la scène :

Comme vu précédemment, lors de la sauvegarde, les données relatives aux objets de la scène seront stockées dans un fichier au format XML. A l'ouverture du logiciel, l'utilisateur pourra choisir d'ouvrir une scène qu'il aura sauvegardée précédemment. Dans ce cas, les données enregistrées seront récupérées pour recréer la scène telle qu'elle était avant la fermeture du logiciel.

Suppression de la scène :

Le logiciel ne proposera pas directement de méthode pour supprimer une scène. Pour cela, il faudra directement aller supprimer le dossier associé à la sauvegarde à l'intérieur du dossier contenant le logiciel.

4.1.2 Les objets

Une fois la scène créée ou chargée, l'utilisateur aura la possibilité d'y placer des objets.

Chargement des fichiers objets :

Les objets en trois dimensions utilisés seront sous la forme de fichiers objets. Deux types seront acceptés : les fichiers d'extension .OBJ (version 3.0, ASCII) et .PLY (versions 1.0, ASCII et binaire). Ils devront permettre de définir des objets à facettes triangulaires. Les objets pourront être ceux de l'utilisateur ou de la bibliothèque implémentée dans le logiciel.

Ces objets ne devront contenir aucun trou, par exemple une face absente sur un cube ou un triangle non généré par le fichier de chargement. Le fonctionnement du logiciel n'est garanti que sur des objets topologiquement valides.

Si un fichier à charger possède trop de polygones, un algorithme de décimation de faces permet d'en réduire le nombre jusqu'à un certain seuil (cf. prototype). Ce traitement n'est pas automatique et l'utilisateur pourra ou non l'appliquer.

Placement d'un objet :

Après le chargement de l'objet il sera possible de le placer dans la scène via les trois translations du repère 3D. Ce placement n'est pas définitif et pourra être modifié ultérieurement lors de la constitution de la scène.

Modification d'un objet :

Trois caractéristiques d'un objet pourront être modifiées : son emplacement, son orientation et sa taille.

Modification de l'orientation d'un objet :

L'orientation d'un objet dans la scène pourra être modifiée en fonction des trois axes de rotation usuels.

Modification de la taille d'un objet :

L'objet pourra être agrandi ou réduit selon le besoin.

Mode de modification d'un objet :

Le placement et le paramétrage de l'objet (orientation, taille ...) n'est pas définitif et peut-être revu par la suite via un mode de modification après avoir sélectionné l'objet à modifier. Toute sortie de ce mode de modification entraînera une sauvegarde automatique de la scène.

Sélection d'un objet :

Il sera possible de sélectionner un objet pour réaliser des modifications.

Suppression d'un objet :

Il sera également possible, après sélection, de supprimer un ou plusieurs objets de la scène. Le ou les objets ainsi supprimés ne seront plus référencés dans la sauvegarde.

4.1.3 La caméra

La caméra possède un repère qui lui est propre et une distance correspondant à celle entre l'origine du repère de la scène (centre de rotation de la caméra) et le sien.

Observation de la scène :

La caméra permettra de se déplacer dans la scène en suivant les trois axes de rotation et les trois axes de translation usuels, ainsi qu'un zoom. Si l'on choisit de se déplacer relativement aux axes de translation, le centre de rotation de la caméra sera modifié.

Type de projection :

Deux types de projections seront disponibles : la projection orthographique (qui sera l'option par défaut) et la projection en perspective.

Zoom de la scène :

Il sera possible de se rapprocher ou s'éloigner de la scène via une fonctionnalité de zoom. Il ne sera pas possible d'atteindre le centre de rotation de la caméra.

Remise à zéro du centre de la caméra :

La position de la caméra pourra, à tout moment, être réinitialisée pour se retrouver dans les conditions initiales de visionnage de la scène.

4.1.4 Le passage en deux dimensions

Mode de passage en deux dimensions :

Différents rendus seront accessibles :

- Photographie résultant d'une projection sans traitement de la scène en trois dimensions.
- Anaglyphe rouge-cyan à partir de paires d'images en noir et blanc avec traitement possible pour l'impression.
- Autostéréogramme SIRDS selon l'algorithme de Witten, Inglis et Thimbleby.
- Flipbook d'une rotation complète ou non autour d'un objet ou d'une scène.

Pour les images générées par les modes présentés ci-dessous, il faudra choisir entre une qualité de 75 ou 300 pixels par pouce.

Mode photographie :

Le mode photographie permettra d'obtenir une image proche de la scène virtuelle créée. Pour ce mode, des informations concernant la qualité de l'image et sa taille devront être fournies.

Mode analgyphe :

Le mode anaglyphe permettra d'obtenir une image rouge, une image cyan, ainsi qu'une image superposant les deux. Pour ce mode, des informations concernant la qualité de l'image et sa taille devront être fournies.

Mode autostéréogramme :

Le mode autostéréogramme permettra d'obtenir un autostéréogramme fixe à une image. Pour ce mode, des informations concernant la qualité de l'image et sa taille devront être fournies.

Mode flipbook :

Le mode flipbook permettra d'obtenir un ensemble d'image sur une trajectoire donnée. Pour ce mode, des informations concernant la qualité de l'image et sa taille, mais également des informations nécessaires au lancement et à la construction du flipbook devront être fournies par l'utilisateur. Ainsi, il devra tout d'abord choisir une trajectoire parmi les trois axes de rotation, les trois axes de translation ou le zoom, puis un point de départ. Au choix, il pourra alors sélectionner un point d'arrivée et un nombre d'images, ou un nombre d'image et un angle pour connaître la mesure entre deux prises. Une barre de progression indiquera le nombre de vues faites par rapport au total. Si le temps d'attente est trop long, il sera toujours possible d'annuler l'action et redéfinir les paramètres.

Affichage du rendu :

Les algorithmes des modes présentés ci-dessus permettront d'obtenir un rendu correspondant aux attentes de l'utilisateur. Ce dernier sera ensuite affiché dans une nouvelle fenêtre contenant le ou les produits demandé qu'il sera possible de sauvegarder avant de quitter.

Enregistrement du rendu :

Si l'utilisateur choisit d'enregistrer les images générées par les algorithmes, les rendus qu'il souhaite sauvegarder devront être choisis parmi les rendus obtenus.

Enregistrement de la photographie :

Pour le cas simple de la photographie, il faudra nommer le fichier et donner son emplacement avant qu'il ne soit enregistré au format PNG.

Enregistrement de l'anaglyphe :

Les images rouge et cyan pourront être enregistrées séparément ou non. Deux options de rendus seront disponibles : pour l'impression et pour l'ordinateur. Elles permettront d'optimiser l'affichage et l'utilisation des lunettes dans les deux cas.

Enregistrement de l'autostéréogramme :

Pour l' autostéréogramme, il faudra nommer le fichier et indiquer son emplacement avant de le sauvegarder au format PNG.

Enregistrement du flipbook :

Dans le cas du flipbook, les images pourront être enregistrées une à une au format PNG, par tranche au format PNG ou sous forme d'animation GIF. Dans le cas d'une sauvegarde séparée de chaque image il faudra les nommer et donner leur chemin d'accès. Par tranche il faudra indiquer la première et la dernière image ainsi qu'un nom commun et un répertoire. Pour l'animation il faudra la nommer et donner son chemin d'accès.

4.2 Besoins non fonctionnels

4.2.1 La fluidité d'affichage

Pour que l'affichage soit fluide la machine devra avoir au minimum 2Go de mémoire vive, un processeur récent avec au moins 2 cœurs cadencés au minimum à 2GHz et les conditions suivantes devront être validées :

- Pour une machine avec processeur graphique type Intel HD Graphics 4000 ou plus récent, au maximum 100 000 points pour un affichage à 25 fps, ou 150 000 points pour un affichage à 20 fps ;
- Pour une machine avec une carte graphique type NVIDIA GTX 720 ou plus récente, au maximum 350 000 points pour un affichage à 25 fps, ou 500 000 points pour un affichage à 20 fps.

Test de validation : ouverture de plusieurs modèles 3D, présentés en annexe, dans Meshlab sur différentes machines. Les résultats sont également disponibles en annexe.

4.2.2 Fluidité d'obtention des rendus

A l'heure actuelle on ne peut garantir un nombre de points assurant l'obtention d'un rendu en un temps raisonnable. C'est pourquoi une barre de progression s'affichera pour que l'utilisateur ait une idée du temps d'attente avant l'affichage du résultat.

4.2.3 Portabilité du logiciel

La portabilité rend le logiciel accessible à un plus grand nombre de personnes. Les systèmes d'exploitation visés seront Windows XP, Windows Seven, Windows 8.1, Ubuntu 14.04, Fedora 20 et Debian Jessie, en plateforme 32 bits et 64 bits.

Test de validation : un prototype, qui pourra être utilisé comme base du projet, a été exécuté sur l'ensemble de ces systèmes d'exploitation (cf. la partie 6 Prototype test).

4.2.4 Extensibilité du logiciel

Le client souhaite que l'extensibilité du logiciel soit facilitée par son architecture. En effet, il envisage par la suite d'ajouter de nouvelles fonctionnalités ou de nouveaux algorithmes au logiciel, et demande à ce que de telles modifications soient aisément envisageables.

La facilitation de cette extensibilité sera permise par l'utilisation d'une architecture spécifique. Celle-ci est détaillée dans la partie 7 Architecture du projet de ce cahier des charges.

4.3 Contraintes

4.3.1 Superposition des objets dans la scène

Pour faciliter le traitement d'une scène, on considérera que deux objets placés dans une scène ne pourront pas être superposés. On parle ici d'intersection des boîtes englobant des objets et non de la gestion des parties non visibles des objets lors de l'affichage.

4.3.2 Utilisation des bibliothèques

Afin de minimiser les dépendances et valider le besoin en portabilité les bibliothèques Qt et OpenGL seront utilisées. CMake permettra de générer les scripts de compilation quel que soit la plateforme.

4.3.3 Langage de programmation

Le langage C++ sera utilisé.

4.3.4 Open Source

Pour permettre l'utilisation du logiciel et son extensibilité à quiconque souhaiterait le modifier, on choisira de laisser le code du projet en Open Source. Pour garantir cela, on s'assura bien que l'on dispose des droits nécessaires pour n'importe quel algorithme ou fragment de code utilisé au cours de ce projet.

5 Maquette

Cette partie montrera l'ensemble des maquettes de la future interface du logiciel.

5.1 Fenêtre principale

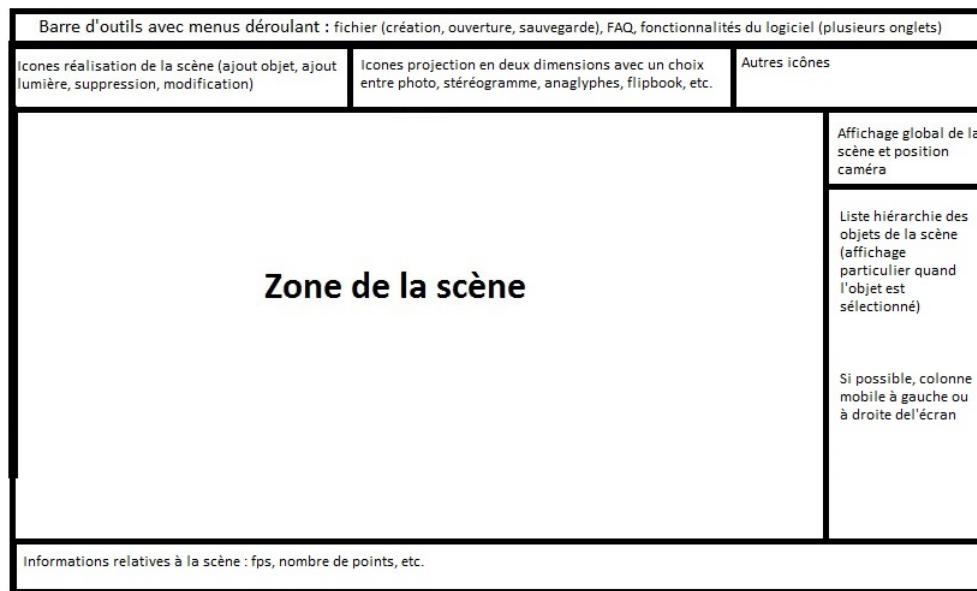


FIGURE 5.1 – Fenêtre principale

La fenêtre principale 5.1 se détaille comme suit :

- Barre d'outils avec menus déroulants

Ce bandeau supérieur correspond à un bandeau supérieur de logiciel classique, comme on le trouve par exemple dans le logiciel Meshlab.

L'intérêt de ces menus déroulants est de proposer l'accès à l'ensemble des fonctionnalités du logiciel. Parmi ces menus se trouvent quelques redondances avec des icônes présentes dans d'autres parties de la fenêtre principale, afin que l'utilisateur puisse choisir l'utilisation du logiciel qui lui convient le mieux.

- Icônes de réalisation de la scène

Les icônes permettant de réaliser la gestion des objets et des lumières de la scène : il sera possible d'ajouter un objet ou de le modifier à partir de ces icônes.

- Icônes de projection en deux dimensions

L'ensemble des icônes permettant d'obtenir l'un des rendus possibles à l'aide du logiciel, c'est-à-dire quatre icônes pour : une projection en deux dimensions classique, la création d'un anaglyphe, la création d'un stéréogramme et la création d'un flipbook.

L'ensemble de ces icônes permettra d'ouvrir une nouvelle fenêtre et de choisir les paramètres de l'image à obtenir.

- Autres icônes

Cette zone pourra contenir d'autres icônes, qui seront triées et rassemblées en fonction de leurs cas d'utilisation.

- Zone de la scène

Vu de la scène du point de vue de la caméra. L'utilisateur pour s'y déplacer librement via la caméra pour pouvoir l'observer depuis différents angles. Il pourra également sélectionner des objets pour les déplacer, modifier leur orientation, les redimensionner ou les supprimer.

- Liste des objets de la scène

L'ensemble des objets présents dans la scène listés : l'utilisateur pourra cliquer sur le nom d'un objet afin de le sélectionner et de pouvoir agir dessus.

- Informations relatives à la scène

Des informations intéressantes pour l'utilisateur par rapport à la scène et à son affichage seront affichées : le nombre d'objets dans la scène, le nombre de points, ou encore la qualité d'affichage en frames par seconde. Cette zone pourra également servir à l'affichage de messages informatifs ou de messages d'erreurs à l'intention de l'utilisateur.

5.2 Fenêtre de chargement d'un objet

L'image 5.2 schématise la fenêtre de chargement d'un nouvel objet. L'utilisateur du logiciel doit d'ores et déjà avoir ouvert une scène. Il pourra ensuite choisir d'y insérer un nouvel objet grâce à la fenêtre ci-dessus, en choisissant un modèle à ouvrir ainsi que le nom de l'objet dans la scène.

Pour le choix du modèle 3D, l'utilisateur pourra utiliser soit la bibliothèque du logiciel soit ses propres fichiers. Toutefois, seules les extensions OBJ et PLY seront acceptées.

Une fois toutes les informations entrées, l'utilisateur pourra cliquer sur le bouton Valider pour passer en mode Modification de l'objet et placer son objet dans sa scène.

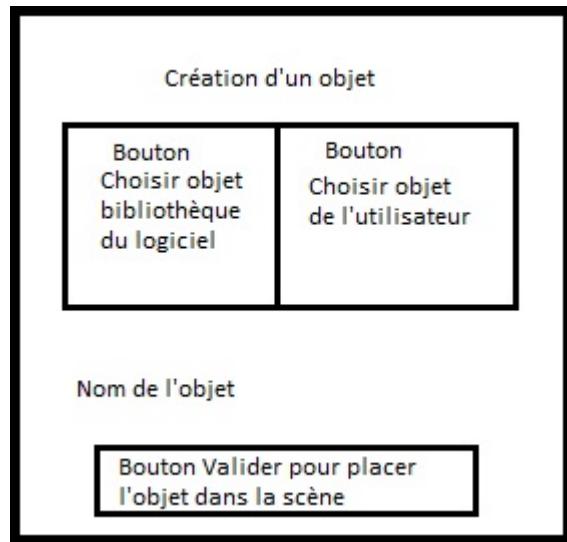


FIGURE 5.2 – Fenêtre de chargement d'un objet

5.3 Interface de modification d'un objet

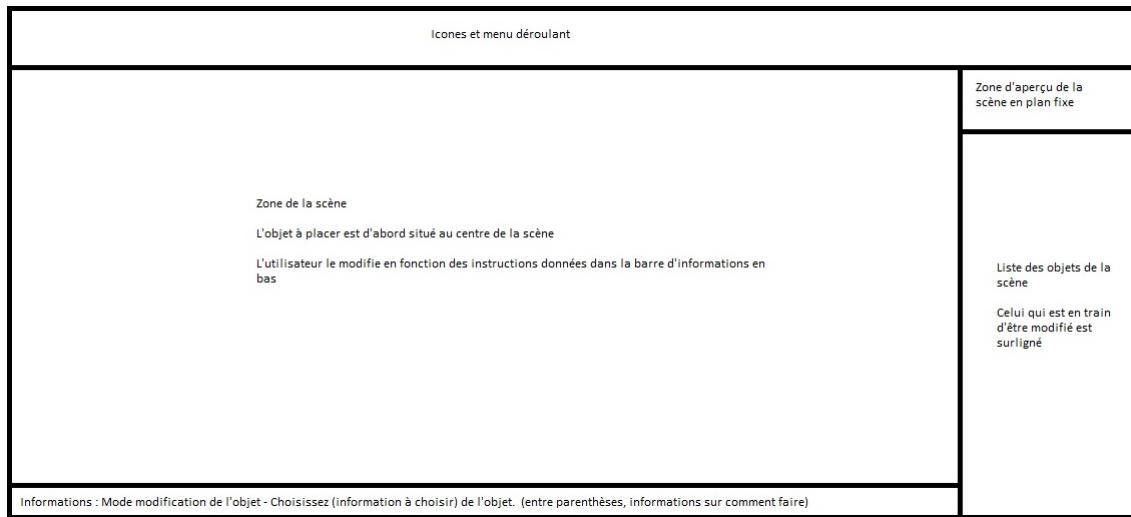


FIGURE 5.3 – Interface de modification d'un objet

Une fois qu'il aura choisi le modèle à placer, l'utilisateur verra de nouveau la fenêtre principale s'afficher, et devra suivre les informations indiquées dans la barre Informations en bas de la fenêtre pour modifier la position, l'orientation et la taille de l'objet.

S'il souhaite de nouveau effectuer des modifications sur son objet, il devra repasser en mode Modification de l'objet, et retrouvera la même interface 5.3. Il devra à chaque fois obligatoirement passer par les trois modes de modification :

position, orientation, taille.

5.4 Fenêtre de choix des options du rendu

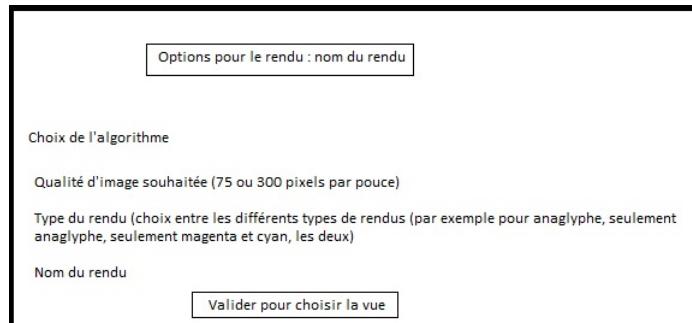


FIGURE 5.4 – Choix des options du rendu

La fenêtre 5.4 s'ouvre lorsque l'utilisateur clique sur le bouton de création d'une projection, d'un anaglyphe, d'un autostéréogramme ou d'un flipbook. Il devra alors choisir l'algorithme à utiliser pour le rendu souhaité (un ou plusieurs algorithmes pourront être proposés), la qualité d'image qu'il souhaite obtenir, ainsi que les types de retour qu'il souhaite obtenir si plusieurs choix existent. Par exemple, dans le cas d'un anaglyphe, le retour pourra contenir ou bien uniquement l'anaglyphe, ou bien les deux vues rouge et cyan, ou bien les deux.

L'utilisateur choisira ensuite le nom du rendu et appuiera sur le bouton "Valider" pour passer en mode de sélection de la vue. Ce mode reviendra sur la fenêtre principale du logiciel, et des informations seront données à l'utilisateur pour sélectionner la vue qu'il souhaite.

Toutes les informations seront obligatoirement complétées pour pouvoir accéder au mode de sélection de la vue.

5.5 Fenêtre d'affichage du rendu

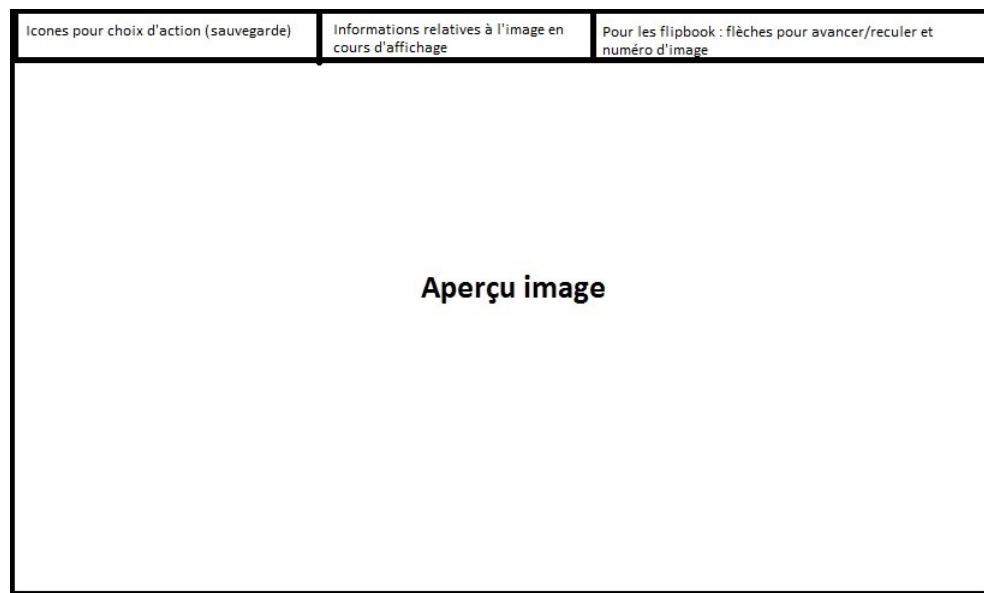


FIGURE 5.5 – Affichage du rendu

Une fois l'action paramétrée, le rendu s'affiche dans la fenêtre 5.5 dont les options sont détaillés ci-dessous.

- Zone de choix d'action

Cette zone contiendra des icônes pour indiquer à l'utilisateur les choix qui s'offrent à lui. On trouvera par exemple une icône pour l'enregistrement du rendu, qui ouvrira une nouvelle fenêtre pour laisser le choix à l'utilisateur de ce qu'il souhaite enregistrer.

- Zone d'affichage des informations

Cette zone concernera uniquement l'image en cours d'affichage dans la zone d'aperçu. On détaillera ici la taille de l'image et sa qualité en pixels par pouce.

- Zone de défilement

Cette zone contiendra des flèches de déroulement vers la gauche ou vers la droite pour permettre à l'utilisateur de voir l'ensemble des images obtenues, ainsi qu'un ou plusieurs boutons pour pouvoir afficher un GIF ou le stopper.

- Zone d'aperçu de l'image

S'affichera ici l'image obtenue après la requête d'un utilisateur.

6 Prototype test

6.1 Format de sauvegarde d'une scène

La sauvegarde d'une scène se fera au format XML. La racine scène contiendra une suite d'objets et une caméra. Un objet aura pour attribut son format et son répertoire d'accès et sera caractérisé par une translation, une rotation et un redimensionnement. La caméra sera caractérisée par une translation et une rotation. Le fichier sera validé par un schéma XML tout au long de la phase de développement et lu via des requêtes XPath.

Le schéma XML ainsi qu'un fichier XML de test ont été réalisés et sont disponibles en annexe.

6.2 Test de fluidité

Le logiciel Meshlab a été utilisé sur différentes machines avec différents modèles 3D pour déterminer un rapport FPS / nombre de points convenable. Les modèles 3D utilisés contiennent entre 40000 et 14 millions de points.

6.3 Test de portabilité

Pour s'assurer de la portabilité de QT et OpenGL sur Windows et Linux, un affichage simple d'un modèle 3D dans une fenêtre QT, sans colorations de face, a été fait et testé sur différentes distributions (fig. 6.1). Les versions 5 de QT et 3.3 d'OpenGL ont été utilisées pour ce prototype et seront utilisées dans la suite du projet. Il a été testé avec des modèles de Stanford comme le dragon, le lapin ou le bouddha.

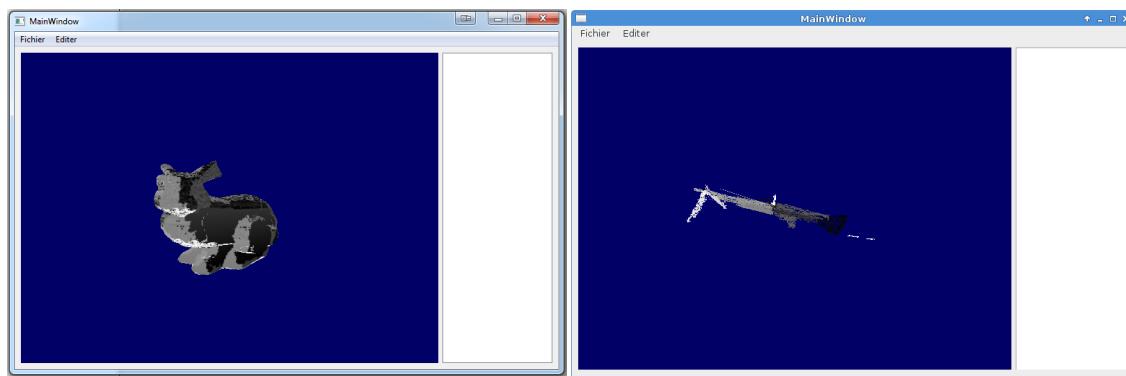


FIGURE 6.1 – Affichage du modèle 3D de lapin sur Windows 7 et M60 sous Linux - Fedora 20

6.4 Réduction du nombre de polygones d'un modèle 3D

Un algorithme permet de réduire le nombre de faces d'un modèle 3D qui en possèderait trop et serait donc impossible à afficher. Il supprime des triangles et fusionne les sommets autour, jusqu'à ce que le nombre de polygones tombe en dessous d'un certain seuil (fig. 6.2).

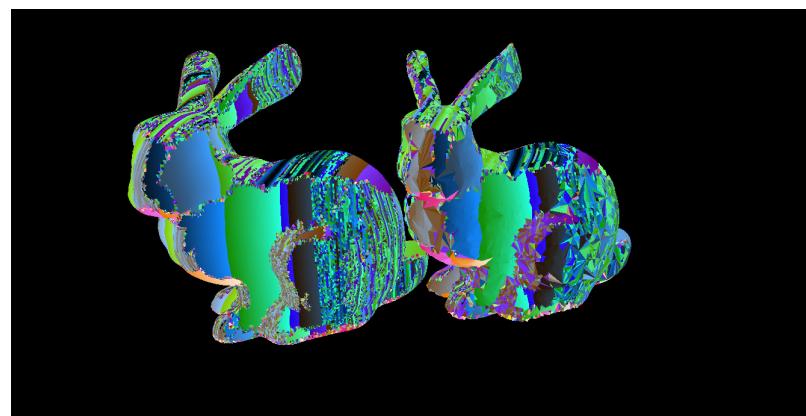


FIGURE 6.2 – À gauche modèle 3D d'origine, à droite modèle après suppression de faces

Ce traitement est effectué lors du chargement et est relativement rapide pour les objets (fig. 6.3).

```
C:\WINDOWS\system32\cmd.exe
Loading OBJ file ./data/bunny0.obj...
Begin polygon reduction
number of triangles : 69451
duration : 6.58854 s
End polygon reduction
number of triangles : 29999
```

FIGURE 6.3 – Un test effectué avec l'algorithme sur le modèle de la figure 6.2 montre qu'il met moins de 7 secondes pour supprimer 40000 faces

7 Architecture du projet

Dans cette partie, nous présenterons un ensemble de diagrammes de séquences et de diagrammes de classes présentant le fonctionnement prévisionnel du logiciel et l'architecture qui en découle. Cette architecture tiendra compte du besoin d'extensibilité du logiciel.

Le diagramme de paquetage présenté 7.1 représente l'architecture globale du futur logiciel. Il montre que l'interface graphique sera le point d'entrée pour l'utilisateur. En effet, c'est celle-ci qui redirigera les actions demandées vers soit le créateur d'image, soit le chargeur. Ce dernier sera toujours utilisé pour accéder à son attribut Scène, car il sera responsable de la création de celle-ci et de la transformation des fichiers objets en modèles 3D.

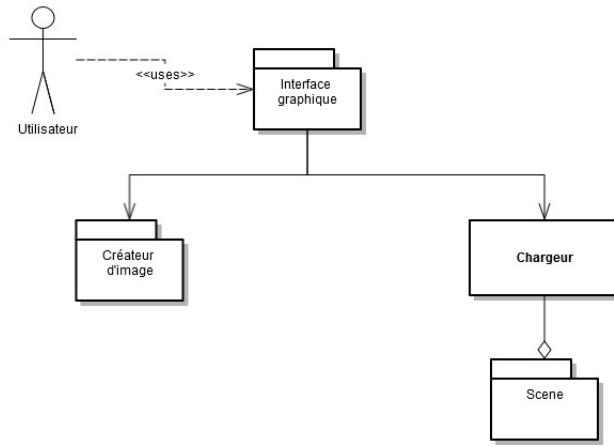


FIGURE 7.1 – Diagramme de paquetages

L'ensemble des diagrammes de séquence et des diagrammes de classe présentés par la suite auront été créés grâce à la version d'essai du logiciel en ligne Gliffy¹⁴.

7.1 Diagramme de séquences

Les diagrammes de séquence permettent de simuler les différentes communications qui auront lieu entre les classes du logiciel.

On considérera que lorsqu'il commence à utiliser le logiciel, la première action d'un utilisateur va être de créer une scène. Grâce à l'interface graphique correspondante, qui a été présentée dans la partie Interface de ce cahier des Charges, l'utilisateur demande à l'interface de créer une scène. Celle-ci passera par l'intermédiaire d'un chargeur, déjà initialisé, qui devra créer l'instance correspondante soit à partir de rien, soit à partir d'une scène déjà existante. Une fois sa tâche effectuée, le chargeur valide à l'interface qu'il a bien répondu à sa demande, et celle-ci peut modifier l'interface pour afficher la fenêtre principale, qui contient désormais la scène de travail.

7.1.1 Crédation d'un objet

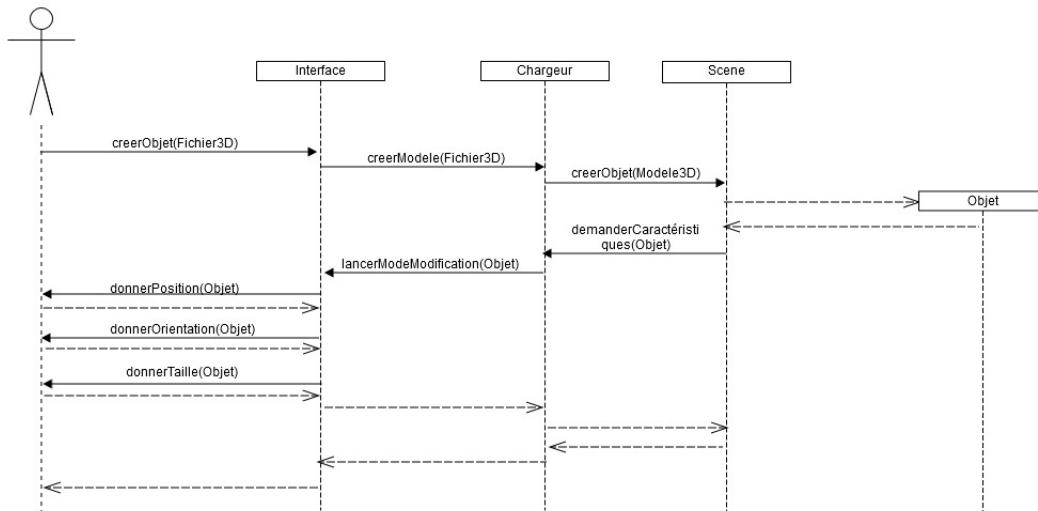


FIGURE 7.2 – Crédation d'un objet

Une fois la scène de travail créée, l'utilisateur pourra y ajouter des objets s'il le souhaite (cf. 7.2). Pour cela, il va une fois de plus devoir communiquer avec l'interface graphique correspondante.

Une fois qu'il aura choisi un fichier objet d'extension .OBJ ou .PLY, l'interface enverra celui-ci au chargeur qui vérifiera sa validité et le transformera en un modèle 3D. Ce dernier sera transmis à la scène, qui se chargera de créer l'objet en plaçant les points du modèle par rapport à son origine. Ce premier objet généré sera initialisé en position centrale de la scène, avec une taille et une orientation de base définie par ses points. Enfin, une confirmation sera envoyée au chargeur, puis à l'interface, qui affichera la fenêtre correspondant au mode modification de l'objet.

14. <http://www.gliffy.com/>

Durant ce mode, l'interface demandera à l'utilisateur de modifier la position, l'orientation et la taille de l'objet s'il le souhaite, et appellera des méthodes soient publiques de l'objet, soit en passant par le chargeur puis la scène, pour effectuer les changements demandés.

Ce diagramme de séquence montre l'importance du chargeur dans l'interaction de l'interface avec la scène. C'est lui qui s'occupera de créer la scène et de lui transmettre les demandes de l'utilisateur.

7.1.2 Déplacement dans la scène

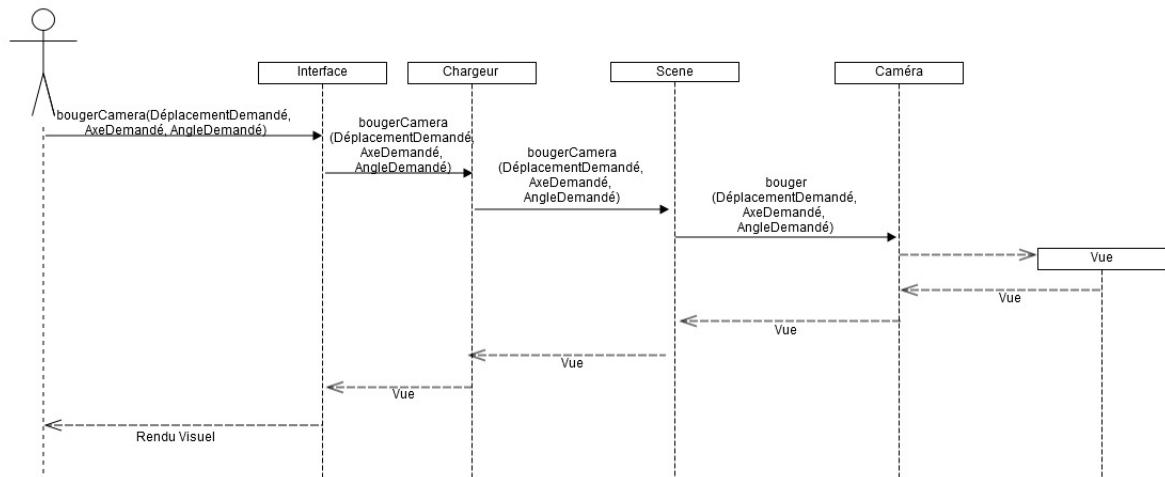


FIGURE 7.3 – Déplacement de la caméra

Quand il aura placé des objets dans sa scène, l'utilisateur pourra avoir envie de s'y déplacer pour observer les différents angles de vue. Pour cela, grâce à une action sur l'interface, il va demander à la caméra de se déplacer (cf. 7.3). La première demande sera donc initialisée par l'utilisateur, qui demandera le déplacement de la caméra à l'interface, selon un type (rotation, translation, homothétie), un axe (horizontal, vertical) et un angle ou une valeur donnée.

La caméra étant un attribut de la scène, l'interface se chargera de transmettre l'information au chargeur, qui lui-même l'enverra à la scène. Cette dernière se chargera d'appeler la fonction adéquate de son attribut qui générera une nouvelle vue correspondant à la projection de la scène depuis l'angle demandé par l'utilisateur. Cette vue sera renvoyée étape par étape jusqu'à l'interface, qui se chargera de la transformer en un rendu visualisable par l'utilisateur dans la zone de la scène.

7.1.3 Crédit d'un autostéréogramme

De la même façon que pour l'anaglyphe, c'est par l'intermédiaire de l'interface et du créateur d'image que l'utilisateur va créer son rendu (cf. 7.4). Pour un autostéréogramme toutefois, un autre type de rendu sera également utilisé : la carte des profondeurs. C'est l'autostéréogramme qui demandera au créateur d'image de la créer pour lui, afin qu'il puisse l'utiliser pour générer l'autostéréogramme qu'il renverra vers l'utilisateur.

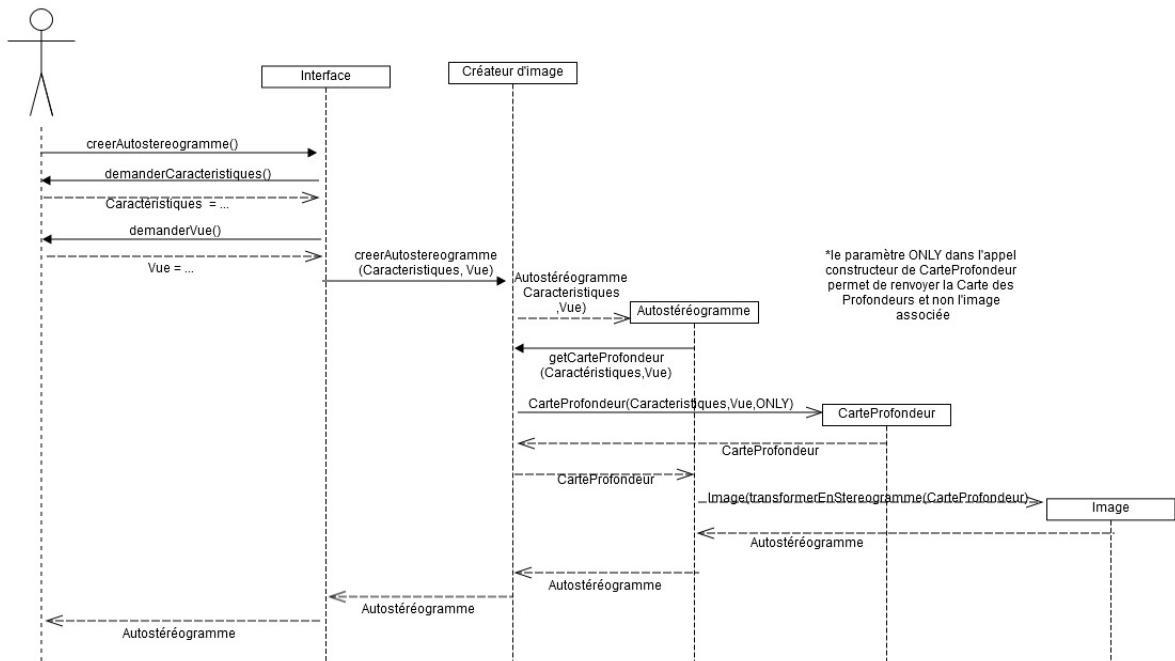


FIGURE 7.4 – Cration d'un autostrogramme

7.1.4 Cration d'un flipbook

Une fois de plus, le passage entre l'utilisateur et le flipbook se fait par l'intermédiaire de l'interface et du créateur d'image (cf. 7.5). Cette fois-ci toutefois, la génération se fera à l'aide d'une boucle qui permettra de remplir un vecteur d'image, qui sera renvoyé jusqu'à l'interface. Celle-ci se chargera de pouvoir afficher toutes les vues les unes après les autres dans la fenêtre d'affichage.

Les arguments `deplacementDemand`, `axeDemand` et `angleVoulu` qui sont utilisés en paramètre de la fonction `prendreVue` sont issus des caractristiques choisies par l'utilisateur. Cette fonction change temporairement la position de la caméra avant de la replacer à sa position initiale une fois la prise de vue faite.

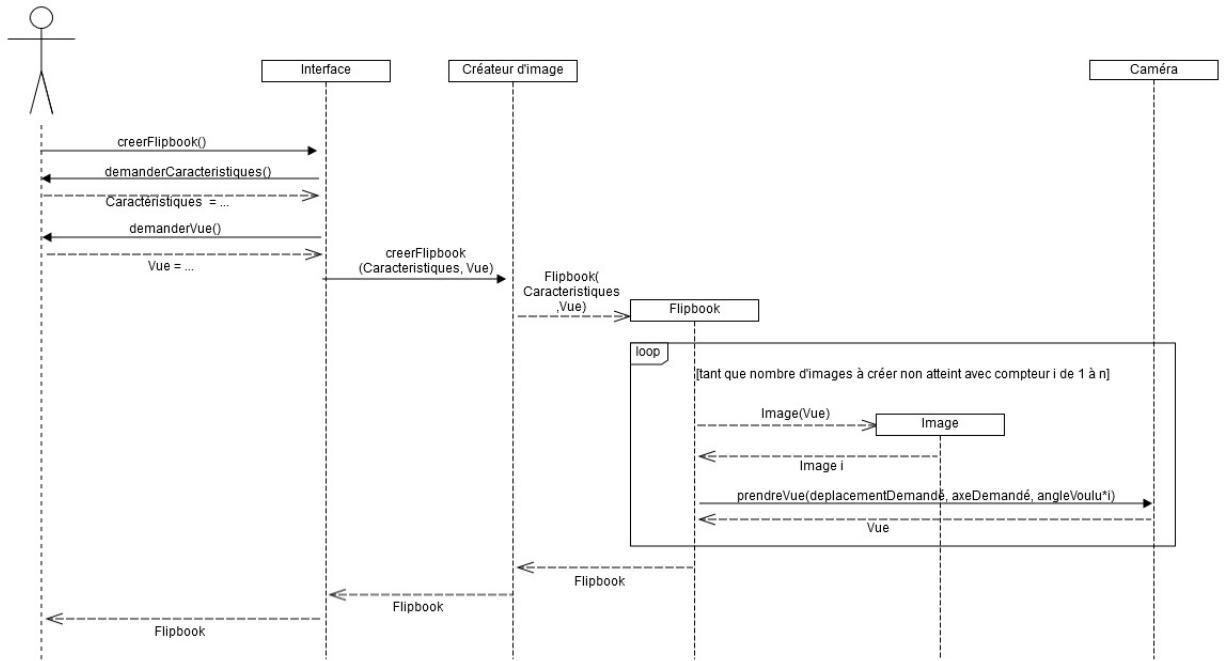


FIGURE 7.5 – Cration d'un flipbook

7.2 Diagrammes de classes

A partir des diagrammes de squences prsents prcdemment, l'architecture du projet a pu tre dtermine. La version finale de celle-ci dpendra grandement des bibliothques OpenGL et Qt et des diffrentes utilisations de leurs modules.

7.2.1 Paquetage Cration

L'architecture du paquetage de cration a t pense pour que de futures extensions puissent tre insres  ce niveau du logiciel (cf. 7.6). En effet, le crateur d'image gnre un rendu, qui peut tre l'un des types situs en dessous. Si l'on souhaite ajouter un nouveau rendu possible, il suffit donc de le placer au mme niveau que les interfaces Flipbook, Autostrogramme, etc. De la mme faon, si l'on souhaite ajouter un algorithme diffrent pour obtenir l'un des rendus, il suffit de l'ajouter en-dessous de la classe virtuelle correspondante. Ainsi cette ralisation rpond au besoin d'extensibilit.

Toutes les classes sont en relation avec la camra par l'intermdiaire de la scne, de faon  pouvoir rcuprer les diffrentes vues ncessaires  la cration des rendus.

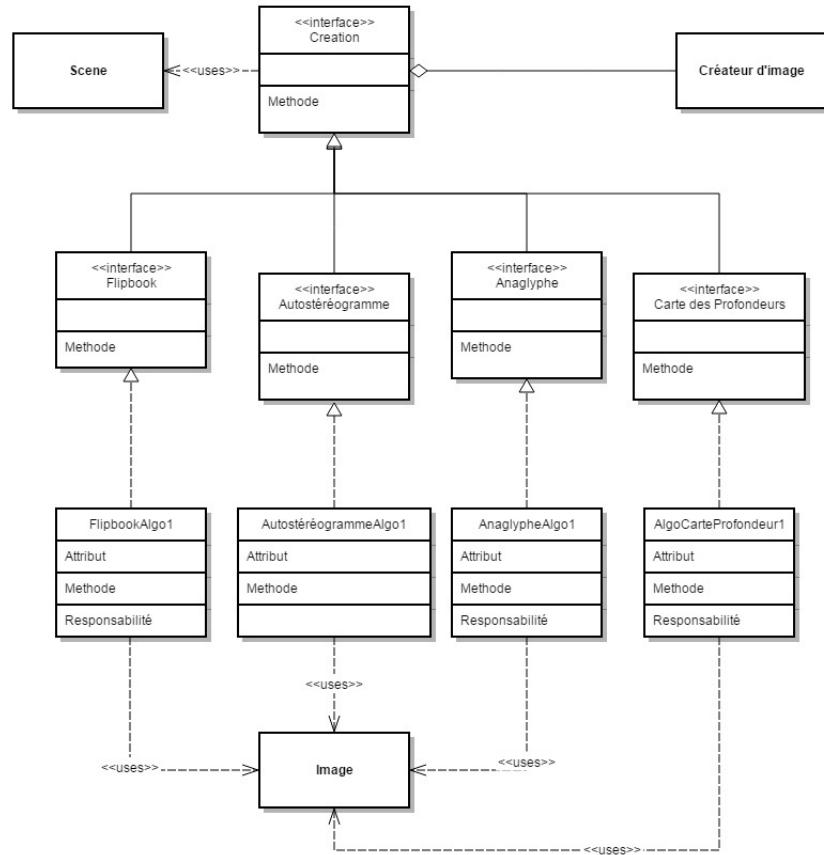


FIGURE 7.6 – Architecture du paquetage Cr eation

7.2.2 Paquetage Sc ene

On consid era pour le logiciel la sc ene comme un ensemble d'objets et une cam era. C'est la sc ene qui g era elle-m me l'acc s   ses diff rentes composantes (cf. 7.7). La cam era utilis e, qui sera li e   la cam era de la biblioth que OpenGL, g en rera des vues qui pourront  tre utilis es pour l'affichage de la sc ene ou la cr ation de rendus. Le contenu r el de ces vues et leur utilisation d pendront de la biblioth que OpenGL.

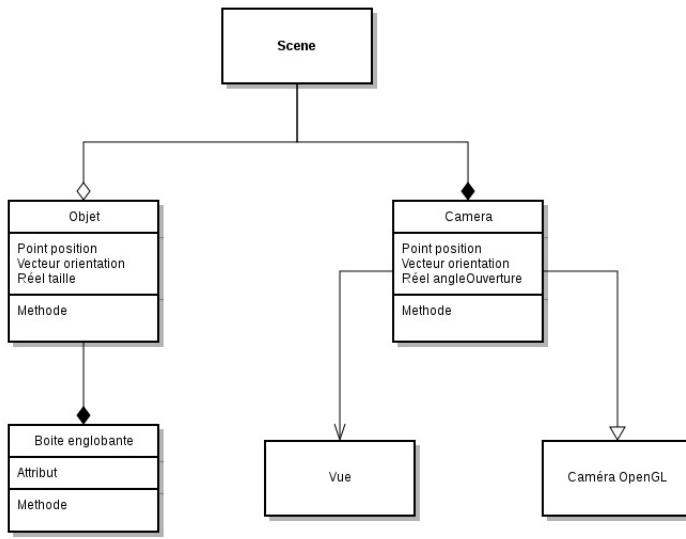


FIGURE 7.7 – Architecture du paquetage Scène

7.2.3 Paquetage Interface

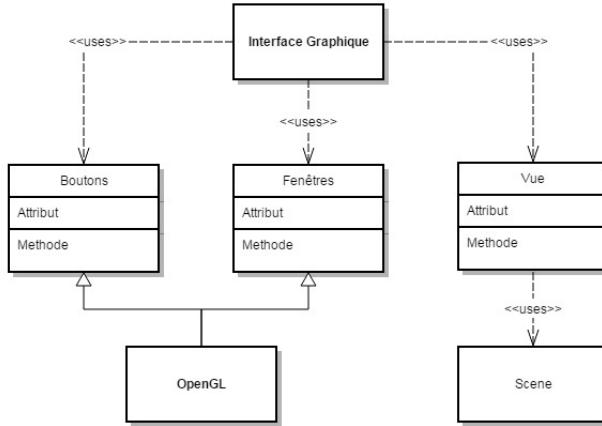


FIGURE 7.8 – Architecture du paquetage Interface

Le paquetage Interface est le paquetage utilisé par l'utilisateur pour interagir avec le logiciel. L'interface sera constituée de boutons et de fenêtres, qui seront des produits de la bibliothèque Qt et du module OpenGL pour Qt (cf. 7.8). L'affichage de la scène se fera à partir des vues générées par la Caméra de la Scène, comme expliqué dans la partie précédente.

La création de l'interface sera pensée pour que l'ajout de boutons et de fonctionnalités au logiciel se fasse de façon simple.

Annexes

A Modèles utilisés pour les prototypes

Le tableau A.1 présente les modèles utilisés en annexe B.

Nom du modèle	Faces	Sommets	Lien de téléchargement
bunny000.ply	79 312	40 256	http://graphics.stanford.edu/pub/3Dscanrep/bunny.tar.gz
cart_obj.obj	162 552	87 909	http://www.oyonale.com/downloads/cart_obj.zip
pan_obj.obj	205 697	103 032	http://www.oyonale.com/downloads/casserole_obj.zip
extincteur_obj.obj	300 632	151 425	http://www.oyonale.com/downloads/extincteur_obj.zip
M60_obj.obj	344 140	174 476	http://www.oyonale.com/downloads/M60_obj.zip
dragon_vrip.ply	871 414	437 645	http://graphics.stanford.edu/pub/3Dscanrep/dragon/dragon_recon.tar.gz
happy.ply	1 087 716	543 652	http://www.cc.gatech.edu/data_files/large_models/happy.ply.gz
blade.ply	1 765 388	882 954	http://www.cc.gatech.edu/data_files/large_models/blade.ply.gz
lucy.ply	28 055 742	14 027 872	http://graphics.stanford.edu/data/3Dscanrep/lucy.tar.gz

FIGURE A.1 – Les modèles utilisés

B Résultats des tests de fluidité

Les résultats de fluidité obtenus avec les modèles de la section A sont présentés dans les tableaux B.1 et B.2. Parmi les machines ayant servi pour les tests, deux sont relativement anciennes et les résultats s'en ressentent (voir tableau B.1, première et troisième machines).

Version de Meshlab Système Carte graphique	V1.3.4 Windows 7 Intel HD Graphics 4000	V1.3.2 Debian Jessie Intel HD Graphics 4000	V1.3.3 Debian Jessie/Sid Intel Mobile Series 4
bunny.ply	54 à 64	-	-
cart.obj	15 à 35	23 à 36	17 à 18
pan_obj.obj	20 à 30	23 à 32	15 à 18
extincteur_obj.obj	7 à 9	19 à 31	11 à 12
M60.obj	6 à 8	14 à 16	9,7 à 11,5
dragon_vrip.ply	6 à 10	12 à 13	-
happy.ply	4 à 8	8 à 10	-
blade.ply	3 à 5	0,5 à 0,6	-
lucy.ply	< 0,5	< 0,5	0,3

FIGURE B.1 – Nombre de FPS obtenus pour chaque modèle (tests réalisés sur des machines avec processeur graphique)

Version de Meshlab	V1.3.4 Windows 7 NVIDIA GTX 770	V1.3.3 Windows 7 NVIDIA GTX 760	V1.3.3 Windows XP ATI HD 4850	V1.3.2 Fedora 20 NVIDIA GTX 520	V1.3.3 Windows 8 NVIDIA GTX 330M
bunny.ply	500	-	-	-	-
cart.obj	200	-	-	-	-
pan_obj	150	-	-	-	-
extincteur_obj	100	-	-	-	-
M60.obj	95	-	-	-	-
dragon_vrip.ply	50 à 60	45 à 55	8 à 9	15 à 18	23
happy.ply	32 à 39	35 à 37	6 à 7	13 à 14	18
blade.ply	16 à 18	18	4 à 5	8 à 9	12,5
lucy.ply	0 à 2	0 à 3	Crash	0,5	Crash

FIGURE B.2 – Nombre de FPS obtenus pour chaque modèle (tests réalisés sur des machines avec carte graphique)

C Schéma XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="coordinates">
    <xsd:sequence>
      <xsd:element name="x" type="xsd:float" />
      <xsd:element name="y" type="xsd:float" />
      <xsd:element name="z" type="xsd:float" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="translation">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="vector" type="coordinates" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="scale">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="factor" type="coordinates" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="rotation">
    <xsd:complexType>

```

```

<xsd:sequence>
    <xsd:element name="angle" type="coordinates" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="object" maxOccurs="unbounded">
    <xsd:complexType>
        <xsd:attribute name="type" type="xsd:string" />
        <xsd:attribute name="src" type="xsd:anyURI" use="required" />
        <xsd:element ref="scale" />
        <xsd:element ref="translation" />
        <xsd:element ref="rotation" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="camera" maxOccurs="1">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="translation" />
            <xsd:element ref="rotation" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="scene" use="required">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="object" />
            <xsd:element ref="camera" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

</xsd:schema>

```