ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE NANTES

PROJET DE RECHERCHE

Etude des shaders en OpenGL : application à la visualisation dans un dôme

Étudiant :
Olivier CATRY
Polytech' Nantes

Encadrant :
Fabien PICAROUGNE
Laboratoire LINA

janvier 2012

Preface

Résumé

problématique : Comment réaliser l'affichage d'une scène en 3D sur un support de géométrie quelconque sans souffrir de déformations visuelles ?
La problématique est étudiée sur le sujet suivant : l'affichage sur un dôme en utilisant les Shaders.
L'objectif de ce projet est donc de réussir à afficher une scène 3D sur un dôme, et que, quel que soit le point de vue, l'image ne soit pas déformée.
La recherche a porté sur la déformation de l'image, le changement de point de vue, et pour finir, sur une technique qui regroupe les deux recherches précédentes et qui est programmable pour du temps réel.
La décision a été prise d'exploiter cette troisième recherche et de proposer de déformer la scène 3D à l'aide des shaders.
Pour cela, une recherche mathématique a été effectuée pour déterminer les transformations à opérer.
La phase de développement n'a toujours pas eu lieu et sera résumée en fin de projet.
Une idée d'ouverture vers un futur projet a été trouvée, afin de capter la forme du support en temps réel pouvoir projeter la scène 3d où l'on veut.

Remerciements

Je remercie M. Picarougne qui a eu la gentillesse de prendre sur son temps pour me faire découvrir le monde de la programmation des shaders et de me proposer ce projet de recherche sur ce thème qui me captive.

Je remercie également l'ensemble des professeurs de Polytech' Nantes responsables des divers cours portant de près ou de loin sur l'imagerie informatique ou tout autre thème exploité dans ce projet.

Table des matières

1

Introduction

1.1 Présentation de la problématique

problématique : Comment réaliser l'affichage d'une scène en 3D sur un support de géométrie quelconque sans souffrir de déformations visuelles ?
Il faut comprendre les enjeux de ce domaine. On se propose des les lister brièvement, mais il faudra se référer à la partie "Etat de l'Art" pour une description plus précise.
L'affichage d'un rendu 3D sur un écran plat présente un inconvénient : l'angle de l'utilisateur par rapport aux deux bords de l'écran doit être sensiblement égal à l'angle d'ouverture de la caméra, sinon des déformations apparaissent à l'approche des bords.
De plus en plus de professionnels et de joueurs se sont tournés vers la multiplication des écrans afin de tendre vers un affichage à 180°
L'affichage sur un écran sphérique (plus généralement un dôme) est une solution qui permet de ne plus constater de sauts ou de différences d'éclairage, et donc une représentation sans déformations.
Il réside un inconvénient global qui se présente avec l'affichage sur un dôme : il est nécessaire de se placer à un point particulier pour ne pas observer de déformations.
Ce projet a pour but de réaliser les déformations

adaptées au point de vue de l'utilisateur afin de percevoir la perspective de la meilleur qualité quel que soit son emplacement dans le dôme.

1.2 Objectifs poursuivis

Une solution serait d'adapter alors l'image au point de vue de l'utilisateur. Dans le cas d'un film ou de tout autre rendu 2D, il n'est pas possible d'arriver à un résultat satisfaisant car l'information au delà des bordures de l'écran n'existe pas. Mais dans le cas d'un rendu 3D, l'information existe mais n'est pas affichée. De plus, il est possible de déformer à volonté ce rendu.
Ce projet a pour but de réaliser ces déformations afin que l'image vue par l'utilisateur ne soit pas déformée à cause de l'emplacement de son point de vue.
La difficulté que présente cette solution est la suivante : déformer un rendu 3D est lourd au niveau du temps de calcul. Mais les cartes graphiques récentes permettent de réaliser de nombreuses modifications sur le rendu de façon suffisamment instantanée pour du temps réel.
La solution envisagée est d'utiliser la carte graphique, et particulièrement les shaders, pour réaliser ces déformations.
Pour ce projet de recherche, nous nous concentrerons sur l'affichage sur un dôme et nous définirons arbitrairement l'emplacement du point de vue de l'utilisateur (non confondu avec le centre du dôme).

1.3 Travail réalisé

La recherche a porté sur un vaste domaine, et on s'est efforcé de se cantonner à des propositions précises afin de ne pas faire une étude de toutes les mathématiques de la perspective.
Elle a porté sur deux domaines principaux, la déformation et la perspective, et a permis d'aboutir à une alliance des deux au travers d'un procédé de transformation de la scène 3D.
La réalisation a débuté au mois d'Octobre, avec l'intégration des librairies Glew et Glut et la réalisations de plusieurs programmes afin d'assurer le fonctionnement des shaders.

1.4 Contribution

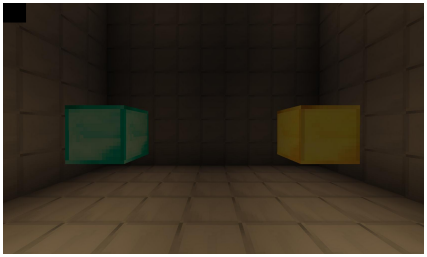
La solution issue est basée sur la déformation de la scène 3D de façon à ce qu'elle soit perçue de façon non déformée et avec les bonnes perspectives pour une point de vue en particulier, indépendant du centre du dôme. La technique est aussi indépendante de la forme du support. Pour assurer le temps réel, une solution informatique est mise en avant : la programmation des Shaders, exploitant le GPU.

1.5 Plan de l’étude

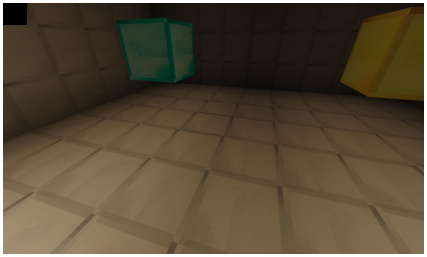
2

État de l'art

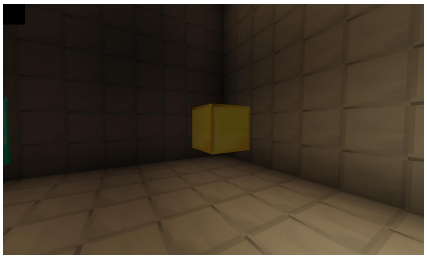
Rappelons tout d'abord la problématique :
problématique : Comment réaliser l'affichage d'une scène en 3D sur un support de géométrie quelconque sans souffrir de déformations visuelles ?
Cette problématique s'inscrit dans un domaine vaste et il faut en comprendre les enjeux principaux avant d'étudier les propositions.
L'affichage d'un rendu 3D sur un écran plat présente un inconvénient : l'angle de l'utilisateur par rapport aux deux bords de l'écran doit être sensiblement égal à l'angle d'ouverture de la caméra, sinon des déformations apparaissent à l'approche des bords. C'est aujourd'hui un problème bien plus important avec la démocratisation des écrans larges. Ces derniers ont offert la possibilité d'accroître le champs de vision, présentant des déformations sur les bordures bien plus importantes. (cf figure 2.1)



(a) Les deux cubes colorés semblent à la même distance de l'objectif



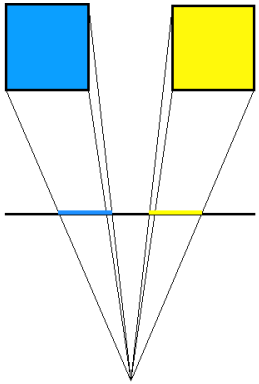
(b) Après rotation, le cube jaune semble plus proche



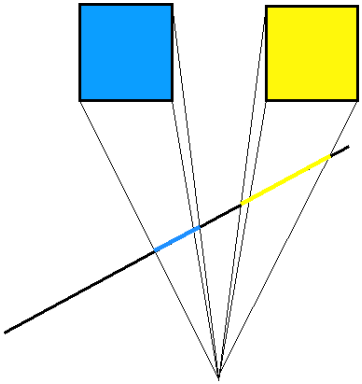
(c) Après une autre rotation, le cube jaune semble plus éloigné

FIGURE 2.1 – Un exemple visuel de déformation sur les bords

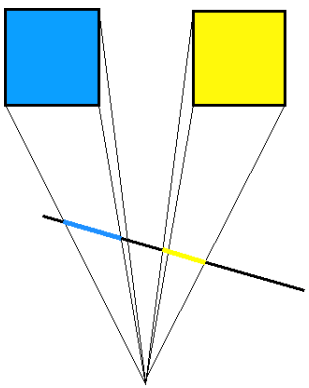
Lors de la rotation, le centre de projection reste le même, **c'est le plan de projection qui tourne autour de ce centre en restant à la même distance de celui-ci**. Les lignes de projection restent donc également les mêmes, mais les intersections qu'elles ont avec le plan de projection sont différentes. Pour comprendre le phénomène, reproduisons-le en écrasant une dimension. (cf figure 2.2)



(a) Les deux projections (bleu et jaune) sont des segments de même taille



(b) Après rotation, la projection jaune est un segment plus grand que celui de la bleue



(c) Après une autre rotation, c'est l'inverse qui se produit

FIGURE 2.2 – Explication du phénomène de déformation sur les bords

On est en droit de remarquer que, de toute façon, les bords de l'écran se trouvent plus loin de nous que son centre, et donc qu'il est normal que les objets s'y trouvent plus grands. [10] Seulement on entre dans la supposition selon laquelle le point de vue de l'utilisateur se trouve à une distance bien définie de l'écran. Plus on s'éloigne, et plus le rapport entre ces distances diminue, ce qui n'est pas le cas du rendu affiché.

De plus en plus de professionnels et de joueurs se sont donc tournés vers la multiplication des écrans. Les écrans latéraux servent alors à afficher les flancs du champs de vision anciennement déformés. (cf figure 2.3)

(a) L'installation de Steve Ferris

(b) L'installation de Luciano Napolitano

FIGURE 2.3 – Installations de plusieurs écrans en arc de cercle dans le cadre de la promotion de l'add-on WideView pour Flight Simulator X [4]

Mais cet artifice présente toujours le problème suivant : les bordures de chaque écran admettent des déformations. Pour le comprendre, étudions le plan de l'installation de Luciano Napolitano [4]. (cf figure 2.4)

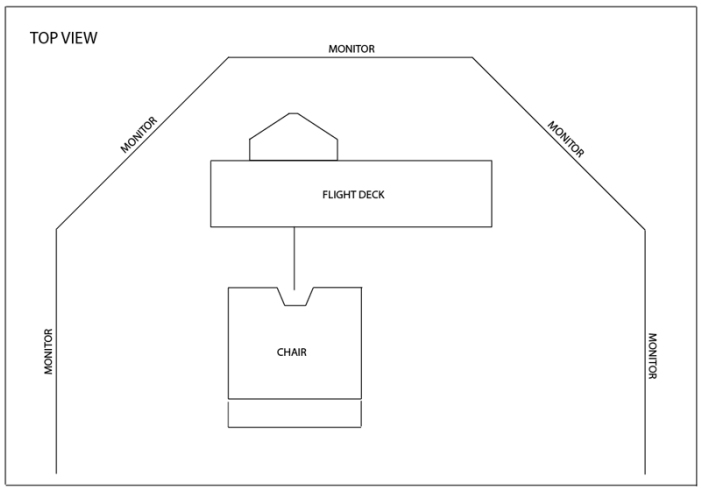


FIGURE 2.4 – Plan de l'installation de Luciano Napolitano

On corrige une erreur en plaçant la caméra bien à la même distance de ces écrans, et on observe le phénomène lors de la projection d'un objet sur deux écrans à la fois. (cf figure 2.5)

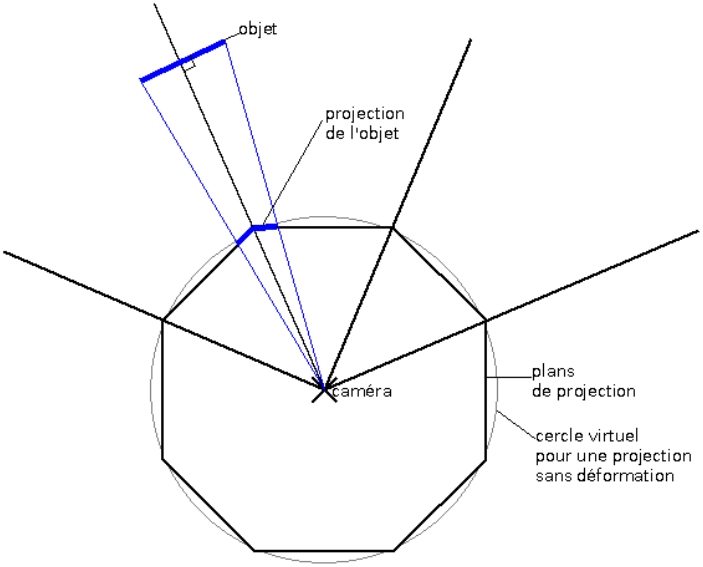


FIGURE 2.5 – Cas de déformation sur une installation de plusieurs écrans

La luminosité perçue n'est pas la même partout car les points de la surface des écrans ne sont pas tous à la même distance du point de vue de l'utilisateur.

De plus, l'objet est affiché de plus en plus proche à mesure que l'on approche de son centre et de plus en plus loin à mesure que l'on approche de ses bords. C'est un phénomène normal que l'on observe de façon naturelle. Mais ici, cette variation est linéaire, alors qu'elle devrait suivre un progression polynomiale. De ce fait, un cercle a été tracé afin de représenter le véritable volume de projection afin d'éliminer cette déformation.

De ce fait, l'affichage sur un écran sphérique (plus généralement un dôme) est une solution qui permet de ne plus constater ces déformations. Le demi-dôme présente aussi l'avantage de couvrir le champ de vision de l'être humain. Il faut savoir que ce n'est pas tout à fait le cas, car un œil couvre moins de 180°, mais les deux yeux couvrent un angle supérieur à 180°. Le demi-dôme couvre donc le champ de vision commun aux deux yeux. [10] L'objectif fisheye, dans le domaine de la photographie, propose un rendu très déformé sur un écran plat mais qui s'adapte à un dôme, à condition que la couverture du dôme soit de l'angle de la prise de vue. (cf figure 2.6)

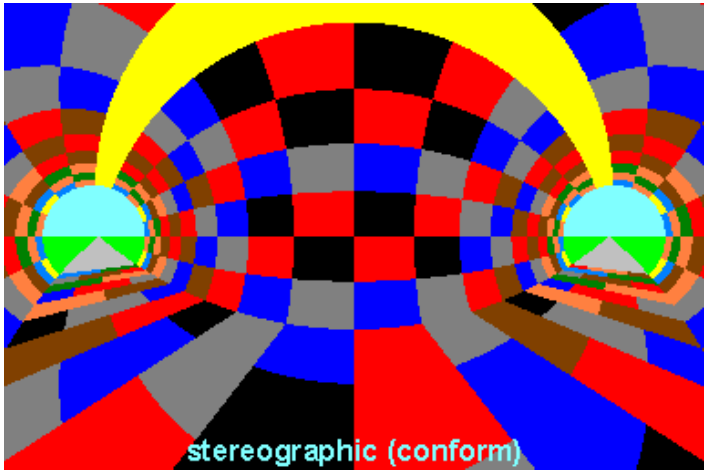


FIGURE 2.6 – Objectif fisheye, prise de l'intérieur d'un tunnel en ligne droite, ouverture d'un angle proche de 180° (image de wikipedia)

Il réside un inconvénient global qui se présente avec toutes ces techniques : il faut admettre que le point de vue de l'utilisateur se trouve à un endroit bien particulier pour ne pas percevoir de déformations. [11] Dans le cas des écrans plats, le cerveau humain s'est habitué à une grande marge spatiale pour placer son point de vue. Par exemple, plusieurs spectateurs d'une salle de cinéma placés à des endroits différents verront une image différente, mais leur cerveau se reconstituera sensiblement la même image.
Le dôme présente deux problèmes :

1. Il est difficile au cerveau humain de se reconstituer une image non déformée.
   Par exemple, des personnes placés à plusieurs endroits dans un planétarium auront des expériences trop différentes les unes des autres. (cf figure 2.7)

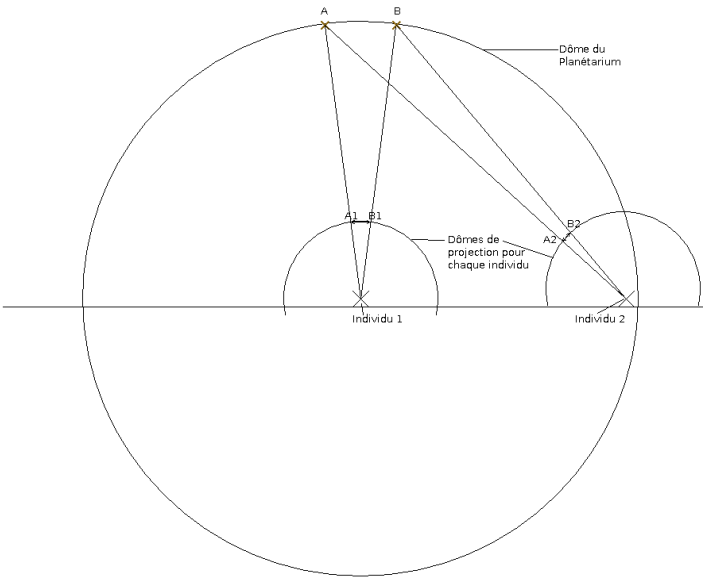


FIGURE 2.7 – Expériences visuelles différentes dans un planétarium

Nous plaçons deux individus à différentes positions dans un planétarium, nous virtualisons deux dômes de projection pour chacun (étant donné que c'est en angle que l'on raisonne,

nous pouvons manier des cercles dont le centre est le point de vue, et comme cela, on perçoit plus facilement le phénomène). Si l'on place deux objets sur le dôme du planétarium, nous remarquons que l'individu 1 les verra éloignés d'une distance différente de celle vue par l'individu 2.
Renommons l'individu 1 en O et l'individu 2 en Q.
De façon mathématique, le phénomène s'explique du fait suivant :

$$\widehat{AOB} \neq \widehat{AQB} \tag{2.1}$$

Pour le prouver :
ABQO est un quadrilatère quelconque
AQ et BO sont ses diagonales
Donc à moins d'être un parallélogramme, les angles AOB et AQB ne sont pas égaux.
Si ABQO était un parallélogramme, les propriétés suivantes sont vrais :

$$[AQ] \neq [AO] \tag{2.2}$$

$$[BQ] = [AO] \tag{2.3}$$

$$[AQ] \neq [BQ] \tag{2.4}$$

Or on sait que pour l'individu 1, les deux objets A et B sont à la même distance de celui-ci.
Mais pour l'individu 2, nous venons de montrer que ce n'est pas le cas.
La déformation est donc également présente dans le cas du parallélogramme.
Donc il faut se trouver au centre du dôme pour ne pas voir de déformation.

2. Il est impossible de placer le point de vue de l'utilisateur à l'emplacement idéal, puisqu'il s'agit de l'emplacement du projecteur dans le cas du dôme de Polytech et de plusieurs autres dispositifs du même type.
   Pour ce problème d'impossibilité, une solution reviendrait à réaliser un dôme sans projecteur central. C'est un sujet pour d'autres branches de l'ingénierie, sortant du cadre de cette recherche.

Avec tous ces enjeux relevés, nous pouvons étudier des propositions.

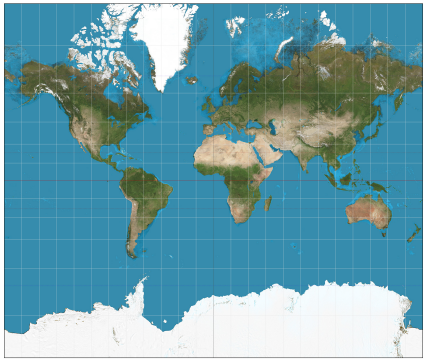
2.1 Proposition *1* : Déformation de l'image

2.1.1 Présentation

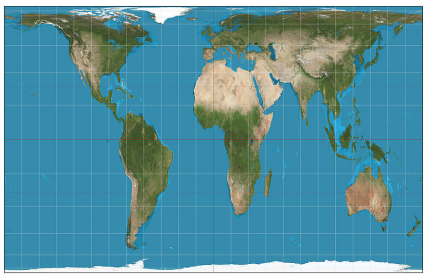
La déformation de l'image est une approche qui devrait permettre d'afficher une image non déformée sur le support.

Si l'on considère que le problème de la déformation provient de la projection de la scène sur un support qui n'a pas la forme de la géométrie sur laquelle a été faite la projection, alors peut-être qu'une déformation de l'image projetée peut lui faire épouser la forme sur le support.
De nombreuses personnes se sont penchées sur ce type de déformation [2] :

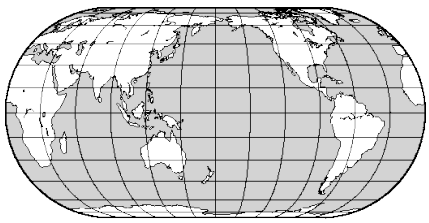
- Gérard Mercator, un mathématicien et géographe du XVIème siècle, a tenté d'établir une carte du monde connu et la projetant sur un cylindre tangent à l'équateur afin de ne pas déformer les angles (le cylindre peut être vu comme un plan). Mais les distorsions sont de plus en plus importantes à l'approche des pôles, vers lesquels les proportions horizontales ne sont plus respectées. [3]
- Arno Peters, un historien et cartographe allemand, a proposé sa projection en 1974. Elle reprend le principe de la projection de Mercator avec la particularité de compresser et étendre verticalement la projection en fonction de la latitude. Ceci compense les problèmes de proportions horizontales et c'est au niveau des aires que les proportions sont alors gardées.
- La projection Eckert IV (1980) reprend la projection de Mercator, mais les parallèles sont des lignes droites non équidistantes. Ceci permet aux méridiens d'avoir une allure elliptique. Le problème de cette projection est qu'on échappe alors à la projection sur un plan.
- Il existe de très nombreuses autres projections du monde sur un plan, et chacune présente un ou plusieurs inconvénient(s).



(a) Mercator



(b) Peters



(c) Eckert IV

FIGURE 2.8 – Quelques techniques de projection du monde sur un plan (images de wikipédia)

2.1.2 Analyse

Ces solutions présentées par la figure 2.8 ont toutes un problème, mais ceci n'a pas freiné le monde du rendu 3D. En effet, dans de très nombreuses applications 3D, la scène est projetée sur un plan pour être ensuite affichée sur un plan. La projection sur un dôme est, quant à elle, rendue possible grâce au rendu *fisheye* qui est une proposition d'affichage sur un dôme a priori satisfaisante .

Dans le même esprit que la projection Eckert IV, nous avons alors une image projetée sur un dôme, affichée déformée sur un plan, mais affichée non déformée sur un dôme.
Se pose alors le problème suivant : l'image projetée dans le dôme ne semble pas déformée pour un utilisateur dont le point de vue se situe au centre de projection du dôme. Mais comment faire lorsque le point de vue de l'utilisateur se situe ailleurs ? Étudions le problème à l'aide d'un schéma. (cf figure 2.9)

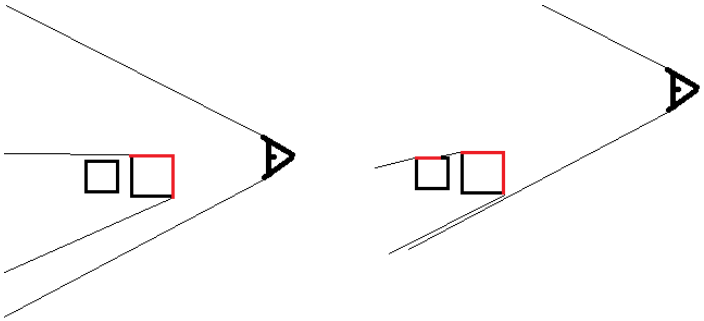


FIGURE 2.9 – Ce schéma montre que la partie visible de la scène 3D (en rouge) n'est pas la même selon deux angles de vue

2.1.3 Limites

La proposition concrète retenue est ainsi de récupérer une image au format FishEye (applicable à un dôme) ce qui la déforme déjà d'un point de vue sur un plan. L'application sur le dôme peut alors se faire sans déformation.
Mais la problématique pose la condition que l'utilisateur puisse avoir un différent angle de vue dans le dôme.
En changeant l'angle de vue, il peut y avoir création d'informations car de nouveaux éléments de la scène peuvent être vus, et d'autres peuvent être alors cachés. La déformation de l'image ne permet pas d'obtenir de tels résultats, c'est pourquoi ce n'est pas une solution que l'on peut retenir.

2.2 Proposition *2* : Déplacer la vue

2.2.1 Présentation

La première proposition impliquait de déformer l'image, mais on a vu qu'on désirait un point de vue différent sur la scène. C'est le même problème qui se pose alors qu'on regarde par une fenêtre : si l'on monte sur un tabouret, le décors vu n'est plus le même, non seulement le champ de vision est différent, mais en plus la perspective a changé. Une proposition très simple que l'on peut faire alors : pourquoi ne pas déplacer la caméra dans la scène pour s'adapter au nouveau point de vue de l'utilisateur ?

La solution a déjà été proposée par la société NaturalPoint qui a sorti un produit nommé le Track Ir. C'est un casque qui détecte les mouvements de la tête sur six degrés de liberté. Le dispositif envoie alors l'information sur les mouvements effectués et c'est la caméra virtuelle qui bouge en fonction. On peut alors faire bouger strictement la caméra de la même façon que les mouvements de tête. Mais dans l'intérêt de ce projet, on peut aussi faire effectuer

à la caméra des mouvements qui reviendraient à effectuer les transformations pour donner cette impression de voir par une fenêtre. Le détail de ces calculs est donné par la suite dans la partie "Conception détaillée".

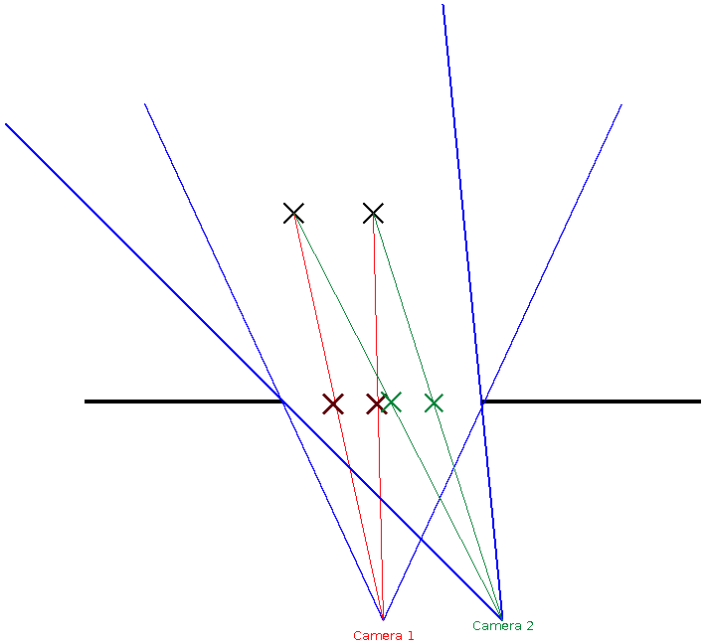


FIGURE 2.10 – Les deux croix noires sont projetées sur un plan pour être vues par la caméra 1. Ce sont alors les deux croix rouges.
Si l'on déplace la vue, on a alors la caméra 2. Les deux croix noires devraient alors être projetés sur les deux crois vertes.

2.2.2 Analyse

L'inconvénient de cette solution vient simplement des différentes formes que peut avoir le support. En effet, la projection du décors 3D se fait de la même façon sur la caméra, or, selon l'angle de vue, le support n'est plus le même. Par exemple, l'écran rectangulaire devient un trapèze, et le demi-dôme devient une forme quelconque et elliptique.

Démonstration par un contre-exemple. (cf figures 2.10 2.11 2.12)

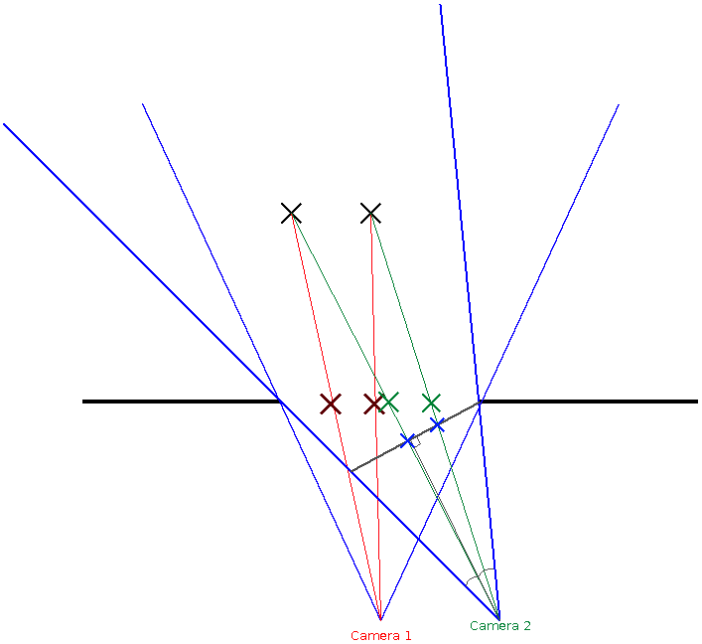


FIGURE 2.11 – Or, le plan de projection de la caméra 2 n'est plus le même, il s'agit de la ligne grise. les deux croix noires sont projetées sur les deux croix bleues.

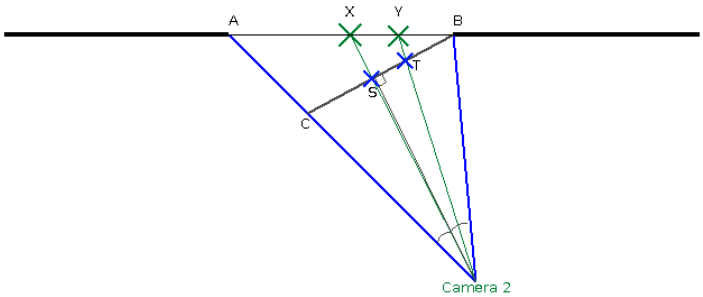


FIGURE 2.12 – Nous sommes alors dans ce cas de figure.

Nous voulons vérifier la double égalité que l'on nomme (E2) :

$$\frac{AX}{AY} = \frac{CS}{CT} \tag{2.5}$$

$$et \frac{BY}{BX} = \frac{BT}{BS} \tag{2.6}$$

En remarquant que l'on peut confondre d'abord Y avec B, puis X avec A, alors (E2) équivaut à :

$$\frac{AB}{AX} = \frac{CB}{CS} \tag{2.7}$$

$$et \frac{AB}{AY} = \frac{CB}{CT} \tag{2.8}$$

D'après le théorème de Thalès : si cette double égalité était vraie, alors XS et YT seraient parallèles.
Or ces deux droites se coupent en un point : la caméra 2.
Donc (E2) est fausse

2.2.3 Limites

La proportion n'est pas conservée, et il y a une déformation de l'image visible à l'oeil. Et comme on cherche à supprimer les déformations visibles sur tout type de support, y compris un support plat, on ne peut pas garder cette proposition. Il faut que le changement d'angle de vue s'opère sur une scène 3D qui se déforme elle-même en fonction de ce changement, afin de conserver toutes les propriétés de la projection (perspective, distances, angles). C'est pourquoi la troisième proposition entre en jeu.

2.3 Proposition *3* : Modifier la scène 3D

2.3.1 Présentation

La première proposition impliquait de déformer l'image, mais on a vu qu'on désirait un point de vue différent sur la scène.
La seconde proposition impliquait de déplacer la vue, mais on a vu que cela ne corrigeait pas les déformations.
Mais si l'on alliait les deux propositions, si l'on modifiait la vue et que l'on tordait aussi l'image, peut-être que l'on atteindrait l'objectif posé.
On se proposer donc de modifier la scène 3D de façon à lui appliquer les transformations nécessaires à la simulation du changement de point de vue et à la torsion de l'image.

La technique employée est issue d'une recherche relativement ancienne, puisque c'est le principe de l'anamorphose ou "horizontorium" : [8] [12]
Le principe consiste à créer des distorsions sur une figure de façon à ce qu'elle soit visible selon les bonnes proportions et les bonnes perspectives selon un point de vue en particulier.
L'avantage de la technique est que les distortions effectuées sur l'image prennent en compte la forme du support sur laquelle elle est appliquée. Les limites de l'anamorphose est que l'image non déformée n'est visible que depuis un unique point de vue, sinon, c'est une adaptation du cerveau qui opère la reconstitution de l'image quand on se trouve à des alentours raisonnable du point de vue voulu. Et au delà, l'image tend à être vue d'une façon totalement déformée pour l'interpréter convenablement. Ceci dit, il a été évoqué pour ce projet qu'une caméra pourrait suivre la position du point de vue de l'utilisateur. Si cette proposition débouche sur une solution technique qui admet du temps réel, l'adaptation à un point de vue mobile est alors possible. (cf figure 2.13)

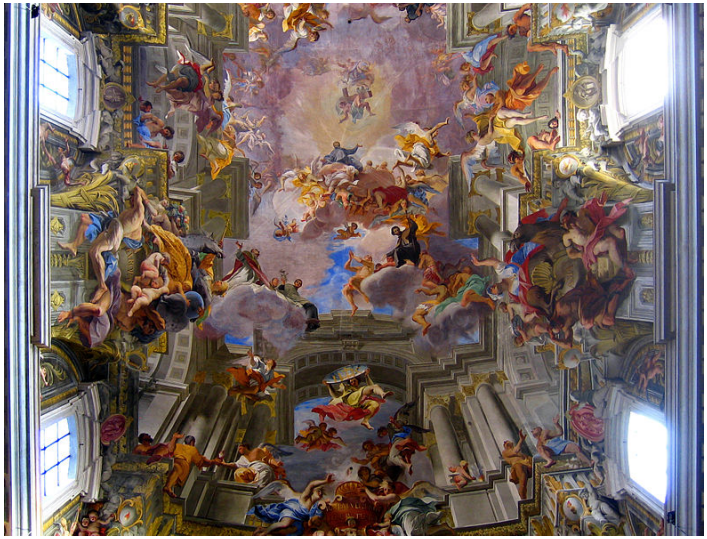


FIGURE 2.13 – Le "trompe l'oeil" du plafond de l'église de Saint-Ignace-de-Loyola de Rome, peint en 1685 (image de wikipedia).
On remarque que la perspective de la peinture ne suit pas parfaitement la perspective de la pièce, soit à cause du point de vue qui n'est pas parfaitement celui prévu, soit parce que la peinture a elle-même des défauts mathématiques.

2.3.2 Analyse

On écrase une dimension. La figure suivante (cf figure 2.14) illustre la situation de base dans le monde virtuel de la projection d'un objet sur un dôme. On récupère les coordonnées de la projection sur la matrice des pixels du dôme (ce qu'on appelle un fragment) et on l'affiche aux coordonnées correspondantes sur le dôme dans le monde réel (ce qu'on appelle un pixel).

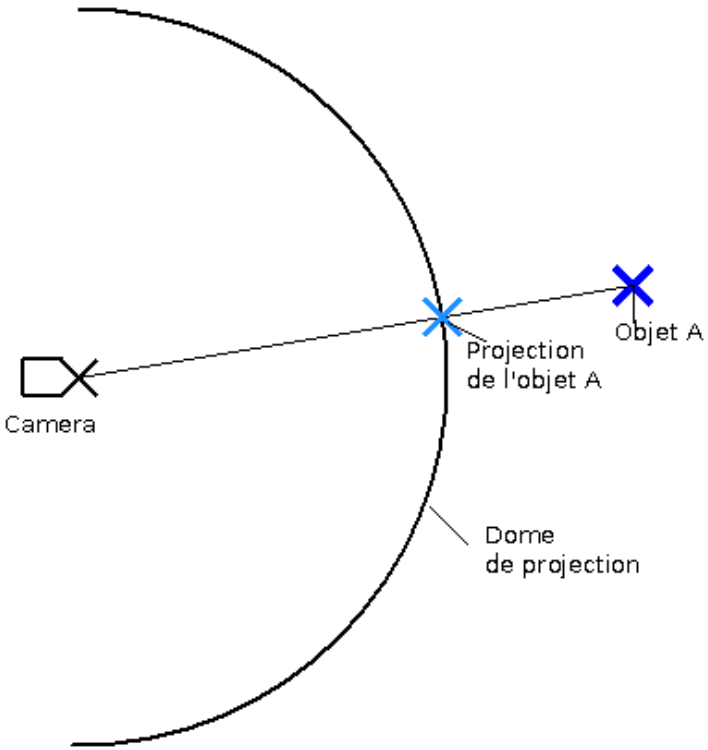


FIGURE 2.14 – Situation de base dans le monde virtuel de la projection d'un objet sur un dôme

Imaginons maintenant que l'utilisateur a un point de vue qui n'est pas au centre du dôme. Plaçons le nouveau point de vue qui est l'emplacement virtuel de l'individu par rapport au centre qui est décrit par la caméra. L'individu voyait jusqu'ici l'objet projeté à l'emplacement que nous avions déterminé. Mais ce que nous voulons, c'est que cet objet soit projeté sur un nouvel emplacement. (cf figure 2.15)

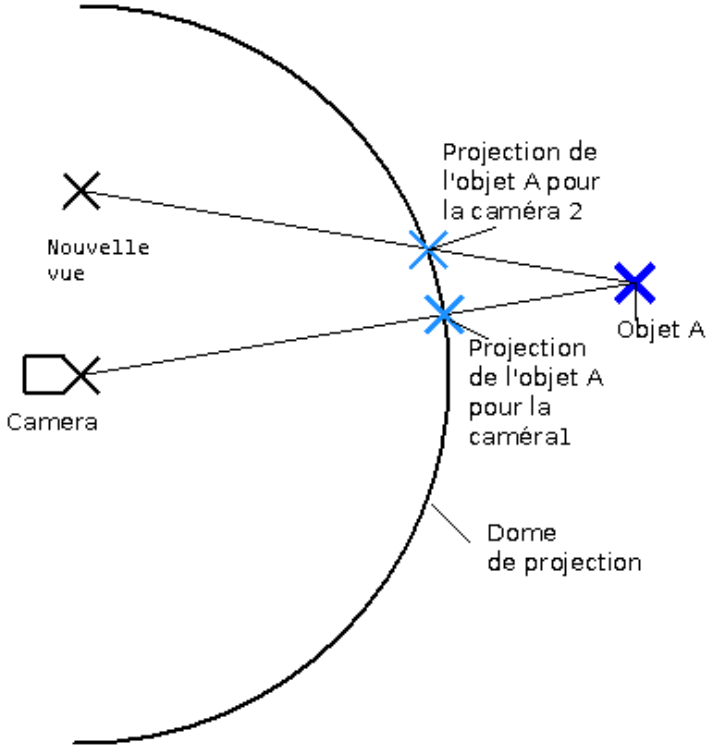


FIGURE 2.15 – Nouvel emplacement de caméra, nouvelle projection

Pour que l'objet soit projeté à cet emplacement, il faudrait que la caméra visualise cet objet de façon à ce qu'il soit projeté sur ce nouveau point de projection. Pour se faire, il suffit de déplacer l'objet dans l'alignement de la caméra et du nouveau point de vue.

Il reste le problème de la distance à laquelle nous le plaçons. Le nouveau point de vue voit cet objet à une distance différente de celle vue par la caméra. Il faut donc que l'objet soit placé à la même distance de sa nouvelle projection. La figure ci-dessous illustre le procédé. (cf figure 2.16)

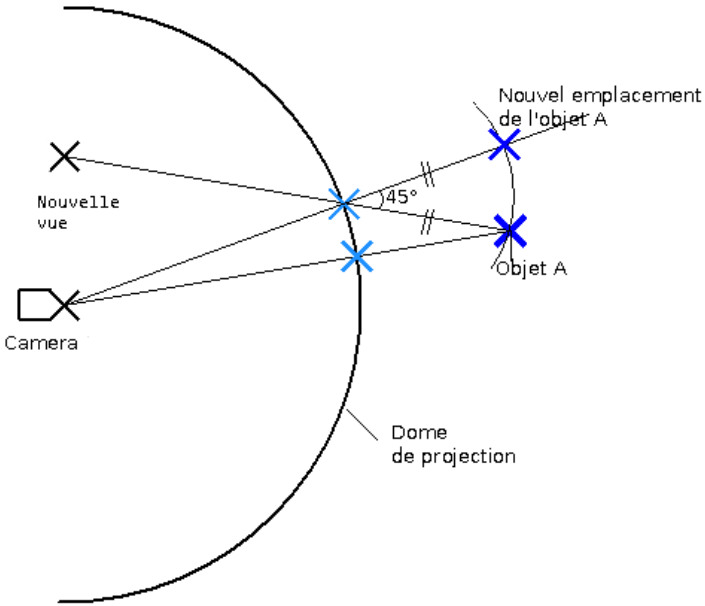


FIGURE 2.16 – Déplacement de l'objet

Il reste cependant un problème, dans des cas de figure où l'on manie des primitives de très grande taille, on se rend compte que la transformation ne peut pas s'effectuer uniquement sur les sommets. Il faudrait pouvoir diviser de façon dichotomique la primitive de façon à obtenir plusieurs sommets qui seront alors déplacés, et la forme reconstruite ne sera alors plus une forme pouvant être représentée par une primitive. (cf figure 2.17)

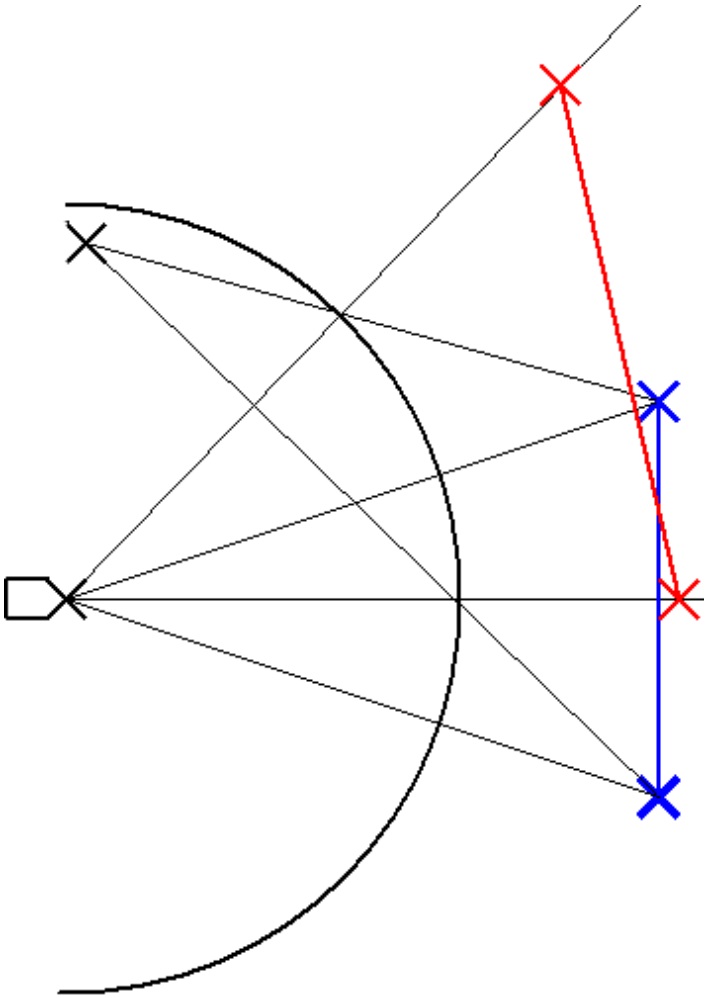


FIGURE 2.17 – Exemple de la reconstruction d'une primitive (un segment dans le monde 2D) et de sa transformation en n'opérant que sur les sommets.

Afin de nous préparer à montrer mathématiquement que nous pouvons déterminer ce nouvel emplacement, nous allons nommer tout ce qui peut nous être utile. Nous colorisons en vert les données connues, en orange les données inconnues et en rouge la donnée inconnue que l'on cherche. (cf figure 2.18)

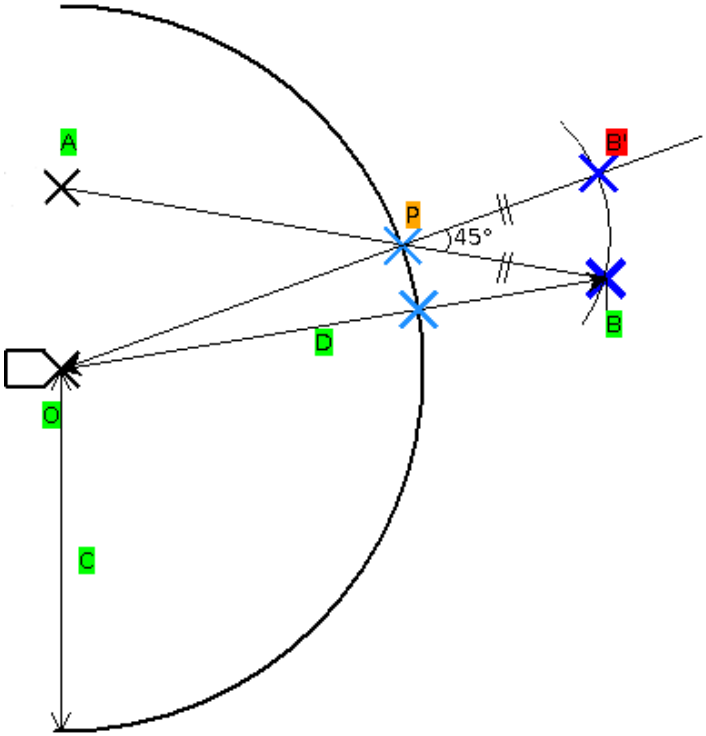


FIGURE 2.18 – Simplification du problème pour une approche géométrique

Nous reverrons dans le détail cette proposition et comment déterminer les nouvelles coordonnées d'un objet.

Cette proposition n'est pas complète sans avoir présenté un moyen informatique de déplacer nos objets.

La méthode la plus simple serait de récupérer l'ensemble des sommets de la scène et les déplacer vers leurs nouvelles coordonnées avant de les afficher. C'est un procédé trop coûteux en temps car il s'agit d'utiliser le pipeline à fonction fixe.
Ce que propose les librairies OpenGL2.0 ou les dernières versions de Direct3D, ce sont les shaders programmables. Selon le guide officiel OpenGL2.0, le rendu graphique utilisant le pipeline fixe s'agence de la façon suivante (cf figure 2.19) : [5]

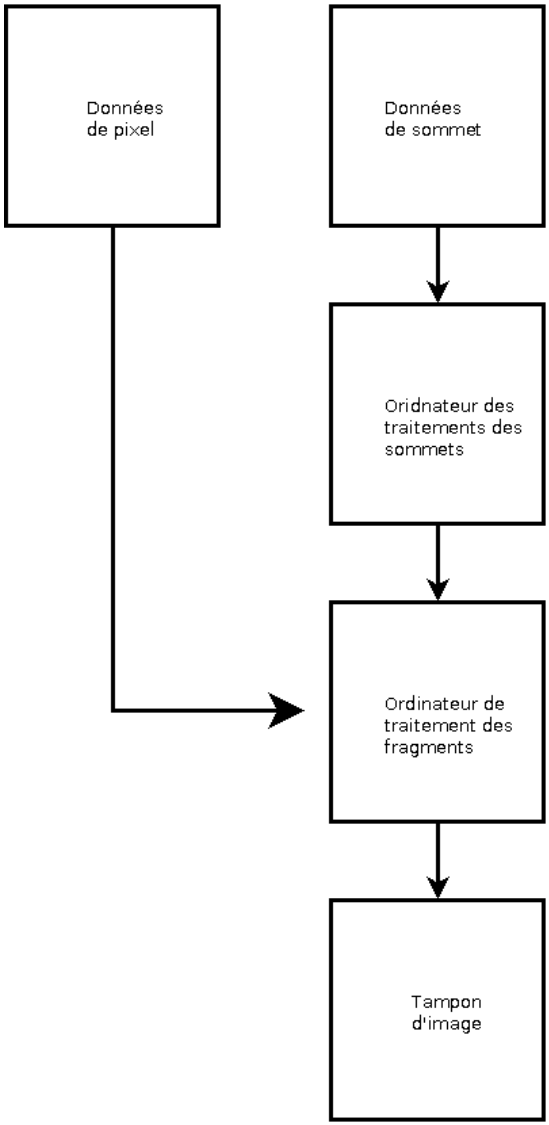


FIGURE 2.19 – Pipeline à fonction fixe d'OpenGL.

Dans ce livre, on décrit le fonctionnement d'OpenGL comme étant sur deux ordinateurs, un gérant les sommets et l'autre les fragments. On est capable de contrôler le fonctionnement des deux ordinateurs en modifiant la façon dont ils sont agencés entre eux, mais on ne peut pas modifier le fonctionnement interne de ces ordinateurs. C'était un fait qui n'est plus d'actualité désormais. On peut désormais programmer le fonctionnement de ces ordinateurs, que l'on appelle Pipeline.
Ce qui nous intéresse, c'est de programmer un pipeline afin de gérer nous-même les transformations à effectuer. OpenGL 2.0 offre la possibilité d'un piepline de shading programmable via un langage : le GLSL (OpenGL Shading Language)
Nous pourrons utiliser deux types de shaders parmi ceux existant selon la liste donnée par les specifications du GLSL [7] :

1. **Le Vertex Shader** : On peut programmer le processeur de vertex pour modifier les attributs d'un vertex (sommet en français). Quand ce code est compilé, on obtient un vertex shader éxécutable qui fonctionne sur le processeur de vertex. Il fonctionne alors pour chaque vertex un par un.
2. **Le Geometry Shader** : On peut programmer le processeur de géométrie pour modifier les attributs d'une primitive construite à partir de vertex. Quand ce code est compilé, on obtient

un geometry shader éxécutable qui fonctionne sur le processeur de géométrie. Il est alors possible, en sortie, de trouver un nombre différent de vertex par rapport à ce qui était en entrée, car il peut y avoir une multiplication des primitives. Ces vertex et primitives passent alors par un pipeline spécial.

Ainsi, avec le Vertex Shader, nous pourrons changer les coordonnées de tous les sommets, tandis qu'avec le Geometry Shader, nous pourrons transformer les primitives quand elles seront trop grandes.

2.3.3 Limites

La démonstration des limites a besoin de la formalisation et de l'approche mathématique. Se référer donc à la partie Proposition.

2.4 Récapitulatif

La première proposition consistait en une torsion de l'image. C'est un procédé qui admet de nombreuses limites, comme en témoigne la longue recherche sur l'application d'une projection sur une figure géométrique appliquée à une autre figure géométrique. Le principal problème de cette solution est que la perspective doit pouvoir changer, et la surface visible doit ne plus être la même après le changement d'angle de vue, ce qui n'est pas possible avec la déformation simple de l'image.
La seconde proposition consistait en un déplacement de la caméra pour suivre la vue. Cependant, la surface de projection de la caméra doit aussi pouvoir changer afin de ne pas tordre l'image aperçue après rendu.
La troisième proposition est finalement un mélange des deux premières mais utilisant une technique différente : il s'agit de modifier la scène 3D en fonction d'une caméra virtuelle décentrée et de la surface de projection de la véritable caméra. Elle s'appuie sur un calcule mathématique de détermination des coordonnées et sur les shaders programmables pour modifier la scène 3D.

avantages	torsion image	déplacement caméra	transformation scène (shaders)
adaptation au support	oui mais limites	non	oui
respect de la perspective	non	oui	oui
rapidité de rendu	très rapide	très rapide	rapide si utilisation des shaders

TABLE 2.1 – Tableau comparatif des propositions.

3

Propositions

La proposition retenue aura été la troisième car les deux premières présentent chacune un inconvénient qui ne répond pas à la problématique du projet. Rappelons brièvement les deux premières et l'idée générale pour les implémenter, avant de présenter de façons développée la troisième.

3.1 Proposition *1* : Déformation de l'image

La première idée était de déformer l'image afin qu'elle s'adapte à la surface. Pour cela, nous prenions l'exemple d'un plan appliqué à un dôme.

Nous pouvons faire une approche d'application en remarquant que le dôme n'est qu'une matrice de pixels. On peut alors appliquer l'algorithme suivant : *Pour chaque fragment de coordonnées (f_i ; f_j), l'afficher au pixel de coordonnées (p_i,p_j) du dôme.*

Mais comme dit dans la partie Etat de l'Art, l'objectif est partiellement atteint car on ne souhaite pas que le point de vue de l'utilisateur se trouve au centre du dôme. Par exemple, c'est une chose impossible avec le dôme de l'école, car à son centre se trouve le projecteur.

3.2 Proposition *2* : Déplacement de la caméra

La seconde idée était de déplacer la caméra selon l'emplacement du point de vue de l'utilisateur.
Soient les coordonnées du centre du dôme-écran dans le monde réel : (xr,yr)
Soient les coordonnées de la caméra dans le monde virtuel : (xv,yv)
Soient les coordonnées du point de vue de l'utilisateur dans le monde réel : (xu, yu)
Les coordonnées du nouvel emplacement de la caméra dans le monde virtuel sont alors : (xv+k(xu-xr),yv+k(yu-yr))
avec k réel quelconque.
Le problème technique du changement de surface de projection est une limite à cette solution qui voit alors une déformation de l'image malgré le respect de la perspective. C'est en voulant résoudre ce problème que l'on va étudier la troisième proposition.
Ces deux premières propositions peuvent être implémentées sans grande difficulté, mais leurs limites nous poussent à passer à la troisième proposition et étudier plus en détail son implémentation car c'est elle qui a été retenue pour ce projet.

3.3 Proposition *3* : Modifier la scène 3D

3.3.1 Idées préliminaires

L'idée pour effectuer cette déformation de l'image en plus du changement d'angle de vue est de déformer la scène 3D. [12]

3.3.2 Formalisation

Reprenons le schéma que nous avions établi dans la partie Etat de l'Art. (cf figure 3.1)

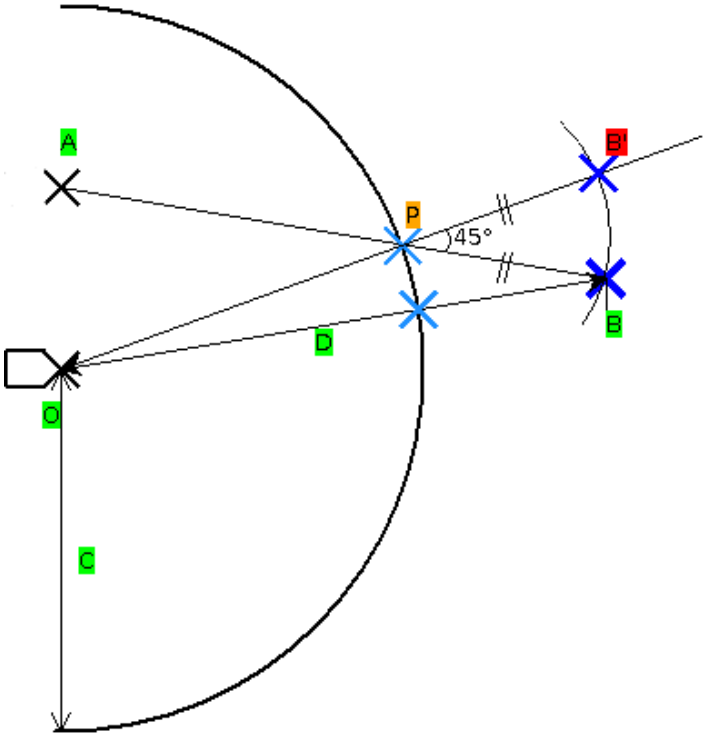


FIGURE 3.1 – Simplification du problème pour une approche géométrique

Le but est de déterminer les coordonnées du point B'.

On connait :

$$O(0,0) \tag{3.1}$$
$$A(xa,ya) \tag{3.2}$$
$$C \tag{3.3}$$
$$B(xb,yb) \tag{3.4}$$
$$D \tag{3.5}$$

On cherche P(xp,yp).
De façon mathématique, on remarque que P est une solution (il peut y en avoir 1, 2 ou aucun) du système d'équation de la droite (AB) et du cercle de centre O et de rayon C :

$$yp = \frac{yb-ya}{xb-xa} \times xp + (ya - \frac{yb-ya}{xb-xa} \times xa) \tag{3.6}$$
$$\sqrt{xp^2+yp^2} = C \tag{3.7}$$

Simplifions certaines parties :

$$\alpha = \frac{yb-ya}{xb-xa} \beta = (ya - \frac{yb-ya}{xb-xa} \times xa) \tag{3.8}$$

On trouve la solution suivante :

$$xp = \frac{2\alpha\beta\sqrt{(2\alpha\beta)^2 - 4(\alpha+1)(\beta-C^2)}}{2(\alpha+1)} \tag{3.9}$$
$$yp = \frac{\frac{2\beta}{\alpha}\sqrt{(\frac{2^2\beta^2}{\alpha}) - 4(\frac{\alpha+1}{\alpha})(\beta^2-C^2)}}{2(1+\frac{1}{\alpha})} \tag{3.10}$$
$$\tag{3.11}$$

ou

$$\tag{3.12}$$

$$xp = \frac{-2\alpha\beta\sqrt{(2\alpha\beta)^2 - 4(\alpha+1)(\beta - C^2)}}{2(\alpha+1)} \quad (3.13)$$

$$yp = \frac{-\frac{2\beta}{\alpha}\sqrt{(\frac{2^2\beta^2}{\alpha}) - 4(\frac{\alpha+1}{\alpha})(\beta^2 - C^2)}}{2(1+\frac{1}{\alpha})} \quad (3.14)$$

Une application informatique permet de savoir quelle partie de la solution nous intéresse. (Il suffit de connaître les coordonnées de l'objet et de vérifier le signe)
Une approche informatique permet de connaître les coordonnées de P d'une autre façon, car on travaille dans un espace discret : Plutôt que de représenter le dôme sous la forme d'une équation, on peut le représenter sous la forme d'une matrice de profondeurs. Ceci permet notamment de construire d'autres géométries pour la surface de projection.
Voici les grandes lignes de l'algorithme à implémenter, en trois dimensions :

Pour chaque fragment du dôme, on a ses coordonnées X,Y et Z.
Pour déterminer si la droite passe par ce fragment, on vérifie si elle passe par la fenêtre que ce fragment constitue avec ses voisins de coordonnées (x,y+1),(x+1,y) et (x+1,y+1).
Si la valeure de Z trouvée à l'équation de la droite pour ces quatre coordonnées est comprise entre la valeure minimum et la valeur maximum de de cette fenêtre, alors la droite coupe cette fenêtre.
Si la droite coupe la fenêtre, on considère que le fragment dont est issue la fenêtre est coupé par la droite, c'est donc ce fragment qui constitue le point P(xp,yp,zp)

Il reste une dernière étape à la formalisation des calculs nécessaires à l'obtention des nouvelles coordonnées d'un objet. Cette étape, c'est la détermination de ces coordonnées de B' grace aux coordonnées du point P que l'on a trouvées.
Comme vu dans la partie Etat de l'Art, la distance [PB] doit être égale à la distance [PB'] et OPB doivent être alignés. On peut donc en déduire que :

$$\overrightarrow{OB'} = \overrightarrow{OP} \times \frac{(|\overrightarrow{OP}| + |\overrightarrow{PB}|)}{|\overrightarrow{OP}|} \quad (3.15)$$

3.3.3 Analyse

La limite de cette solution est l'utilisation de calculs mathématiques sur un espace qui est finalement discret. On l'a vu avec la façon de déterminer les paramètres de la surface de projection par exemple. L'autre limite, c'est le parcours de chaque fragment du dôme de projection qui est lourd en calcul. Enfin, la dernière limite est le manque de preuve selon laquelle la distance de [PB] est égale à la distance de [PB']. Nous l'avons déterminé arbitrairement. Ce qui est prouvé, par contre, c'est que l'ordre est conservé, car on conserve les proportions par ce

procédé. On peut donc garder cette solution avec cette démonstration manquante.

3.4 Conclusion

Cette proposition allie les deux premières propositions et permet d'éviter les déformations et conserver la perspective. C'est l'objectif du projet et cette proposition entre le mieux dans le cadre de cet objectif, même si elle use d'artifices, le rendu est celui voulu. Elle a le mérite d'utiliser un procédé connu pour déformer la scène 3D. Nous verrons l'utilisation technique des shaders par la suite de ce rapport.

4

Expérimentations et résultats

4.1 Modifier la scène 3D à l'aide des *shaders*

4.1.1 Développements

Pour réaliser la proposition retenue, il a été nécessaire de prévoir plusieurs contraintes techniques. D'un part celles liées à l'utilisation de la librairie OpenGL, ce qui a limité une certaine faisabilité, et d'autre part celles liées à la recherche de performances. Nous distinguerons donc deux grandes familles de contraintes :

– les contraintes générales, liées à l'utilisation de la librairie OpenGL et du langage GLSL
– les contraintes intrinsèques au choix de développement, et notamment concernant les performances.

les contraintes générales

– La version d'openGL disponible sur les systèmes d'exploitation Windows ne permet pas de réaliser les fonctionnalités prévues dans ce projet (notamment les shaders). Ce problème a été résolu par l'utilisation de la librairie GLEW qui permet de gérer les extensions OpenGL. Mais ceci n'a pas permis de conserver l'idée de

programmer les *geometry shaders*. Nous verrons par la suite en quel sens ce type de shader aurait pu améliorer le rendu envisagé.
- La programmation des *shaders* implique une programmation massivement parallèle. C'est une contrainte dans ce domaine qui implique d'éviter les boucles dont on ne connaît pas le nombre d'itérations, et qui incite donc à privilégier des suites de déclarations davantage calculatoires de façon à ce que chaque *vertex* soit traité efficacement et sans itérations inutiles.
- Une autre contrainte avec les shaders est que c'est un langage à part qui nécessite d'être compilé à part et est utilisé par le programme principal. Il faut donc s'habituer à ce type de programmation.

les contraintes intrinsèques au choix de développement

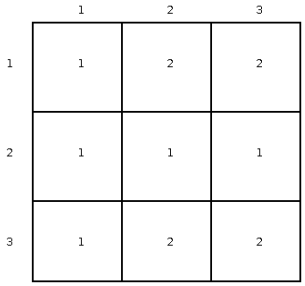
- En se référant au schéma de la proposition 3.1, pour calculer la projection de l'objet sur la surface de projection (le point P), on se proposait de déterminer l'équation de la droite qui passe par l'objet et par la nouvelle vue dans l'univers virtuel (la droite (AB)) et l'équation de la surface de projection. Le point d'intersection de ces deux éléments est le point P. Mais ceci pose un problème d'adaptation : la surface de projection n'est pas forcément un dôme ou un plan. Elle peut être quelconque. On a donc besoin de décrire la surface. Pour cela, on peut employer une équation, mais pour des raisons calculatoires, il est difficile de décrire toutes les surfaces avec une équation dans le cadre d'un *shader*. On peut donc alors description discrète de la surface par l'intermédiaire d'un tableau des profondeurs à deux dimensions. On aurait pu également utiliser des *nurbs* (surface de Bézier) pour une approximation plus juste de la surface, mais cela engendre plus de calcul.
- L'utilisation de tableau dans le style "langage C" est à prohiber dans la programmation de shaders. Or, pour stocker la matrice représentative de la surface de projection, il est nécessaire d'utilisation une structure de tableau. C'est une seconde contrainte pour laquelle la solution basée sur l'utilisation de texture sera vue par la suite.

les solution techniques

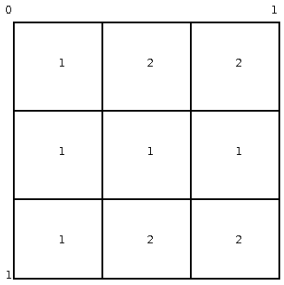
Pour représenter la surface de l'écran, c'est à dire le tableau de profondeurs, il s'agit d'utiliser une texture OpenGL. Cette texture est une matrice x*y et chaque case contient un nombre réel z. Elle représente la surface de projection : écrasée sur la dimension Z, il s'agit d'une image des profondeurs,

qui sont indiquées pour chaques couple de coordonnées (x,y). Cette texture est construite à la main, ou via un dispositif permettant de mesurer les profondeurs (technique de *structured light* par exemple), et est représentative de l'écran réel qui affiche la scène.

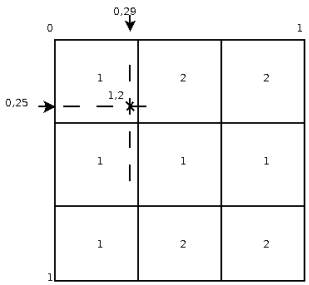
Pour l'implémenter, il faut créer une texture openGL, la remplir avec des nombres réels, la *binder* dans un index et faire passer cet index dans le *shader*. Dans ce dernier, on accède aux éléments de la texture par le principe des "coordonnées de texture", qui fait manipuler la texture comme une texture de dimension 1*1. On peut donc la découper en un nombre arbitraire de parties, sachant que le point aux coordonnées (x,y) de la texture est le résultat de l'interpolation linéaire de la valeur des textels les plus proches. Cette solution offre donc la possibilité de déterminer avec exactitude la projection de l'objet sur la surface de projection car cette surface est définie sur un espace quasi-continu (avec la précision des nombres réels). Par exemple : 4.1



(a) La texture 3x3 est remplie avec des valeurs inscrites dans le corps des réels



(b) Dans le shader, la texture est lissée, on peut accéder à n'importe quelle coordonnée entre 0 et 1



(c) L'accès aux coordonnées (0,29 ; 0,25) est une valeur qui est le résultat de l'interpolation linéaire des valeurs connues, en fonction de la distance à ces valeurs.

FIGURE 4.1 – La gestion de la texture dans le shader

L'algorithme qui calcule la projection du vertex sur la surface consiste à parcourir tout le tableau

des profondeurs et déterminer par où passe la droite reliant le vertex au nouvel angle de vue. Pour cela, on utilise les valeurs voisines afin de faire des encadrements. Il existe une solution plus rapide pour la machine qui consiste à procéder par dichotomie : pour un fragment donné de la surface de projection, on peut savoir dans quelle moitié le rayon passe. Récursivement, on peut alors trouver la plus petite zone par laquelle le rayon passe.

Pour programmer le shader, les calculs conçus ont été implémentés en utilisant la puissance du GLSL : types vecteurs et opérations sur les angles sont les principales utilisations dans ce programme-ci, mais de nombreuses fonctionnalités très puissantes sont utilisables en GLSL. Cet extrait de code montre comment sont déterminées les coordonnées de la projection sur un plan et celles de la nouvelle position du sommet, en utilisant la notation du schéma 3.1.

```
//Détermination de la projection :
coordonnées de la projection
//du vertex par rapport a la camera
secondaire et à un plan (cadre de test)
proj.y=((_3Dvertex.y-A.y)/
(_3Dvertex.z-A.z))*proj.z
+(A.y-((_3Dvertex.y-A.y)/
(_3Dvertex.z-A.z))*A.z);
proj.x=((_3Dvertex.x-A.x)/
(_3Dvertex.z-A.z))*proj.z
+(A.x-((_3Dvertex.x-A.x)/
(_3Dvertex.z-A.z))*A.z);

//angles OP dans chaque plan
des dimensions ecrasees
anglex=acos
(proj.x / sqrt(proj.x*proj.x
+ proj.y*proj.y));
angley=acos
(proj.y / sqrt(proj.y*proj.y
+ proj.z*proj.z));
anglez=acos
(proj.z / sqrt(proj.z*proj.z
+ proj.x*proj.x));

//nouvelles coordonnees
E.x=cos(anglex)*sqrt
((_3Dvertex.x-proj.x)
*(_3Dvertex.x-proj.x)
+ (_3Dvertex.y-proj.y)
*(_3Dvertex.y-proj.y))
+proj.x;
E.y=cos(angley)*sqrt
((_3Dvertex.y-proj.y)
*(_3Dvertex.y-proj.y)
+ (_3Dvertex.z-proj.z)
*(_3Dvertex.z-proj.z))
```

```
+proj.y;
E.z=cos(anglez)*sqrt
((_3Dvertex.z-proj.z)
*(_3Dvertex.z-proj.z)
+ (_3Dvertex.x-proj.x)
*(_3Dvertex.x-proj.x))
+proj.z;
```

Ce fragment de code montre deux choses intéressantes à relever. La première est la similitude très forte entre le code et les formules proposées dans la partie Proposition. La seconde est l'utilisation en direct des attributs x,y,z affectés par défaut aux variables de type vecteur et qui rendent la lecture du calcul très proche de la lecture naturelle des calculs mathématiques.

Enfin, dans ce shader, on passe en paramètre la texture qui est le tableau des profondeurs qui décrit la surface de projection. On profite au passage de l'approximation linéaire utilisée dans les textures pour déterminer le point d'intersection entre la surface de projection et la droite reliant le vertex au nouvel angle de vue.

4.1.2 Expérimentations

Le test pour vérifier la bonne déformation de la scène a été effectué sur un cube placé à différents endroits. Il était nécessaire de réaliser les calculs à la main sur des jeux de tests afin de déterminer à quel endroit de l'écran chaque sommet devait être affiché. Pour cela, une modification a été effectuée au *shader* afin qu'il déplace chaque vertex à l'emplacement de sa projection. Or, comme nous l'avons vu, cette projection peut être approximée par l'imprécision choisie sur la matrice de la surface de projection et l'interpolation (linéaire, et non suivant une courbe de Bézier). Avec la détermination de la projection à l'aide de l'équation du plan de projection, chaque sommet est projeté aux bonnes coordonnées de l'écran (en faisant abstraction du caractère discret de la surface de l'écran)

Le *vertex shader* programmé intervient sur chaque vertex, de façon massivement parallèle. Pour vérifier si le temps réel est bien assuré, le *shader* a été testé sur plusieurs scènes 3D, mobiles ou immobiles, durant des intervalles de 30 secondes et avec le limitateur de FPS fixé à 60. La valeur moyenne du nombre d'images par seconde (FPS) a été reporté pour chaque cas :

- test 1 : un cube
- test 2 : un cube qui tourne sur l'axe Y et qui se déplace en translation sur l'axe Z
- test 3 : cent cubes
- test 4 : cent cubes qui tournent sur l'axe Y et qui se déplacent en translation sur l'axe Z
- par manque de temps, le test 5 n'a pas été testé : une scène complexe incluant des mou-

vements, un grand nombre de polygones et des textures. Une piste indique que ce genre de scène peut être téléchargeable auprès de services de benchmark.

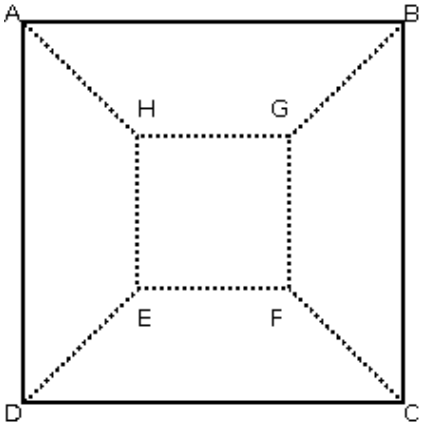
Les résultats ont été reportés dans un tableau :

n° test	moyenne sans le shader	moyenne FPS avec le shader
1	60	60
2	60	60
3	60	60
4	59	59

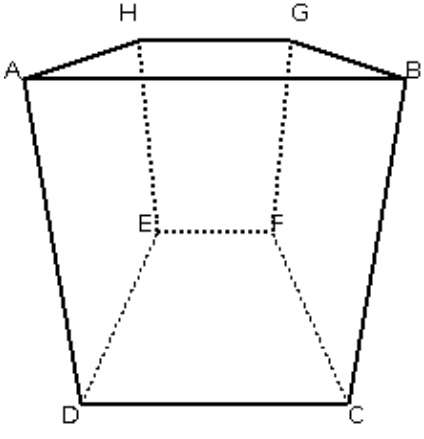
TABLE 4.1 – Tableau des résultats en terme de performance, durant 30 secondes, pour chaque tests.

4.1.3 Résultats

Les tests du bon fonctionnement du rendu ont été concluants. Voici des dessins qui expliquent ce que le rendu donne visuellement et l'explication du phénomène à l'aide d'une vue en deux dimensions 4.2 4.3.

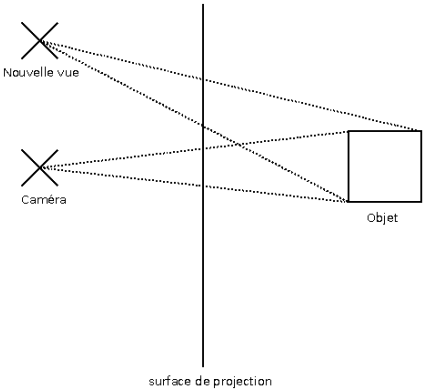


(a) La caméra voit un cube de face. On imagine les arrêtes invisibles en pointillé

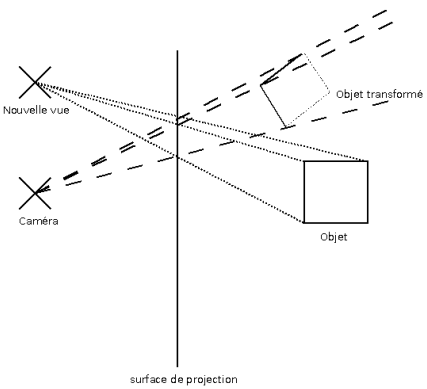


(b) La nouvelle vue doit voir le cube du dessus, on devrait découvrir une nouvelle face. La scène a été transformée pour que la caméra la voit ainsi.

FIGURE 4.2 – Comparaison entre la scène originale et la scène transformée



(a) La caméra voit un cube de face. Les projections imaginaires sont déssinées pour la caméra et pour la nouvelle vue



(b) La scène est transformée pour qu'elle soit projetée telle qu'elle l'aurait été si la nouvelle vue avait été la caméra.

FIGURE 4.3 – Explication en écrasant une dimension

Ce qu'on peut déduire des mesures de FPS, c'est que le *shader*, grace au principe des calculs massivement parallèles, influe de façon transparente sur le rendu de la scène d'un point de vue des performances. Un effet secondaire est le chargement du *shader* qui exige un temps de latence au lancement du programme.

4.2 Conclusion

Le *shader* programmé est fonctionnel pour une caméra fixe et une scène en mouvement car il travaille en temps réel sur chaque vertex et pour chaque *frame*. Il reste néanmoins à le tester avec un benchmark plus complet et à revoir la gestion de la surface représentée par une matrice. Un point en suspens lors du développement reste le choix de la proportionnalité ; c'est la distance entre la surface de projection par rapport à la caméra divisée par la distance entre la caméra et la nouvelle vue. Dans le monde réel, ce quotient est égal à la distance entre le projecteur et l'écran divisée par la distance entre le projecteur et la vue de l'utilisateur. L'implémentation des *geometry shaders* prévue initialement reste également à faire mais pour cela il faut porter le projet sur une nouvelle version d'OpenGL. Le retour d'expérience que l'on peut faire de cette

phase de développement est la difficulté à intégrer des *shaders* à cause de la gestion hasardeuse des versions d'openGL, et la difficulté à manier les shaders au sein d'un programme. Néanmoins, la programmation des *shaders* s'avère moins obscure que les premiers essais peuvent le laisser penser, et il est possible d'aboutir rapidement à des résultats malgré le manque de *logs* d'erreurs. L'amélioration de ce programme peut donc s'effectuer de façon efficace puisqu'il s'agit essentiellement de programmer et reprendre des *shaders*.

5

Conclusion

5.1 Résumé du travail effectué

L'état de l'art nous a révélé plusieurs points importants. D'abord, l'importance de chercher à afficher une scène 3D sur un dôme est capitale, et les écrans de type "plat" ont une limite qui nous force à nous adapter à des déformations qui paraissent trop évidentes quand on s'y concentre. Mais l'affichage sur un dôme présente lui aussi des déformations encore plus importantes du fait que le point de vue de l'utilisateur n'est pas centré sur le dôme. Nous avons donc étudié la recherche effectuée sur l'application d'une image issue d'une sphère sur un plan, à travers les recherches effectuées sur les planisphères. Ceci nous a permis d'aboutir à une première proposition issue de cette recherche mais elle ne résout pas le problème du décentrage du point de vue. Nous avons alors découvert une société récente qui propose un dispositif qui adapte la caméra du monde virtuel à la tête de l'utilisateur dans le monde réel, mais cette proposition ne résout pas le problème de la déformation sur le dôme. De ce fait, nous avons cherché une solution qui allie intelligemment les deux, et qui repose sur le principe de l'anamorphose, ce qui permet de conserver la perspective sans pour autant déformer l'image. La solution issue est basée sur la déformation de la scène 3D de façon à ce qu'elle soit

perçue de façon non déformée et avec les bonnes perpectives pour un point de vue en particulier, indépendant du centre du dôme. La technique est aussi indépendante de la forme du support, on l'a vu à travers la démonstration mathématique du procédé. Pour assurer le temps réel, une solution informatique est mise en avant : la programmation des Shaders, exploitant le GPU.

Le développement de cette solution a posé de nouvelles interrogations liées essentiellement au maniement d'OpenGL, notamment au niveau de la gestion des textures et des matrices de caméra. Mais au delà de ces quelques contraintes générales, les contraintes intrinsèques se sont avérées plus souples et la programmation en elle-même du *shader* s'est avérée être très proche de la conception. Les résultats sont satisfaisants même si l'on sent un manque de *geometry shaders* pour fragmenter les lignes qui peuvent avoir tendance à tromper la nouvelle perspective, et il serait captivant de visualiser la transformation sur une scène complexe et directement sur le dôme.

5.2 Enseignements

J'ai personnellement tiré un enseignement important de ce travail grace à M. Picarougne qui a pris sur son temps pour me donner une description globale du monde de la programmation des shaders et son fonctionnement. Les Vertex Shaders, les Geometry Shaders et les Fragment Shaders ont été cités et leur fonctionnement étudié. Ce travail, dans l'ensemble, offre un enseignement sur une partie du monde du rendu 3D, et particulièrement l'OpenGL. On pourra également retenir un fragment de la recherche effectuée sur la représentation de la perspective avec les quelques techniques étudiées.

5.3 Perspectives de recherche

Une perspective intéressante serait la création d'un dispositif permettant d'enregistrer en temps réel le support sur lequel est affichée la scène 3D. Puisque ce support est enregistré sous forme de matrice, un dispositif enregistrant la distance des éléments par rapport à lui, comme le Kinect par exemple, serait capable d'assurer cette tâche. Nous pourrions alors concevoir une construction de ce type. (cf figure 5.1)

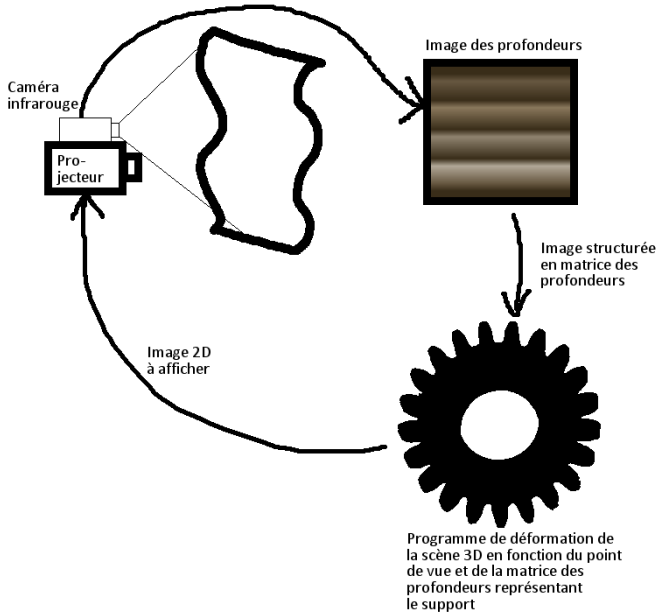


FIGURE 5.1 – Schéma de proposition pour une extension à cette recherche, en se basant sur une caméra infrarouge qui enregistre en temps réel la forme du support.

Autre amélioration : pour représenter la surface de projection, on peut se pencher sur les *nurbs* qui permettent d'approximer une surface et de nous affranchir de la représentation par l'intermédiaire de segments. Non seulement l'approximation pourrait être plus exacte sans cette discrétisation, mais on pourrait utiliser moins de points pour paramétrer la surface.

6

Bibliographie

Bibliographie

[1] Microsoft Corporation *Pipeline Stages (Direct3D 10)*, http://msdn.microsoft.com/en-us/library/bb205123.aspx, - page consultée le 5 Octobre 2011.
Résumé : Le Microsoft Developer Network (MSDN) est une section de Microsoft qui, entre autre, fournit une aide aux développeurs au travers de divers tutoriaux, manuels et documentations. Cette documentation en ligne fournit des explications sur le fonctionnement et l'utilisation du pipeline programmable.
Analyse : Cette documentation apporte des explications théoriques et des définitions. Elle se focalise sur Direct3D mais la théorie des shaders est la même en OpenGL. Il faut cependant faire attention avec certains termes qui ne sont pas les mêmes entre les deux mondes.

[2] Bernie Ashmore *Mapping our world : an innovative approach to mapwork for ages 9-13*,
Résumé : Ce document présente aux écoliers les différentes façons de représenter la Terre ainsi qu'un survol des tenants et aboutissants de la cartographie terrestre.
Analyse : Il s'agit d'une ressource pour enfants, et pourtant elle soulève un problème important : celui de la projection sur un plan et les distorsions engendrées. Le document a le mérite d'établir une liste des différents types de projections, qui sont le résultat de plusieurs siècles de recherches pour certaines. Il permet d'éliminer la proposition de déformation de l'image dans le cadre de ce projet. L'inconvénient est qu'il ne montre pas le problème d'un point de vue mathématique. 16

[3] Robert Rolland, Chercheur associé au laboratoire ERISCS et à L'Institut de Mathématiques de Luminy *Quelques problèmes mathématiques liés à la navigation (version 6)*,
Résumé : Ce rapport présente la problématique d'un point de vue mathématique que pose la navigation maritime
Analyse : Une partie de ce rapport se concentre

sur la représentation de Mercator et démontre mathématiquement les distorsions engendrées. L'inconvénient est qu'il ne présente que cette déformation. 16

[4] Luciano Napolitano *Wideview and other tools for Microsoft Flight Simulator and cockpit builders*, http://www.wideview.it/index.php - page consultée le 12 octobre 2011
Résumé : Ce site fait la promotion d'un Add-on pour Flight Simulator X permettant aux utilisateurs de construire leur propre cockpit à la maison.
Analyse : La ressource est intéressante car elle montre ce que les utilisateurs réussissent à produire afin d'augmenter l'immersion en offrant un champs de vision très large. C'est une alternative à l'affichage sur un dôme et on peut facilement analyser les points négatifs du procédé grace aux nombreuses explications fournies. 12, 69

[5] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner *OpenGL 2.0 Guide officiel 4ème édition*,
Résumé : Ce livre explique au lecteur comment utiliser la librairie graphique OpenGL afin de créer des rendus. La version 2.0 présentée ici inclut la programmation sur le pipeline.
Analyse : La partie qui nous intéresse, celle dédiée au GLSL, a le mérite d'expliquer l'architecture et l'agencement du pipeline graphique afin de mieux comprendre comment les shaders sont traités. Il explique dans le détail comment programmer ses shaders et fournit de nombreux tableaux d'exemples de variables. Le point négatif est l'absence d'un véritable tutoriel pas-à-pas. Il est donc judicieux d'allier cette ressource avec un tutoriel.
Pertinence : Ce livre m'a été conseillé et prêté par M. Picarougne et est suffisamment récent par rapport au sujet qui nous intéresse (les shaders). Il contient également de nombreuses références à d'autres ouvrages et documentations qui permettent d'élargir le champs de recherche. 26

[6] Lighthouse3d.com *GLSL tutorial*, http://www.lighthouse3d.com/opengl/glsl/ - pages consultées régulièrement depuis le 02/10/2011
Résumé : Ce tutoriel en ligne décrit pas à pas comment créer ses shaders avec la librairie OpenGL.
Analyse : Régulièrement, des portions de code sont fournies, voir des programmes totalement fonctionnels. On peut les comprendre grace aux explications du tutoriel, mais il est aussi judicieux de les allier avec les explications théoriques du livre cité précédemment.

Pertinence : L'auteur de ce tutoriel n'est pas connu, mais le sérieux du blog est les nombreux commentaires positifs au sujet de ce tutoriel nous permettent de conserver cette source et de l'exploiter. Comme dit, il vaut tout de même mieux l'allier avec une source plus pertinente afin de vérifier la véracité des explications données.

[7] John Kessenich, Dave Baldwin, Randi Rost *The OpenGL® Shading Language*,
Résumé : Ce document contient les spécifications de la version 4.20 de l'openGL shading language
Analyse : Les spécifications sont complètes, on a également un rappel des fonctions dépréciées et une partie traite des geometry shaders. C'est un document de spécification et n'est pas fait pour apprendre le sujet dont il parle, mais pour être utilisé comme un boîte à outils.
Pertinence : Ce document est recommandé par le livre OpenGL 2.0 Guide officiel 4ème édition, il est très technique et est donc à allier avec d'autres documents du même sujet. 27

[8] *Saturday Magazine, Janvier-Juin 1842*,
Résumé : L'article en page 17 présente le principe de l'anamorphose et ses limites.
Analyse : L'article explique dans le détail comment créer soit-même un dessin soumis à l'anamorphose, et montre un exemple clair du procédé. On regrette cependant l'absence d'explications mathématiques. 21

[9] *ImageMagick v6 Examples – Distorting Images*, http://www.imagemagick.org/Usage/distorts/#affine_projection, - page consultée le 22 Octobre 2011
Résumé : Page de manuel des fonctions imagemagick
Analyse : La partie sur les projection montre d'un point de vue technique comment est opérée la transformation des images lorsqu'on les déforme. On peut alors mieux comprendre comment OpenGL fonctionne pour les affichages. Le document sert surtout pour les personnes désirant faire de la retouche technique d'image, puisque l'utilisation d'OpenGL nous épargne ce genre de calculs.

[10] Andrew S. Glassner *Principles of digital image synthesis, Volume 1*,
Résumé : La partie 1.5 du livre porte sur l'analyse de la perspective et du champ de vision.
Analyse : Cette partie explique avec de nombreux exemples les phénomènes de perspective et de projection des objets que nous percevons. Nous avons surtout un exemple qui justifie l'utilisation d'un dôme pour l'être humain, ainsi que quelques analyses sur les phénomènes d'objets qui apparaissent plus ou moins gros et comment

on peut jouer sur ce phénomène pour faire illusion sur la distance. Cette partie pourrrait également être intéressante dans le cadre d'un projet sur la vision en relief. 11, 13

[11] Thomas Morris *A popular outline of perspective or graphic projection, 1869*,
Résumé : C'est un livre d'architecture du XIXe siècle qui traite de la recherche dans le domaine de la perspective.
Analyse : Le chapitre "pan-angulare perspective" nous fait la présentation de la projection sur un dôme et du véritable phénomène de perspective selon lequel les lignes n'apparaissent pas droites mais courbées. Le diagramme de la projection sur un dôme montre la non-parallélisation des droites du monde perçu. Cette source est à coupler avec une source plus moderne qui pourrait traiter de l'informatisation du phénomène. 14

[12] *The Magazine of science, and schools of art, Volume 1, 1842*,
Résumé : (page 66) Ce court article traite de l'anamorphose en décrivant le phénomène observé et le principe de construction
Analyse : L'article a le mérite de présenter un algorithme qui permet de déterminer la transformation à opérer sur un point de façon à ce que sa projection suive l'illusion donnée par l'anamorphose. D'un point de vue informatique, c'est donc plutôt intéressant. Il faut l'allier avec l"article sur l'anamorphose de Saturday Magazine. 21, 31

7

Planification

Figure 7.1
Figure 7.2

Ce qui est le plus flagrant comme différence entre ce qui avait été prévu et l'effectif, c'est la recherche de documents qui a duré tout le long de cette première phase. Et elle pourrait durer au cours de la seconde phase. La recherche bibliographique est quelque chose de constant qui doit s'effectuer tout au long du projet de recherche car il n'est pas possible de récupérer l'intégralité des documents de l'humanité traitant d'un sujet en particulier, d'autant plus que le sujet portant sur un procédé relativement récent peut voir de nouvelles publications apparaître durant son développement.

Figure 7.3
Figure 7.4

La différence majeure entre ce qui a été prévu et ce qui a été effectif se situe au niveau du temps prévu pour intégrer des shaders. Voyant que l'importation de geometry shaders ne serait pas possible dans l'immédiat, la décision a été prise de ne pas pousser davantage les recherches de ce coté et de se concentrer sur la programmation du vertex shader. On notera aussi quelques incident technique liés à la tempête qui ont empêché temporairement l'accès à la salle du dôme, et qui a obligé quelques modifications sur le planning, pour lequel des marges avaient été prévues. C'est pourquoi on notera sur la

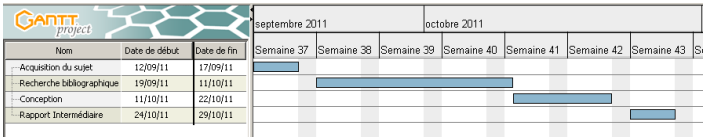


FIGURE 7.1 – Planification prévisionnelle intermédiaire (via Gant Project)

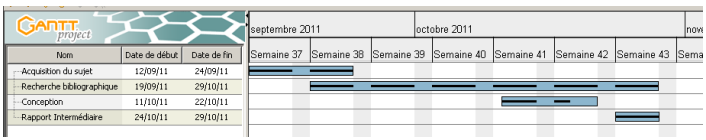


FIGURE 7.2 – Planning effectif intermédiaire (via Gant Project)

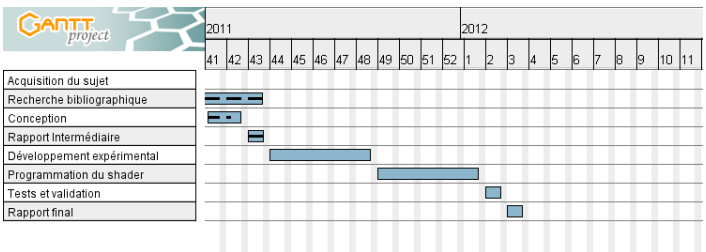


FIGURE 7.3 – Planification prévisionnelle finale (via Gant Project)

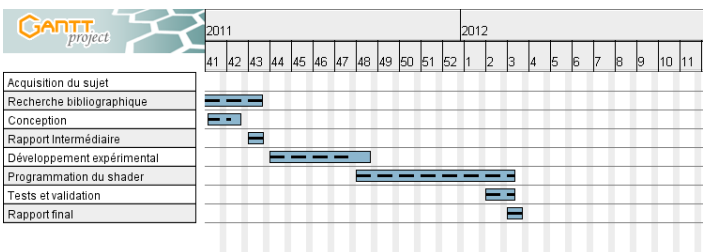


FIGURE 7.4 – Planning effectif final (via Gant Project)

fin quelques activités en parallèles.

8

Fiches de suivi

8.0.1 semaine 38

Temps de travail : 7 heures.

Travail effectué

- Cerner la problématique : quelle transformation faut-il effectuer ?
- Trouver l'application mathématique qui permet de passer des coordonnées d'un vertex aux nouvelles coordonnées. Difficulté : vérifier que l'application mathématique est juste.
- Transformer toute l'application mathématique vers un calcule de vecteurs et de matrices.

Travail non effectué

- Lire la théorie sur les shaders sous QT avec openGL ; (en cours de lecture)

Échange

- Comprendre à quoi est équivalent la transformation.
- Comment est effectué l'affichage sur le dôme
- Choix du geometry shader pour modifier la géomtrie des vertex, et du displacement mapping pour modifier le positionnement des textels.

Planification

- Passer à la rechercher sur la programmation sur GPU
- Finir de lire l'article de developpez.com "http://gbelz.developpez.com/remi-achard/gpu-avance-avec-qt/"
- Tenter un pseudo-code de programmation des shaders à partir de l'application mathématique trouvée.

8.0.2 semaine 39

Temps de travail : 8 heures.

Travail effectué

- Implémentation sur GPU : le tutoriel de NeHe semble le plus approprié : "http://nehe.gamedev.net/" (recommencé par des chercheurs du laboratoire LISA à Bruxelles) -> Vertex shader, geometry shader. Par encore de tutoriel sur le displacement mapping.
- Comprendre les tenants et aboutissants des vertex shaders et geometry shaders grace aux documentations de Nvidia. Le wiki de Valve Software contient de nombreuses références exploitables "http://developer.valvesoftware.com/wiki/Shader" mais le document en lui-même n'est pas suffisamment exploitable.
- Trouvé un ouvrage suffisamment récent sur l'openGL que la BU va recevoir : Computer Graphics Through Opengl : From Theory to Experiments (je vais me renseigner cette semaine si je peux l'avoir à temps, sinon je me contenterai d'ouvrages numérisés)

Travail non effectué

- Tenter un pseudo-code de programmation des shaders à partir de l'application mathématique trouvée. Trop tôt par rapport aux informations que j'ai trouvées.

Échange

- Je tente d'entrer en contact avec un responsable informatique du futuroscope qui pourrait me donner plus d'informations sur la technique d'affichage sur un dôme. Je pourrais espérer avoir une base déjà expérimentée.

Planification

- Déterminer technqiquement comment allier les shaders et mes calculs.
- Essayer de trouver un ouvrage sur l'openGL

suffisamment récent pour inclure le displacement mapping et le geometry shader.

8.0.3 semaine 40

Temps de travail : 8 heures.

Travail effectué

- Déterminer théoriquement comment allier les shaders et mes calculs.
- Trouver un ouvrage sur l'openGL suffisamment récent pour inclure le displacement mapping et le geometry shader. Il m'a été conseillé par M. Picarougne et il traite du GLSL, le langage que je devrai utiliser pour manipuler les shaders.
- J'ai découvert le principe de l'Anamorphose : http://fr.wikipedia.org/wiki/Anamorphose qui s'avère a priori être la technique que je dois automatiser. Ceci pourrait être une aide quant à l'interrogation que j'ai eue avec M. Picarougne sur la proportionnalité de la projection. C'est une piste tout à fait secondaire.

Travail non effectué

- Déterminer techniquement comment allier les shaders et mes calculs.

Échange

- Entrevue avec M. Picarougne pour avoir plus de précisions sur les différents shaders (vertex, geometry et fragment) et leur utilité dans ce projet.
- Même entrevue : présentation de mes calcules et comment les adapter à la discrétisation de la surface du dôme.
- Même entrevue : attribution de mon poste de travail avec le matériel nécéssaire (carte graphique)

Planification

- Creer mes premiers shaders afin de préparer la phase de réalisation
- Terminer d'adapter mes calculs à la structure en matrice.

8.0.4 semaine 41

Temps de travail : 7 heures.

Travail effectué

- Tentative de configuration du compilateur VS2010 64 bits pour intégrer les deux librairies OpenGL (Glew et Glut)

– Nouvelle tentative sur un compilateur Mingw32 : glut ok, glew ok, mais shaders non fonctionnels

Échange

– Entrevue avec M. Picarougne grace à qui j'ai les librairies sur la machine du dôme, avec le compilateur VS2010 32 bits.

Planification

– Creer mes premiers shaders afin de préparer la phase de réalisation

Temps de travail : 4 heures.

Travail effectué

– Revue des inconvénients de ma façon de poser le calcule (équation du dôme) et de la façon efficace de le faire (matrice des profondeurs)
– Lecture des avantages d'OpenGL par rapport à d'autres systèmes de rendu graphique.

8.0.5 semaine 42

Temps de travail : 9 heures.

Travail effectué

– Premiers shaders en GLSL (vertex shaders pour modifier la position des vertex d'une figure)
– (partiellement) conception détaillée

Travail non effectué

– créer moi-même le shader (ne pas se contenter d'utiliser un vertex shader et le parametrer via les variables "uniforme" et "attribute")

Planification

– Finir absolument le rapport et le support de la soutenance durant les vacances et profiter du temps libéré la semaine de la rentrée pour continuer le développement.

8.0.6 semaine 43

Temps de travail : 9 heures.

Travail effectué

– Finir le rapport et le support de la soutenance.
– Elargissement des sources bibliographiques, notamment pour la programmation des shaders.

Planification

– Comme l'avance est prise, je vais pouvoir me concentrer sur le développement cette semaine.

8.0.7 semaine 44

Temps de travail : 9 heures.

Travail effectué

– Comme l'avance est prise, j'ai pu me concentrer sur le développement cette semaine.
– Répétitions pour la soutenance

Planification

– Il faudrait qu'à la fin de la semaine je sois capable de modifier les coordonnées d'un vertex via le vertex shader.

8.0.8 semaine 45

Temps de travail : 9 heures.

Travail effectué

– Etude des deux types de variables dans la programmation des shaders : attribut et uniforme

Travail non effectué

– modifier les coordonnées d'un vertex via le vertex shader. Raison : Il faut que je comprenne comment modifier la valeur des variables, et que je choisisse quel type (attribut, uniforme)

Planification

– Savoir modifier les coordonnées des vertex via shaders.

8.0.9 semaine 46

Temps de travail : 8 heures.

Travail effectué

– Modifier les coordonnées des vertex via shaders.
– Fonctionnel dans le cas d'une primitive.

Planification

– Trouver comment manier les entrées dans un shader program pour calculer les nouvelles coordonnées à donner.

8.0.10 semaine 47

Temps de travail : 8 heures.

Travail effectué

- Implémentation du calcul de détermination des nouvelles coordonnées en utilisant une support plat, et pas encore la matrice représentant le dôme.
- Il reste une opération que je ne peux pas faire sur les vertex : mettre au carré une coordonnée. Or, dans les calculs, il faut pouvoir le faire.

Planification

- Trouver un code source dans lequel des coordonnées de vertex sont changées via un shader afin de voir si ma pratique est bonne.
- Même si je ne peux pas le tester, créer une matrice sommaire représentant le support et l'intégrer aux calculs.

8.0.11 semaine 48

Temps de travail : 11 heures.

Travail effectué

- La projection sur le support est faite
- Le déplacement du vertex à ses nouvelles coordonnées, via le vertex shader, fonctionne avec la scène de test
- Le support pris en compte pour la projection est pour l'instant un plan.

Travail non effectué

- La détermination des nouvelles coordonnées s'effectue en fonction d'un point fixe. La caméra doit donc se trouver à ce point. La raison : je n'ai pas encore trouvé comment importer les coordonnées de la caméra dans le vertex shader, ce qui devrait se faire avec les variables de type "attribute".

Planification

- Fragmenter les primitives en utilisant le geometry shader.
- Tester davantage et améliorer le vertex shader que j'ai programmé.
- à plus long terme : intégrer au calcul une matrice représentative du support.

8.0.12 semaine 49

Temps de travail : 11 heures.

Travail effectué

- Tester davantage et améliorer le vertex shader que j'ai programmé.

– Améliorer le vertex shader, particulièrement pour les coordonnées aux frontières.

Travail non effectué

– Fragmenter les primitives en utilisant le geometry shader, car c'est en cours.

Planification

– Fragmenter les primitives en utilisant le geometry shader. (continuer)
– à plus long terme : intégrer au calcul une matrice représentative du support.

8.0.13 semaine 50

Temps de travail : 10 heures.

Travail effectué

– Fragmenter les primitives en utilisant le geometry shader. (approche)
– Première tentative pour intégrer au calcul une matrice représentative du support. -> implémentation de la structure.

Planification

– Continuer l'implémentation des geometry shaders et de la matrice.
– Construire une scène 3D plus complexe pour tester.

8.0.14 semaine 51

Temps de travail : 6 heures.

Travail effectué

– Construction d'une scène 3D qui permette de tester le fonctionnement (des cubes cachés derrière d'autres cubes et qu'on peut voir grace au shader) -> Faire le test à la rentrée avant de poursuivre sur la matrice et les geometry shaders
– Recherche de codes sources sur Internet d'utilisation des geometry shaders à défaut de tutoriaux. -> Tenter d'implémenter celui que j'ai trouvé à la rentrée.

Planification

– Continuer : Fragmenter les primitives en utilisant le geometry shader. -> à faire de retour à Polytech

– Continuer l'implémentation des geometry shaders et de la matrice. -> à faire de retour à Polytech

8.0.15 semaine 52

Vacances de Noël

8.0.16 semaine 01

Temps de travail : 9 heures.

Travail effectué

– Multiples corrections au shader et tests sur différentes formes.
– Création d'un moyen de visualiser la transformation (caméra tertiaire mobile)
– Implémentation de la matrice. Il reste un problème de gestion de la mémoire qui empêche l'utilisation de la matrice malgré le fait qu'elle soit correctement formée.
– Echange avec M. Picarougne pour faire un premier état des lieux sur ce qui a été fait, ce qui est fait mais ne fonctionne pas, et ce qui n'a pas encore été fait
– Rédaction du rapport d'inovation
– Rédaction partielle du rapport de développement
– Veille supplémentaire (prévue durant la phase de recherche) pour constater de l'avancement de l'Imax, des projections sur dôme, et de l'évolution des cartes graphiques durant ces quatre derniers mois et pour les années à venir.

Planification

– Planification en cours de prévision avec M. Picarougne.
– à défaut : tenter de résoudre le problème avec la matrice, ce qui permettrait de rendre le programme fonctionnel sur tout type de support valide.
– Continer le rapport de développement et le support pour la présentation.

8.0.17 semaine 02

Temps de travail : 9 heures.

Travail effectué

– Prise de RDV avec M. Picarougne pour constater de l'état du projet.
– Tentative de résoudre le problème de la matrice sur le plan théorique, car accès interdit à la salle du dome durant la semaine.
– Rédaction du rapport final (en cours)
– Rédaction du rapport d'innovation (terminé)

Planification

- Terminer le rapport final et préparer la soutenance
- Autre planification à voir durant la réunion.

9

Auto-contrôle et auto-évaluation

				3	2	1	0
Rapport	Organisation	Plan	Equilibre	X			
			Cohérence	X			
		Fluidité	Introductions (partielles)	X			
			Transitions	X			
			Conclusions (partielles)	X			
		Tableaux, figures	Numérotés	X			
			Légendés	X			
			Référencés (non "en ligne")	X			
	Rédaction	Orthographe	Coquilles	X			
			Fautes évitables	X			
			Franglais, jargon				X
		Rédaction	Aisée	X			
			Absence de plagiat !	X			
	Bibliographie	Références	Suffisantes (nombre, intérêt)				X
			Pérennes	X			
			Complètes (auteurs, pages..)				X
			Conséquentes (volume)				X
		Références dans le texte		X			

Proposition de note haute	15,55
Proposition de note basse	15,55

Proposition de note du jury

			3	2	1	0
Projection	Organisation	Plan	X			
		Liaisons	X			
		Numérotation	X			
	Contenu	Informatif	X			
		Concis		X		
		Clair			X	
		Orthographe	X			
		Illustrations	X			
Oral	Présentation	Aisance	X			
		Tenue		X		
		Articulation, compréhension				X
	Durée	Respect	X			
		Temps de parole équilibré				
	Réponses	Pertinence		X		
		Argumentation		X		

Proposition de note haute	15,74
Proposition de note basse	13,2

Proposition de note du jury

				3	2	1	0
Travail	Etude	Bibliographie	Adéquate	X			
			Suffisante				X
		Cahier des charges	Clair	X			
			Formalisé				X
		Hypothèses envisagées	Nombre	X			
			Pertinence	X			
			Analyse *a priori*	X			
		Validation	Tableau comparatif				X
			Choix argumenté(s)	X			
			Faisabilité	X			
	Complexité	Temps consacré		X			
		Résultats obtenus		X			
		Difficulté	Intrinsèque	X			
			Vis-à-vis du binôme				
	Annexes	Fiches d'avancement	Régularité	X			
			Détaillées				X
		Gantt	Prévisionnel et justifications				X
			Effectifs, erreurs, corrections	X			

Proposition note haute	14,11
Proposition note basse	14,11

Proposition de note du jury	

				A	B	C	D
Rapport	Organisation	Plan	Equilibre		x		
			Cohérence	x			
		Fluidité	Introductions (partielles)		x		
			Transitions	x			
			Conclusions (partielles)	x			
		Tableaux, figures	Numérotés	x			
			Légendés	x			
			Référencés (non "en ligne")		x		
	Rédaction	Orthographe	Coquilles	x			
			Fautes évitables	x			
			Franglais, jargon				x
		Rédaction	Aisée	x			
			Absence de plagiat !	x			

Proposition de note haute	16,92
Proposition de note basse	14,61

Proposition de note du jury	

				A	B	C	D
Projection	Organisation	Plan					
		Liaisons					
		Numérotation					
	Contenu	Informatif					
		Concis					
		Clair					
		Orthographe					
		Illustrations					
Oral	Présentation	Aisance					
		Tenue					
		Articulation, compréhension					
	Durée	Respect					
		Temps de parole équilibré					
	Réponses	Pertinence					
		Argumentation					

Proposition de note haute	
Proposition de note basse	

Proposition de note du jury	

				A	B	C	D
Travail	Conception	Générale	Clarté	x			
			Niveau de détail adéquat		x		
			Argumentation	x			
			Validation		x		
		Détaillée	Formalisation adéquate (algorit	x			
			Validation		x		
	Réalisation	Développement	"Fonctionnalités"			x	
			Volume, environnement...		x		
		Tests (unitaires, d'intégration, fonctionnel, de performance...)			x		
	Recette	Livraison du paquetage		x			
		Redéploiement du paquetage		x			
	Complexité	Temps consacré		x			
		Résultats obtenus			x		
		Difficulté	Intrinsèque	x			
			Vis-à-vis du binôme				
	Annexes	Fiches d'avancement	Régularité	x			
			Détaillées	x			
		Gantt	Prévisionnel et justifications				x
			Effectif, erreurs, corrections		x		

Proposition de note haute	15,55
Proposition de note basse	13,33

Proposition de note du jury	

PROPOSITION DE NOTE (II)

NOTE PROJET 67

10

Table des figures

Table des figures

11

Glossaire

– **shader** : un programme exécuté sur une carte graphique qui influe sur le rendu de ce sur quoi il opère (Vertex, primitive, fragment)
– **vertex** : le point où des lignes se rencontrent. En géométrie, il s'agit d'un sommet.
– **primitive** : une forme géométrique construite à partir de vertex.
– **fragment** : plus petit constituant de l'image projetée dans la synthèse d'image. Il faut noter qu'un pixel de l'écran n'est pas forcément égal à un fragment.