République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique

Université Mouloud MAMMERI de Tizi-Ouzou
Faculté de Génie électrique et Informatique
Département Informatique

Mémoire

De fin d'études

En vue de l'obtention du diplôme
De Master en informatique

Thème

Modélisation d'une scène 3 D

Proposé et dirigé par :

M^me^ CHAMEK Linda

Réalisé par :

M^lle^ HAMDAD Amel

2012/2013

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique

Université Mouloud MAMMERI de Tizi-Ouzou
Faculté de Génie électrique et Informatique
Département Informatique

Mémoire

De fin d'études

En vue de l'obtention du diplôme
De Master en informatique

Thème

Modélisation d'une scène 3 D

Proposé et dirigé par :

M^me^ CHAMEK Linda

Réalisé par :

M^lle^ HAMDAD Amel

2012/2013

Remerciements

D’abord je remercie le bon dieu de m’avoir donné santé, courage, volonté et foi pour réaliser ce travail.

Je tiens à exprimer ma profonde gratitude à ma promotrice Madame CHAMEK Linda pour tout ce qu’elle m’a apporté comme aide, connaissances et conseils pour l’accomplissement de ce travail.

Je remercie vivement les membres du jury pour avoir accepté d’évaluer mon travail.

Aux enseignants de département d’informatique de Mouloud MAMMERI pour l’effort qu’ils ont déployé afin d’assurer ma Formation, pour leurs compétences, et surtout leur modestie.

Je remercie mon ami Rabah qui m’a soutenu tout au long de mon projet .

Dédicaces

Je dédie ce modeste travail :

En premier lieu à mes très chers parents qui n'ont jamais cessé d'être à mes cotés par leurs précieux conseils et orientation ainsi que leur soutien moral.

A la mémoire de mes grands parents.

A mon oncle Zizi Moh Cherif .

A mes sœurs et frères .

A mes beaux frères .

A mes neveux et nièces.

A tous mes oncles, tantes et cousin(e) s sans exception .

A tous mes amis en particulier

(Nadia,Meriem,Celia,Nabila,Rabah,Takfa,Dahmane,

Walid,Sidali,nounou,Djamel,Ghiles,Yacine,Nacim,Hocine)

A tous ceux que j'aime .

Toutes personnes qui m'ont soutenue tout au long de mes études.

A.HAMDAD

Table des matières

Liste des Figures

Introduction générale :

Aujourd'hui, lorsqu'on parle de bibliothèque de développement 3D, les termes qui reviennent le plus souvent sont Direct3D et OpenGL.

Direct3D est une API (Application Programming Interface) développée par Microsoft pour son OS Windows. Propriétaire et non portable, Direct3D fait partie de l'unité DirectX et est entièrement dirigé vers les 'Jeux vidéo'. Son avantage essentiel est d'être bien supporté par les fabricants de cartes graphiques. Pour sa part, OpenGL (Open Graphic Library, ce qui se traduit en français par 'Bibliothèque Graphique Ouverte') est une API développée depuis 1992 par Silicon Graphics(http://www.sgi.com), un des grands leader de l'informatique graphique professionnelle.

Particulièrement développé pour les stations graphiques haut de gamme Silicon Graphics, l'aspect ouvert d'OpenGL lui a valu d'être porté sur la majorité des plateformes modernes, et sur la plupart des systèmes d'exploitation. Aujourd'hui, OpenGL est reconnu par les professionnels comme un standard, et est utilisé par tous les logiciels professionnels de synthèse d'images (3DSMax, Maya, Softimage...), de CAO (ProEngineer, Catia...), mais également dans le domaine des jeux vidéos 3D (Quake (1,2,3), Unreal...).

OpenGL est orienté vers l'interaction 'temps réel', c'est à dire qu'un programme engendré avec OpenGL permet en général à l'utilisateur d'interagir par l'intermédiaire de périphériques d'entrée (clavier, souris, trackball...) avec la scène (déplacement de caméra, modification de focale, déplacement des objets....), et d'en voir aussitôt l'effet. En conséquence, pour que le jeu vidéo soit 'fluide', c'est-à-dire que les déplacements du joueur ne paraissent pas trop irréguliers, le programme doit être capable de calculer et d'afficher des images en moins d'un dixième de secondes. Dans le cas d'un jeu vidéo on peut observer que si l'image n'est pas recalculée une quarantaine de fois par secondes, le jeu ne sera pas fluide.

OpenGL est donc une librairie graphique, c’est à dire une interface logiciel permettant d’effectuer des opérations d'affichage sur un écran graphique. Cette interface est constituée d’environ 150 fonctions graphiques, de l'affichage de points et segments

jusqu'au plaquage de textures sur des objets tridimensionnels. OpenGL présente l'intérêt majeur d'être indépendante du matériel employé, donc de l'architecture sur laquelle l'application graphique est développée. De fait, OpenGL n'a pas de fonctions de vérification du matériel, en sortie ou en entrée. Le développeur doit, soit utiliser directement les fonctions du système sur lequel il développe son application (Xwindow par exemple pour Unix), soit utiliser une interface logiciel située en amont d'OpenGL et qui propose des fonctions pour cela (Glut par exemple). De manière équivalente, OpenGL ne propose pas de fonctions graphiques de haut niveau utilisant des primitives géométriques complexes (sphères, splines, etc.). La réalisation des ces fonctions est laissée au développeur ou à l'interface logiciel en amont d'OpenGL. Cette spécification très ciblée de la librairie lui assure une portabilité importante et a fait d'OpenGL le standard graphique actuel.

La première question qui vient évidemment à l'esprit est la suivante : Que peut-on faire avec OpenGL ? La réponse est la suivante : OpenGL permet de représenter à l'écran des scènes tridimensionnelles réalistes. Le terme 'réaliste' est ici tout a fait relatif : on peut considérer comme réaliste une image de synthèse ou tous les objets représentés sont correctement placés au niveau de la perspective sans que les couleurs correspondent à ce qu'on observe dans la nature. Le réalisme ultime peut être atteint lorsqu'il est impossible à première vue de savoir si une image est issue d'un calcul ou si elle simplement une photo numérisée (on parle alors de photo-réalisme). Au risque d'en décevoir certains, OpenGL ne permet pas de générer des images photo-réalistes.

Afin de rendre l'aspect des objets et images plus attrayant,et d'obtenir des rendus graphiques vraisemblables OpenGl met à notre disposition deux moyens pour simuler une scène graphique réaliste : l'éclairage et la texture.

I. Introduction

Nouvelle, innovante, révolutionnaire, c'est une discipline qui a changé la face de l'informatique. Vous l'avez deviné, il s'agit bien de l'infographie 3D.

Ce nouveau support visuel est aujourd'hui un outil moderne employé dans un très grand nombre de domaines, l'infographie 3D a envahi les médias, notamment la télévision et le cinéma, mais son usage ne se limite pas à ces domaines. En effet, son influence s'étend à d'innombrables applications, allant de l'ingénierie à l'éducation, gagnant de plus en plus d'importance, et devenant même parfois indispensable. Toutefois, peu de personnes se rendent compte de l'ampleur de ce phénomène , pour beaucoup, l'infographie 3D est associée à "des dessins animés".

I.1 L'infographie

L'informatique graphique est née il y a près de trente-cinq ans maintenant et elle s'est depuis considérablement développée, pénétrant de nombreux domaines industriels comme l'architecture et l'urbanisme, l'audiovisuel, la CAO, la chimie, la médecine, le multimédia, la simulation, la visualisation scientifique.

L'œil reçoit une projection 2D (une photographie) de l'environnement dans lequel nous évoluons, et transmet cette image à notre cerveau. A partir de cette représentation partielle, le cerveau reconstruit l'environnement 3D en compensant par de nombreux « à priori » la perte d'information induite par cette projection (distance aux objets, partie cachée, …).
L'infographie consiste, par calcul, à construire une scène virtuelle (généralement 3D, mais parfois 2D) et à reproduire cette projection pour l'afficher sous forme d'une matrice rectangulaire de pixels: l'écran de l'ordinateur. L'œil capte alors cette image 2D et nous effectuons une reconstruction mentale plus ou moins réussie de la scène en fonction du degré de réalisme et du niveau des informations produites.

I.2 OpenGL

OpenGL est une librairie graphique, c'est à dire une interface logiciel permettant d'effectuer des opérations d'affichage sur un écran graphique. Cette interface, développée par Silicon Graphics, est constituée d'environ 150 fonctions graphiques, de l'affichage de points et segments jusqu'au plaquage de textures sur des objets tridimensionnels.

OpenGL présente l'intérêt majeur d'être indépendante du matériel utilisé, donc de l'architecture sur laquelle l'application graphique est développée. De fait, OpenGL n'a pas de fonctions de contrôle du matériel, en sortie ou en entrée. Le développeur doit, soit utiliser directement les fonctions du système sur lequel il développe son application (Xwindow par exemple pour Unix), soit utiliser une interface logiciel située en amont d'OpenGL et qui propose des fonctions pour cela (Glut par exemple).

De manière équivalente, OpenGL ne propose pas de fonctions graphiques de haut niveau utilisant des primitives géométriques complexes (sphères, splines, etc.). La réalisation des ces fonctions est laissée au développeur ou à l'interface logiciel en amont d'OpenGL. Cette spécification très ciblée de la librairie lui assure une portabilité importante et a fait d'OpenGL le standard graphique actuel.

En résumé OpenGL est une librairie graphique de bas niveau utilisant les primitives géométriques de base : points, segments et polygones. L'interfaçage avec le matériel (carte graphique, mémoire vidéo, etc.) ou avec des commandes de haut niveau (modèles 3D) doit être ajouté pour construire une application graphique.

II. Qu'est-ce que l'Informatique Graphique

L'informatique graphique est née il y a près de trente-cinq ans maintenant et elle s'est depuis considérablement développée, pénétrant de nombreux domaines industriels comme l'architecture et l'urbanisme, l'audiovisuel, la CAO, la chimie, la médecine, le multimédia, la simulation, la visualisation scientifique. Dans cet ouvrage, les auteurs décrivent les méthodes et les algorithmes utilisés pour produire des images de synthèse. Outre des techniques de base comme des éléments de mathématiques et de théorie du signal, les courbes et les surfaces et les algorithmes et structures de données bidimensionnels.

II.1 Domaine d'application de l'informatique graphique

•Art, loisirs, éditions :

- Production de films, animations, effets spéciaux
- Jeux sur ordinateurs
- Sites web
- Conception de présentations, de livres, de magazines

• Traitement de l'image et graphisme sur ordinateur

• Suivi de processus

• Environnement de simulation (vol, réalité virtuelle,…)

• Conception assistée par ordinateur :

- Architecture
- Circuits électroniques
- Pièces mécaniques
- Aéronautique, automobile (fuselages, turbulences,…)

• La discrétisation :

Les algorithmes et les matériels utilisés en informatique graphique donnent une représentation discrète d'un modèle géométrique exact (s'il est conceptuel) ou presqu'infiniment détaillable (s'il est réel)

II.2 Synthèse d'images

1. Modélisation

Structures de données pour la représentation de scènes 2D ou 3D

2. Rendu

Construction d'images 2D à partir de modèles 2D ou 3D

3. Animation

Simulation de changements au cours du temps

4. Interaction

Domaine d'application de l'informatique graphique

III. Quelques repères historiques

En 1960, le concepteur William Fetter essayait de concevoir un nouveau processus afin de maximiser l'efficacité de la disposition de l'intérieur de l'habitacle des Boeings. Son produit final fut une vue orthographique informatisée de la forme humaine.

Fetter conçut le terme "Computer Graphics" pour décrire sa création, donnant naissance à une série d'événements qui révolutionnera par la suite le monde du divertissement, de la publicité et des médias.

Un des contemporains de Fetter, Ivan Sutherland, mit la machine en marche lorsqu'il présenta son doctorat (PhD), une thèse intitulée "Sketchpad: AMan-machine Graphical communications System" (Sketchpad: une interface de communication Homme-Machine). Le logiciel permettait pour la toute première fois à une personne de créer interactivement une image sur un écran d'ordinateur.

Selon Sun Microsystems, dont Sutherland est actuellement vice-président, Sketchpad fut un pionnier des concepts de calculs graphiques en incluant les structures de la mémoire permettants de stocker les objets, d'étirer des lignes, de pouvoir zoomer sur l'affichage et faire des lignes, des coins, et des joints parfaits. C'était le premier GUI (interface utilisateur graphique) bien avant que le terme soit inventé.

Au moment de recevoir son Doctorat, Sutherland, qui était alors l'un des plus grands innovateurs en informatique, fut appelé sous les drapeaux de l'armée américaine. Plus à l'aise dans le domaine de l'infographie, Sutherland resta à l'Université de l'Utah où il aida à transformer le système informatique en institut de recherches qui, aujourd'hui encore, a une grande influence sur l'industrie graphique.

Les toutes premières images en 3D étaient extrêmement rudimentaires comparées aux standards et consistaient en une représentation de différentes figures géométriques. D'après 3DVF, c'était acceptable, bien que l'on pût voir ce qu'il y avait devant et derrière l'image. Ceci amena les collègues de Sutherland, Evans, Wylie, Romney, et Erdahl à développer l'algorithme Scan line HSR (Hidden Surface Removal) pour créer des rendus d'objets solides. Beaucoup d'algorithmes de HSR furent présentés au cours des années incluant le "back-face detection", le "depth sorting", le "ray casting", le "Z-Buffer" et l' "area subdivision". Plus tard, Sutherland et ses compères publièrent un article intitulé "Caractérisation de 10 algorithmes en

Hidden Surface", ce qui couvrait les algorithmes déjà connus à l'époque. D'ailleurs, ceci sera la dernière contribution directe de Sutherland à la recherche en infographie. Aujourd'hui, on utilise encore ces divers algorithmes scan line.

Shading de Gouraud et Shading de Phong

Sur le chemin du réalisme, la prochaine étape pour les développeurs était d'augmenter l'apparente complexité d'une scène sans en augmenter le nombre de faces, par conséquent de conserver la précieuse mémoire système. Dans les systèmes de rendu précédents, la seule manière d'augmenter l'apparente complexité d'un modèle était d'ajouter davantage de polygones. On perdait cet effet lissé si la caméra se rapprochait du modèle, compte tenu du fait que dans les premiers moteurs de rendu, le seul modèle d'ombrage était le ''flat shading'', appelé aussi ''faceté''. Ce modèle d'ombrage trouva le vecteur normal en relation avec une face et utilisa cette information pour ombrer tous les pixels.

Tout ceci changea quand Henry Gouraud développa son célèbre et largement utilisé modèle de shading Gouraud. Le professeur Malgouyres nous explique que, dans le cas du modèle de Gouraud, on calcule l'intensité renvoyée par la surface aux sommets du polyèdre, puis, pour calculer une intensité renvoyée en un point quelconque d'une facette du polyèdre, on interpole les valeurs précédemment calculées aux sommets de la facette.

Un chercheur du nom de Phong Bui-Tuong étendit ses recherches sur le modèle de shading d'Henry Gouraud en franchissant l'étape suivante. Au lieu d'interpoler l'intensité renvoyée par les sommets du polyèdre, l'idée du modèle de phong, selon le Professeur Malgouyres, est d'interpoler le vecteur normal à la surface en ces sommets. En interpolant sur toute la surface basée sur des normales, on obtient une surface extrêmement lisse avec des rehauts précis. Le principal inconvénient, selon Dmitry Shklyar, est que le Phong

est connu pour sa lenteur. Si l'on compare le modèle de Phong avec celui de Gouraud sur deux modèles identiques, on verra qu'il faudra jusqu'à huit fois plus de temps au modèle de Phong pour rendre que celui de Gouraud.

Bump Mapping

Au fur et à mesure que les recherches de nouvelles méthodes de shading avançaient, Jim Blinn découvrit qu'en modifiant les normales de surface, on pouvait simuler l'apparition d'un nombre de faces plus dense. Cette technique devint célèbre sous le nom de « bump mapping » ou « cartes de déformation » et est aujourd'hui toujours très utilisée dans des applications allant des jeux temps réel aux longs-métrages.

L'avantage du bump mapping, selon le Professeur Malgouyres, est de permettre de représenter une surface irrégulière sans avoir besoin de représenter ces irrégularités au niveau de la modélisation géométrique, ce qui serait beaucoup plus coûteux. Ainsi, au lieu de représenter une surface complexe, on applique une perturbation du vecteur normal avant de calculer l'illumination d'un point, donnant l'illusion d'une déformation de surface.

3DVF note que récemment, l'idée de modifier les normales de surface au niveau du pixel a davantage été étudiée. La technique de mapping de normales a été présentée afin de pallier au manque de bande passante du matériel graphique actuel, qui n'était pas assez rapide pour traiter les modèles à grande échelle pour une « qualité cinéma ».

Une solution a été trouvée en créant une structure de données pouvant capturer l'information de la normale d'un modèle ultra détaillé et ensuite l'appliquer à une version en résolution réduite. L'effet est semblable à celui du bump mapping, dans le sens où la géométrie extérieure n'est pas modifiée, pourtant, on atteint l'illusion d'un modèle complexe. Les futurs jeux comme Doom3 de ID Software, Half Life 2 de Valve Software ou Halo 2 de Bungie font appel à cette technique afin de produire en temps réel des images à couper le souffle.

Les Systèmes d'illumination

Depuis les années 60, le but était d'obtenir des rendus comparables à des photographies, une sorte de "Graal" de la 3D. Au cours des années, on a dû surmonter bien des difficultés pour atteindre le photoréalisme, ce qui nous conduit à l'une des solutions les plus prisées d'aujourd'hui : l'Illumination Globale (Global Illumination, ou GI). La GI regroupe toutes les techniques de simulation de lumière dispersée dans un environnement artificiel.

Illumination Globale

A vrai dire, la GI n'est pas un algorithme, mais un effet produit par plusieurs algorithmes. Le premier algorithme conçu pour traiter ce problème fut le Ray Tracing (lancé de rayon). Le ray tracing fut initialement conçu comme méthode pour calculer la GI. La toute première mise en oeuvre fut exécutée par IBM, en particulier un chercheur, qui publia un papier intitulé «Some Techniques for Shading Machine Renderings of Solids ». Tout ceci se passait en 1968 et aucun ordinateur de l'époque n'était capable d'utiliser pratiquement cet algorithme.

Cela ne fut pas envisageable avant 1980, date où Turner Whitted publia un article édifiant intitulé « An Improved Illumination Model for Shaded Display ». L'article de Whitted décrivait un algorithme de ray tracing prévu pour rendre des ombres, des réflexions et des réfractions. La méthode de Whitted était tellement simple qu'elle pouvait être décrite en seulement trois fonctions, comprenant à peine une douzaine de lignes de prétendu code. C'est la raison pour laquelle, selon 3DVF, la plupart des solutions de rendu du commerce sont basées sur les recherches de Turner Whitted.

Le Ray Tracing

D'après le professeur Rémy Malgouyres, à chaque pixel d'une image est associée une seule couleur, qui est idéalement une moyenne des couleurs de tous les rayons lumineux qui traversent le pixel et dirigés vers le centre de projection, qui est le lieu de l'observateur.

Selon lui, le premier problème du lancer de rayon est de générer tous les rayons qui traversent les pixels de l'image vers le centre de projection. En physique, la lumière part des sources lumineuses pour aboutir, après diverses réfractions et réflexions, à l'observateur. Beaucoup n'aboutissent jamais à l'observateur. Pour éviter de traiter tous ces rayons inutiles, dans le lancer de rayon, on suit les rayons à l'envers, à partir de l'observateur vers les sources lumineuses.

L'Echantillonnage Stochastique

Robert L.Cook, qui a contribué à la création du très populaire Renderman Standard, a cherché à résoudre ce problème et a publié ses résultats dans un article de 1986 intitulé « Stochastic Sampling in Computer Graphics ». La méthode d'échantillonnage stochastique de Cook fonctionne en échantillonnant de façon aléatoire les zones entre les pixels. Avec cette méthode, l'effet

extrêmement discordant du crénelage est réduit à un bruit aisément toléré par l'oeil humain.

Ce bruit, outre le fait qu'il peut remplacer le crénelage, peut aussi être utilisé avec succès pour simuler de nombreux phénomènes « fuzzy ». Par exemple, l'échantillonnage stochastique peut être utilisé pour simuler avec précision une réflexion floue en appliquant une perturbation sur les rayons secondaires.

Pour 3DVF, beaucoup de moteurs de rendu basés sur le scan-line, comme le RMan de Pixar, ont entièrement tiré parti des travaux de Cook. L'échantillonnage stochastique peut être utilisé pour traiter ces phénomènes où le moteur de rendu calcule ce qu'on appelle le « cercle de confusion », et donc perturber les rayons appropriés qui se rapportent au cercle afin de créer un effet de flou.

Malgré l'addition d'un nouveau répertoire tout entier d'effets, l'échantillonnage stochastique n'arrange pas les plus gros inconvénients en solution pour la GI. Le ray tracing résout en partie le problème, dans le sens où il simule seulement l'illumination directe. Une méthode pour simuler les réflexions diffuses était nécessaire pour compléter la GI, à savoir la radiosité.

La Radiosité

Avec leur article de 1984 « Modeling the Interaction of Light Between Diffuse Surfaces », Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg et Bennett Battaile présentèrent le second algorithme de la décennie qui révolutionna le monde de l'infographie. D'après des théories de physique, la lumière se comporte comme une particule et comme une onde, on parle ainsi de dualité. Et bien la radiosité gère la lumière dans une scène 3D suivant les propriétés ondulatoires de la lumière. Cependant la radiosité, comme le ray tracing, ne résolvait qu'une partie de l'équation de la GI, et ne prenait pas en compte les effets tels que les réflexions et les transmissions spéculaires.

Puisque le ray tracing avait réussi à gérer ces effets avec succès, l'étape suivante la plus logique pour les chercheurs était de combiner les deux algorithmes et d'y reporter ainsi tous les avantages de l'échantillonnage stochastique. Le résultat final donnait une méthode vraiment solide pour la création de scènes photoréalistes.

La radiosité n'est cependant pas dépourvue d'inconvénients. Le simple fait qu'elle doive subdiviser la scène en sous-éléments exige d'énormes ressources du système. Le couple

radiosité/faces crée de même d'autres problèmes, par exemple, des artefacts visibles, le calcul de changements brusques dans l'illumination et d'ombres trop vagues.

Le Photon Mapping

Au milieu des années 90, un chercheur du nom de Henrik Wann Jensen créa une parade à ces inconvénients. Durant ses études en PhD en 1994, à l'Université Technique du Danemark, Jensen inventa l'algorithme connu sous le nom de photon mapping. Ses résultats furent officiellement publiés en 1996 dans son article « Global Illumination Using Photon Maps ».

L'algorithme de Jensen se déroule en deux passes. La première émet un nombre défini par l'utilisateur de photons dans la scène. Ceux-ci rebondissent pendant une durée déterminée, ou quittent complètement la scène. Au fur et à mesure, chaque rebond du photon perd en intensité, et prend au passage les couleurs d'autres surfaces. Ces rebonds sont ensuite mémorisés dans une structure de données appelée le photon map.

La seconde passe sample basiquement chacun des pixels de la scène, comme un ray tracer, et calcule combien chaque percussion du photon contribuerait à la couleur et à l'illumination de ce pixel. Une fois de plus, tous les avantages du ray tracing peuvent être mis en œuvre, comme l'échantillonnage stochastique.

En éliminant virtuellement chaque inconvénient de l'utilisation d'une radiosité hybride et d'une solution de ray tracing, le photon mapping apporte quelques surprises bienvenues en plus. En raison de la nature des particules du photon mapping, celui-ci devient une méthode extrêmement compétente pour la simulation de nouveaux effets.
Ceux-ci incluent les caustiques, où les photons sont concentrés dans un volume et sont retransmis au travers, et le sub surface scattering qui prend en compte l'exacte modélisation de surfaces translucides, comme la cire de bougie, la peau humaine etc.

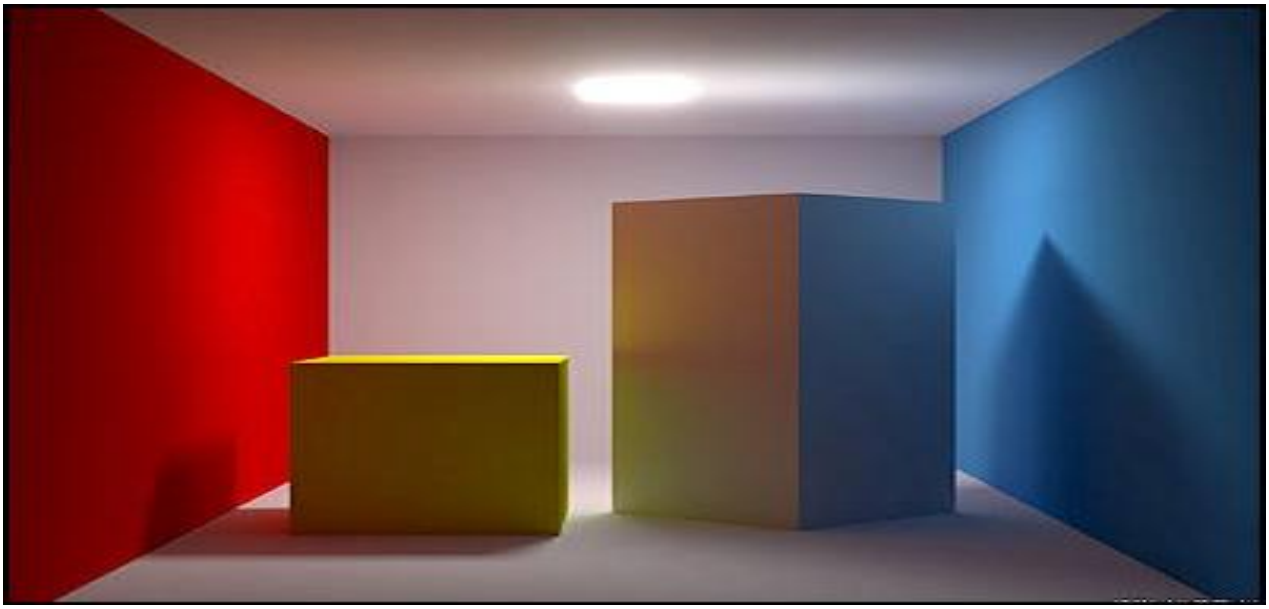


Figure 1 : Système d'illumination temps réel

Ces dernières années, l'industrie du jeu vidéo a été l'un des principaux moteurs de la technologie informatique. La conception multimédia et les simulations scientifiques occupent une partie relativement modérée du marché comparé au jeu vidéo, mais cela ne signifie pas qu'ils ne bénéficient pas de ces avancées.

Les deux acteurs de cette relation sont indirectement interdépendants, étant donné que les avancées des techniques graphiques s'immiscent dans le réseau de production du jeu. Récemment, les techniques comme la radiosité et le photon mapping ont été utilisées pour augmenter le photoréalisme des environnements de jeux-vidéo.

Il y a quelques années, le matériel destiné au graphisme a tout particulièrement excellé dans l'affichage des polygones et leurs textures . Les quelques dernières innovations de Nvidia et ATI ont apporté quelque chose de complètement différent sur le marché; il est désormais possible d'exécuter des shaders personnalisés avec du matériel graphique.

La prochaine étape dans l'entrée des algorithmes de GI dans le royaume de l'interaction temps réel est d'exécuter le photon map de Jensen.

Based lighting

L'une des exigences pour obtenir un photoréalisme dans une scène est l'apparition de tous les objets à l'intérieur d'une scène, exerçant une influence les uns sur les autres. Les réflexions, les réfractions ainsi que les ombres peuvent être simulées facilement et de manière convaincante, même en temps réel, bien que les réflexions diffuses présentent un problème. Une nouvelle technique appelée based lighting tente d'y remédier. Au lieu d'utiliser beaucoup de calculs douteux, l'image based lighting essaye de mémoriser l'information de l'éclairage dans un fichier image. Cette image haute précision n'enregistre pas seulement l'information de la couleur à chacun des pixels, mais aussi une valeur de radiance. Celle-ci est ensuite utilisée pour tester la façon qu'a chaque pixel de contribuer à la diffusion de la lumière d'une scène.

IV. Les quatre éléments de l'IG

- **Polylignes** : droites, cercles, courbes, plus ou moins épaisses, continues ou pointillées, …
- **Texte** : police, taille, style, …
- **Régions remplies** : généralement limitées par une polyligne
- **Images digitalisées** : sur une *pixmap*

V. Architectures logicielles et matérielles pour la visualisation

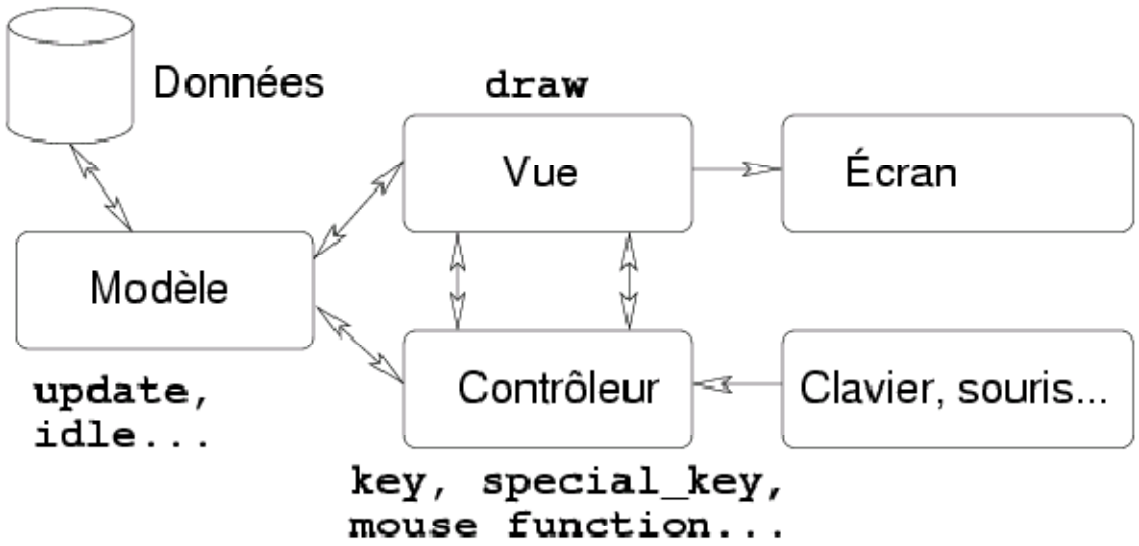


Figure 2 : Architecture générale d'un système graphique interactif

V.1 Le pipeline graphique

V.1.1. *Introduction* :

Une image analogique diffère d'une image numérique en ce que cette dernière code les informations concernées sous forme numérique et non sous forme continue.

Pour l'image numérique, on peut distinguer l'image-vecteurs (où l'image est constituée d'un ensemble de segments joignant des points de l'écran) de l'image pixels (où l'image est constituée d'un tableau de pixels, chaque pixel étant la quantité élémentaire d'information codée sous forme de n bits).

Le domaine de l'image numérique s'est beaucoup développé depuis quelques dizaines d'années. Il peut être représenté par le schéma de la figure 9 qui fait apparaître deux niveaux distincts : un niveau physique, où une image est un signal bidimensionnel, et un niveau conceptuel qui comprend les structures et/ou les modèles qui permettront de construire les images du niveau physique.

La synthèse d'images est donc un domaine de l'image numérique, celui qui. À partir de structures et/ou de modèles, permet d'obtenir des signaux numériques pouvant être visualisés sur un écran graphique par exemple. Dans ce premier chapitre, nous allons décrire le processus permettant de générer une image à partir de modèles. En fait, ce processus peut être représenté par la figure 10, le logiciel graphique jouant un rôle d'intermédiaire entre l'opérateur humain et le programme d'application. Il s'agit d'un pipeline dans lequel on construit d'abord un modèle objet avant de visualiser celui-ci.

Pour préciser la terminologie utilisée ici (et qui n'est certainement pas standard), on peut penser à l'exemple de la conception d'une voiture. Le programme d'application est un logiciel de CAO capable, par exemple, d'effectuer des calculs de structure sur les éléments rentrés dans la mémoire de l'ordinateur. Le logiciel graphique peut permettre, lui, de visualiser la carosserie de la voiture en fonction de la peinture choisie et dans des conditions d'éclairement données.

La séparation entre le programme d'application et le logiciel graphique n'est pas toujours très nette. Le logiciel graphique peut se réduire à un simple ensemble matériel géré par l'application ou, au contraire, proposer une interface de plus haut niveau et un ensemble d'outils spécifiques.

Nous allons donc successivement étudier les paramètres, appelés attributs graphiques, qui permettent de constituer le modèle objet dans le paragraphe V.1.2 puis, dans le paragraphe V.1.3 , les attributs de visualisation qui régissent le processus d'affichage. Nous conclurons cette partie en présentant les deux grandes approches utilisées pour la réalisation de logiciels graphiques : l'approche spécialisée puis l'approche générale.

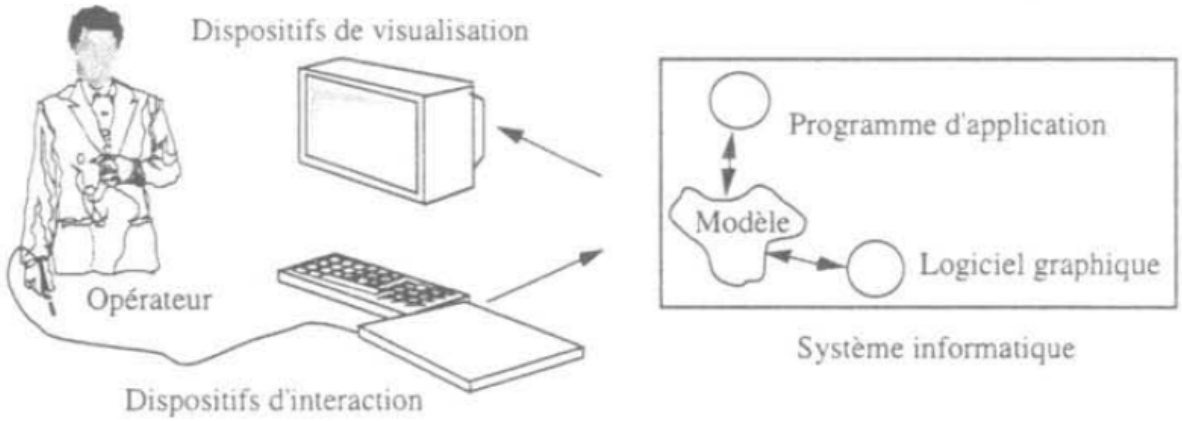


Figure 3 : Le processus de visualisation scientifique

V.1.2 *Les attributs graphiques*

En informatique graphique, le programme d'application contient en général un modèle des entités manipulées, sous forme de structures de données ou de procédures.

Parmi ces données, certaines sont propres à l'application (la masse, la résistance des matériaux, par exemple). D'autres sont liées à la représentation graphique. Ce sont celles qui vont nous intéresser ; nous les appellerons les attributs graphiques . Dans ses travaux il y a cinq attributs comme suit :

- la morphologie, qui permet de définir la forme intrinsèque des objets (par exemple, un cube, une sphère, une surface gauche, etc.) ;
- la géométrie, qui permet de positionner les objets les uns par rapport aux autres ;
- l'aspect, qui permet de définir l'apparence d'un objet (sa couleur, sa texture, sa brillance, son mode de réflexion de la lumière, etc.).

Ces trois attributs sont purement graphiques ; les deux qui suivent sont surtout utilisés pour la gestion des données :

- l'identité, qui permet la nomination, la désignation d'objets ou de groupes d'objets .
- la structure, qui exprime les relations liant certains objets (l'appartenance, l'inclusion, le contact, la proximité, etc.).

Ces cinq attributs graphiques permettent de définir les éléments pouvant conduire à un modèle des objets. L'élaboration de ce modèle se fait, en général, en deux étapes :

- la description du modèle, qui permet de définir les éléments constitutifs du modèle et les liens relationnels entre eux .
- la construction du modèle, qui fournit une réalisation effective du modèle.

Un exemple illustrant la distinction établie ci-dessus est celui d'un objet de révolution. Un tel objet est décrit par un contour plan et un axe de révolution et la construction peut fournir une approximation de l'objet à l'aide de facettes polygonales.

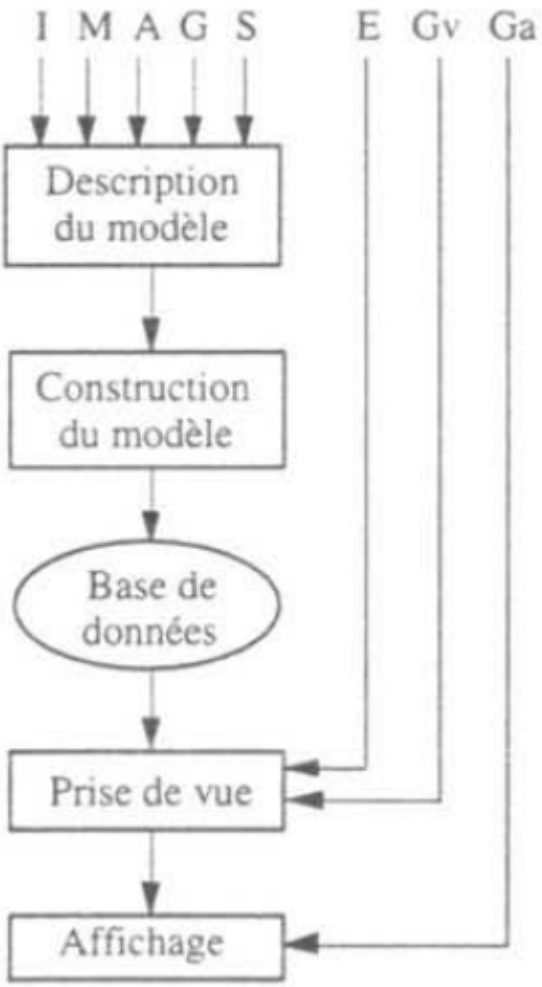


Figure 4 : Le pipeline graphique

VI. Le monde d'interaction direct avec un système graphique

1. Les Dispositifs d'entrée :

- clavier/souris
- tablette graphique
- joystick & trackball
- scanner
- gant numérique

2. Les Données d'entrée :

- chaînes
- valeurs numériques simples (longueur) ou complexes (coordonnées, couleurs, …)
- sélections

VII. Conclusion

L'infographie 3D est une méthode de synthèse d'images qui ne date que d'une trentaine d'années, est aujourd'hui un pilier majeur de l'informatique. Elle intègre à l'imagerie traditionnelle une dimension supplémentaire, la 3ème dimension.

L'objectif du ce chapitre d'introduction à l'Informatique Graphique est de donner les bases de la réalisation de scènes graphiques interactives en dimensions 2 et 3. Nous avons abordé les notions importantes de la programmation graphique à travers des exemples inspirés du domaine du jeu vidéo, de l'architecture de la visualisation scientifique ou du design.

I. Introduction :

OpenGL est une librairie graphique, comportant environ 250 fonctions distinctes.

Ces fonctions permettent la création et la manipulation d'objets tridimensionnels en vue de leur affichage au sein d'applications interactives.

L'un des objectifs des concepteurs de cette librairie a été de la définir indépendamment de toute plate-forme.

Ceci implique que rien n'est prévu dans cette librairie pour gérer les aspects haut niveau liés à l'interaction avec l'utilisateur .

De même, par souci de simplicité et d'universalité , OpenGL ne fournit que des primitives graphiques de très bas niveau pour construire les objets à afficher et à manipuler. À charge à l'utilisateur ou à d'autres bibliothèques spécialisées de construire des objets plus complexes à l'aide de ces primitives.

II. Tracé de primitives 2 D :

II.1 Crénelage et lissage :

Le lissage ou (Antialiasing en anglais) est une technique visant à éliminer l'effet de créneaux quand on trace des lignes ou des points (utilisation des niveaux de gris).

On peut réduire l'effet d'escalier des images, en créant des dégradés de couleurs le long des contours pour les lisser .

Le lissage est un problème d'échantillonnage (discrétisation de valeurs continues).

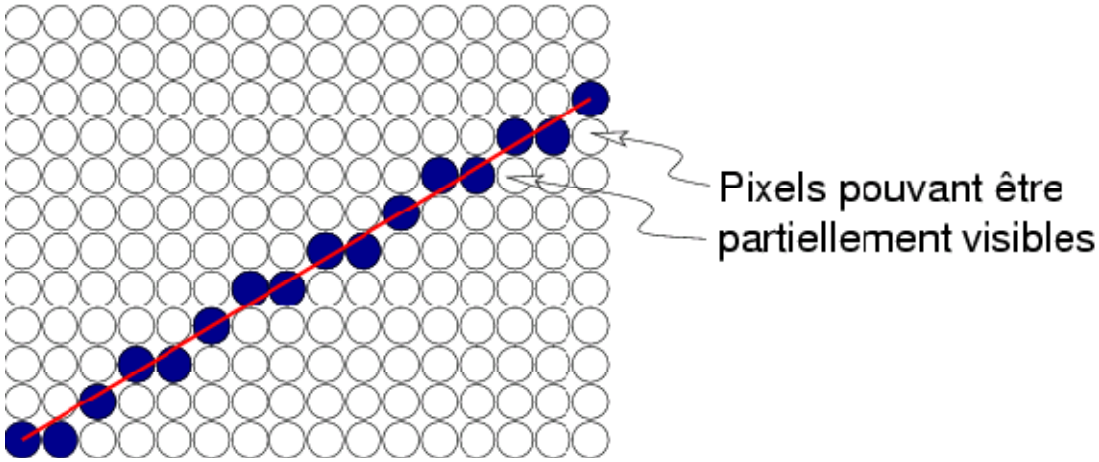


Figure 5 : Le lissage .

II.2 Échantillonnage réalisé en rastérisation :

• Rastérisation : transformation de données géométriques continues (primitives géométrique projetées sur un plan) en une représentation spatiale discrète (pixels sur un écran) .

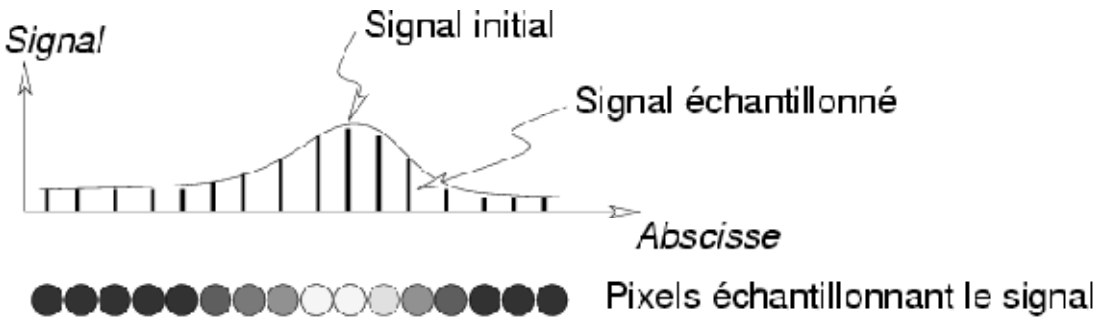


Figure 6 : Échantillonnage réalisé en rastérisation

II.3 L'échantillonnage :

II.3.1 Reconstruction par filtrage :

On peut reconstruire un signal à partir de son échantillonnage par filtrage

II.3.2 Application de la reconstruction :

Permet magnification, minification, projection et changement d'échelle de textures.

Scène échantillonnée (par ex. texture)

reconstruction

Signal continu (texture reconstruite)

transformation (rotation, changement d'échelle, perspective...)

Signal continu transformé

rééchantillonnage

Scène rééchantillonnée (nouveau rendu)

II.3.3 échantillonnage d'aire non pondéré :

- Utilise la forme exacte des primitives géométriques pour estimer la fraction de l'aire d'un pixel couverte par une primitive.
- Pondère le signal de la primitive en fonction de cette fraction d'aire.
- Antialiasing d'une droite: une droite est considérée comme un rectangle.

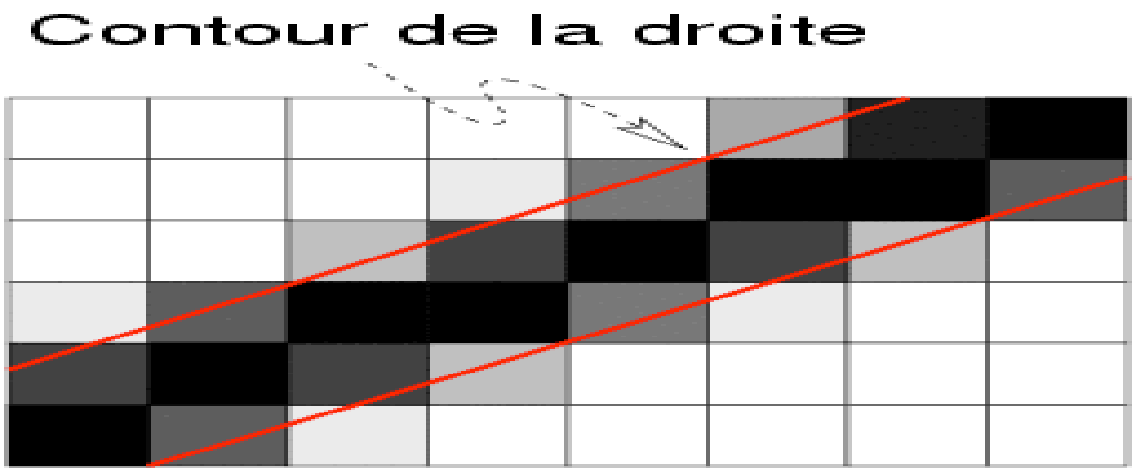


Figure 7 : Antialiasing d'une droite

Principes :

P1 : l'intensité d'un pixel est fonction de l'aire du pixel couverte par la primitive

P2 : les primitives ne contribuent à un pixel que si elles le couvrent

P3 : toutes les zones de la primitive ont la même contribution

Le calcul de la surface couverte peut être approximé en divisant un pixel en sous-pixels et en comptant le nombre de sous-pixels couverts par la primitive.

II.3.4 Échantillonnage d'aire pondéré :

Fonction de pondération :

- Une fonction de pondération proportionnelle à la distance du centre de la primitive adoucit les contrastes entre les pixels
- Rechercher l'intégrale de cette fonction de pondération revient à chercher le volume d'intersection d'un cône avec le pixel
- Le cône dépasse généralement la primitive en surface (le rayon du cône vaut l'unité de la grille).

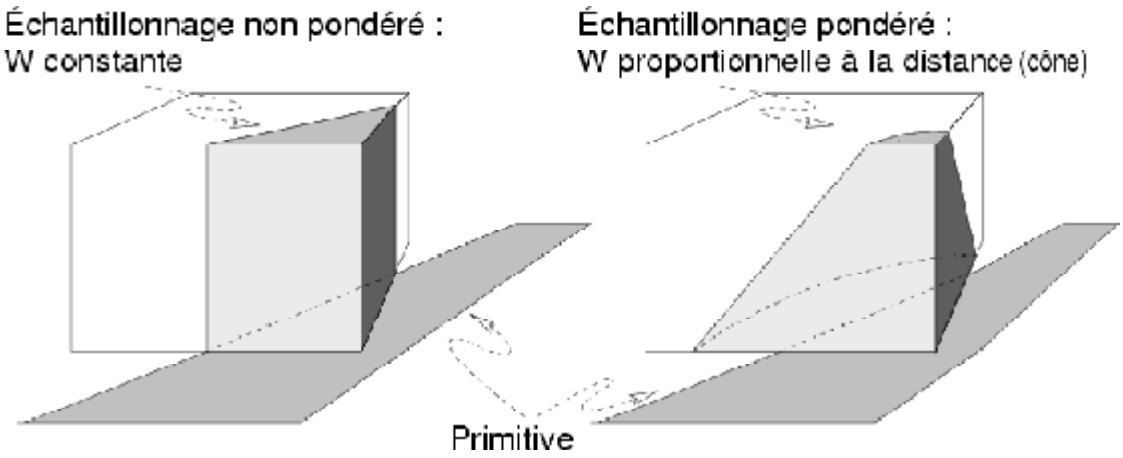


Figure 8 : Comparaison visuelle entre les deux pondérations

III. Les collisions et intersections :

Principe :

méthodes rapides pour trouver les contacts/intersections entre volumes :

Modifications de la scène en conséquence changements de trajectoires, déformations .

Simplifications et optimisations pour que la gestion des intersections et des collisions soit *compatible avec un rendu temps réel.*

Constats:

Recherche systématique des intersection entre des triangles de maillages lourde et inopérante.

Dans des scènes animées, l'œil est assez tolérant aux approximations sur la gestion des collisions.

III.1 Applications :

III.1.1 Applications du calcul des intersections :

• évitement d'obstacles (réalité virtuelle, réalité augmentée, jeux, robotique...) .
• saisie d'objets (télémédecine, télémaintenance, formation...) .
• tracé par lancer de rayon: rendu photo-réaliste .
• pointage (souris ...) .
• Applications de la gestion des collisions.
• simulation de systèmes physiques (jeux, simulation scientifique, effets spéciaux et animation...) .

Figure 9 : Effets spéciaux générés en informatique graphique

III.1.2 Les méthodes des collisions et intersections :

1.Volumes englobants :

simplification de la géométrie et éventuellement hiérarchisation

2. Cohérence spatiale :

régularité géométrique et connexité des volumes

3. Cohérence temporelle:

régularité géométrique et connexité des trajectoires

III.1.3 Les étapes de calcul de collisions :

• Détection de collision : réponse booléenne sur la possibilité de collision entre deux objets

• Détermination de collision : localisation précise de la collision

• Réponse : modifications cinématiques et déformations géométriques

III.1.4 Les intersections :

Pour accélérer et simplifier les recherches d'intersections, on crée des volumes englobants qui sont supposés contenir tout le volume initial.

Il existe trois modes de construction :

1. Méthode ascendante bottom-up :

combine des primitives sur des critères de proximité

2. Méthode descendante top-down :

part d'un volume englobant maximal puis subdivision récursive: recherche d'un axe de subdivision et d'un lieu de subdivision sur cet axe Insertion incrémentale d'arbre :

3. insertion d'une nouvelle primitive à chaque étape et reconfiguration de l'arbre pour optimiser le placement de la primitive (pour minimiser la profondeur de l'arbre obtenu) .

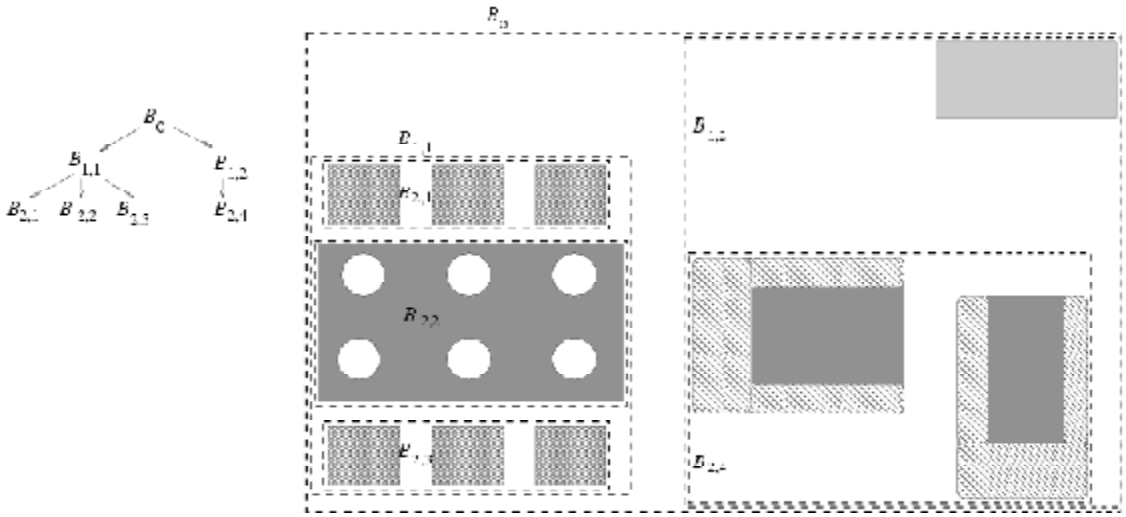


Figure 10 : Volumes englobants hiérarchie

IV. Texture :

IV.1 Application de texture :

texturage : transfert d'une texture définie sur une surface canonique en 2D vers une zone d'une représentation 2D d'un volume 3D.

En OpenGL, on fait précéder la définition de chaque sommet par la définition de ses coordonnées dans la texture d'origine (dans le carré canonique (0,0), (1,1)).

Cas particulier : texturage d'une surface plane.

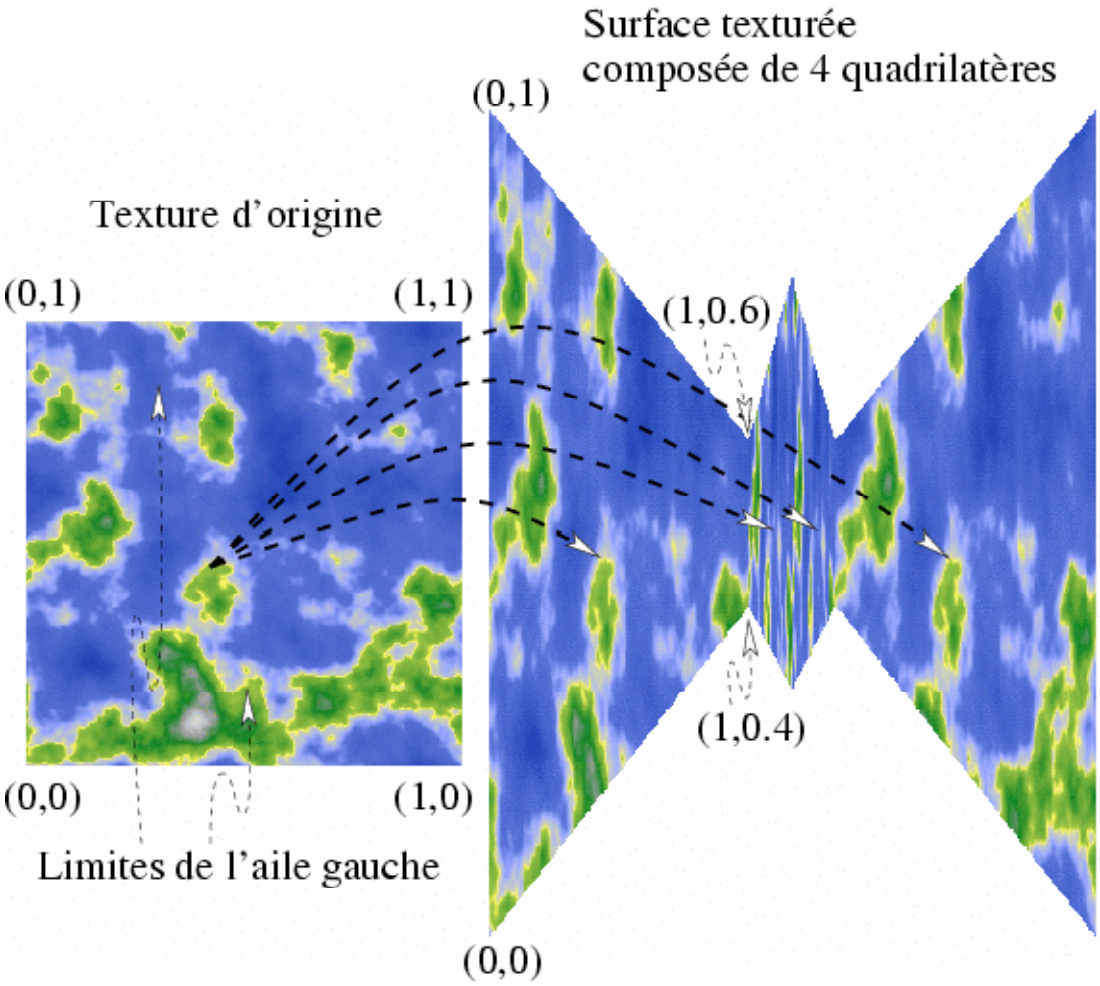


Figure 11 : Exemple de surface texturé

IV.2 Deux modèles d'interpolation de texture :

IV.2.1 linéaire vs. Perspective :

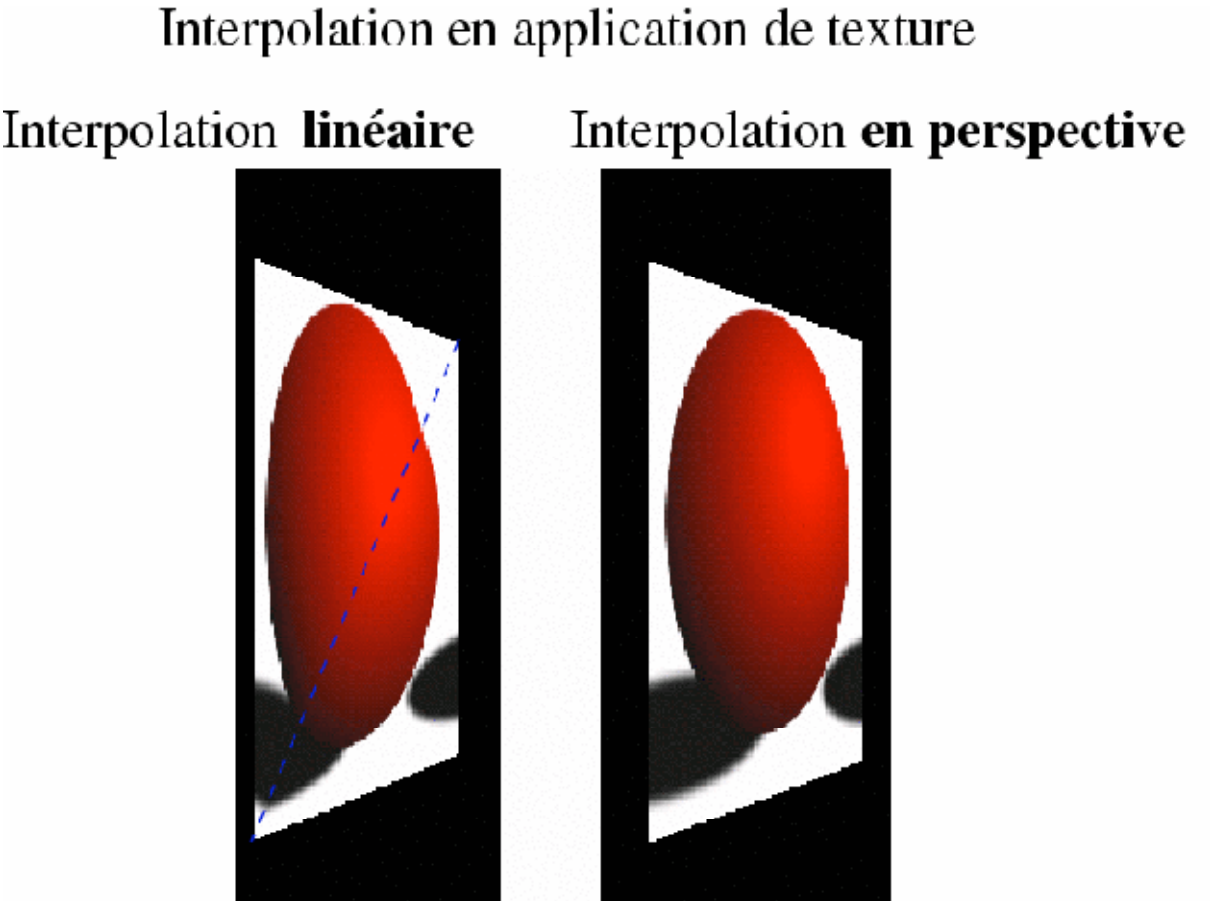


Figure 12 : Illustration des deux modes d'interpolation

le quadrilatère est tracé au moyen de deux triangles : l'interpolation linéaire est problématique à la frontière entre ces deux triangles.

IV.2.1.1 Interpolation linéaire :

Ce type d'interpolation est une transformation affine du quadrilatère d'origine vers le quadrilatère cible.

Elle conserve les rapports des combinaisons affines.

Application au tracé d'un quadrilatère texturé.

IV.2.2 Interpolation en perspective vs .interpolation linéaire :

1. projection en perspective

- conserve l'alignement (transformation affine)
- ne conserve pas les rapports des combinaisons affines

2. l'interpolation linéaire ne peut pas rendre compte fidèlement des images par une projection en perspective.

V. Couleur :

On peut distinguer deux approches pour définir l'intensité lumineuse :

• En physique : l'intensité dépend du nombre d'électrons dans le flux reçu.

• En psychologie de la perception :

La *brillance* est une notion subjective liées aux capacités visuelles (et cognitives) humaines qui est différente de la réalité physique.

La même différence d'intensité est perçue différemment selon la gamme où elle est située : on perçoit mieux les changements à faible intensité.

V.1 Correction Gamma :

Objectif c'est de modifier les intervalles d'intensité pour qu'ils semblent linéaires sur le plan perceptif.

On utilise pour cela une échelle logarithmique : $x' = xg$

La correction Gamma sert aussi à rééchelonner les intensités lumineuses pour une photo sur ou sous-exposée.

Figure 13 : correction Gamma d'une photo de nuit sous-exposée.

V.2 Désignation des couleurs :

Une couleur pure combinée à du blanc ou du noir produit différent *tons* :

- ils correspondent tous à la même teinte
- leur saturation (quantité de couleur par rapport au gris de même intensité) varie
- leur clarté (value/lightness) varie (position entre le blanc et le noir)

V.3 Analyse des couleurs :

Il existe trois types de cônes sur la rétine respectivement sensibles au bleu, au vert et au rouge.

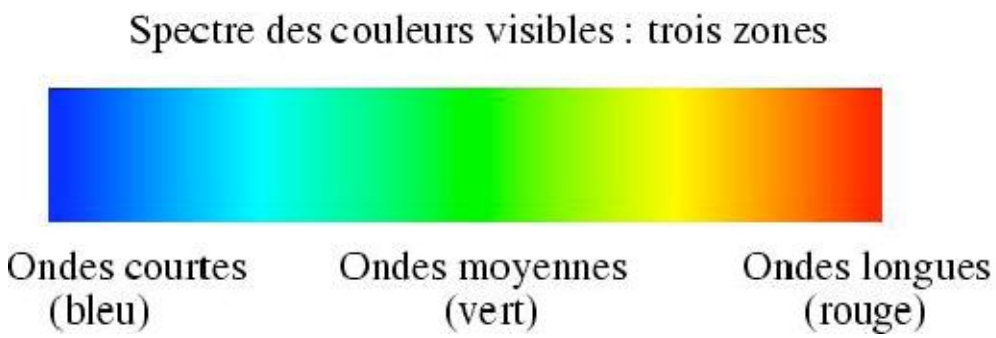


Figure 14 : Spectre des couleurs visibles

Il existe trois modèles de couleur :

Les couleurs s'obtiennent en mélangeant trois couleurs primaires.

» Modèle universel : modèle de la Commission Internationale de l'Éclairage (CIE) :

Modèle sans recours à des coefficients négatifs et reposant sur trois primaires standard X, Y et Z.

Deux modèles particuliers :

» Rouge Vert Bleu RVB (RGB) :

Combinaison additive des couleurs primaires.

» Cyan Magenta Jaune CMJ (CMY) :

Combinaison soustractive des couleurs primaires.

Le spectre visible dans le diagramme de chromaticité de la CIE (1931)

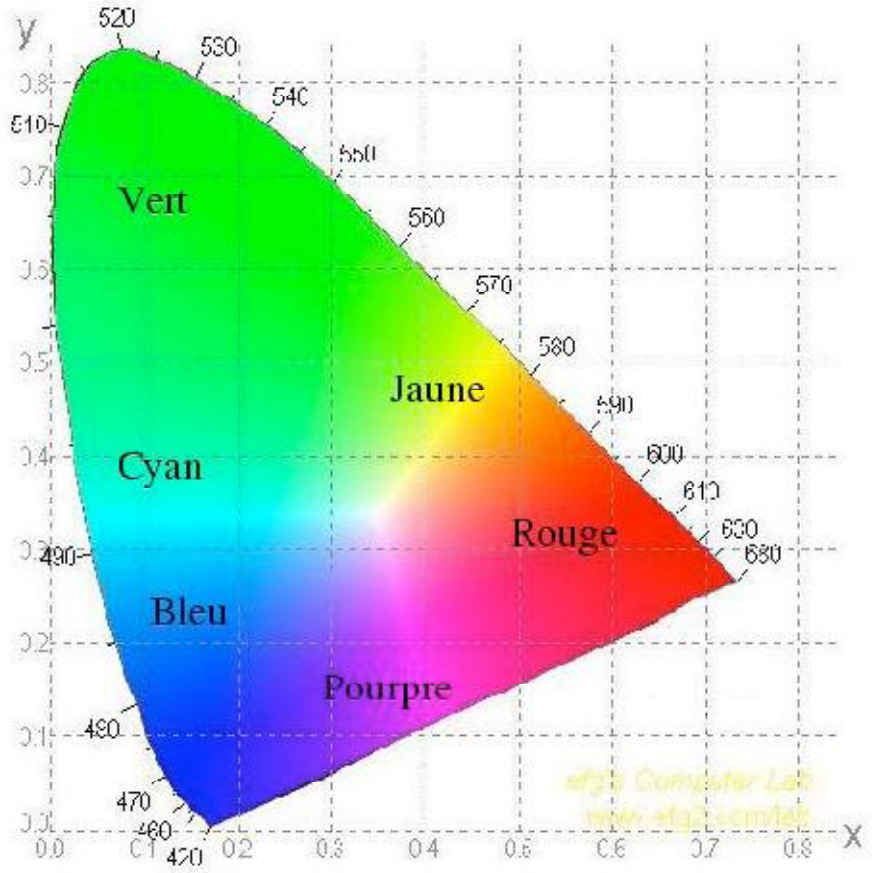


Figure 15 : Le diagramme de chromaticité de la CIE

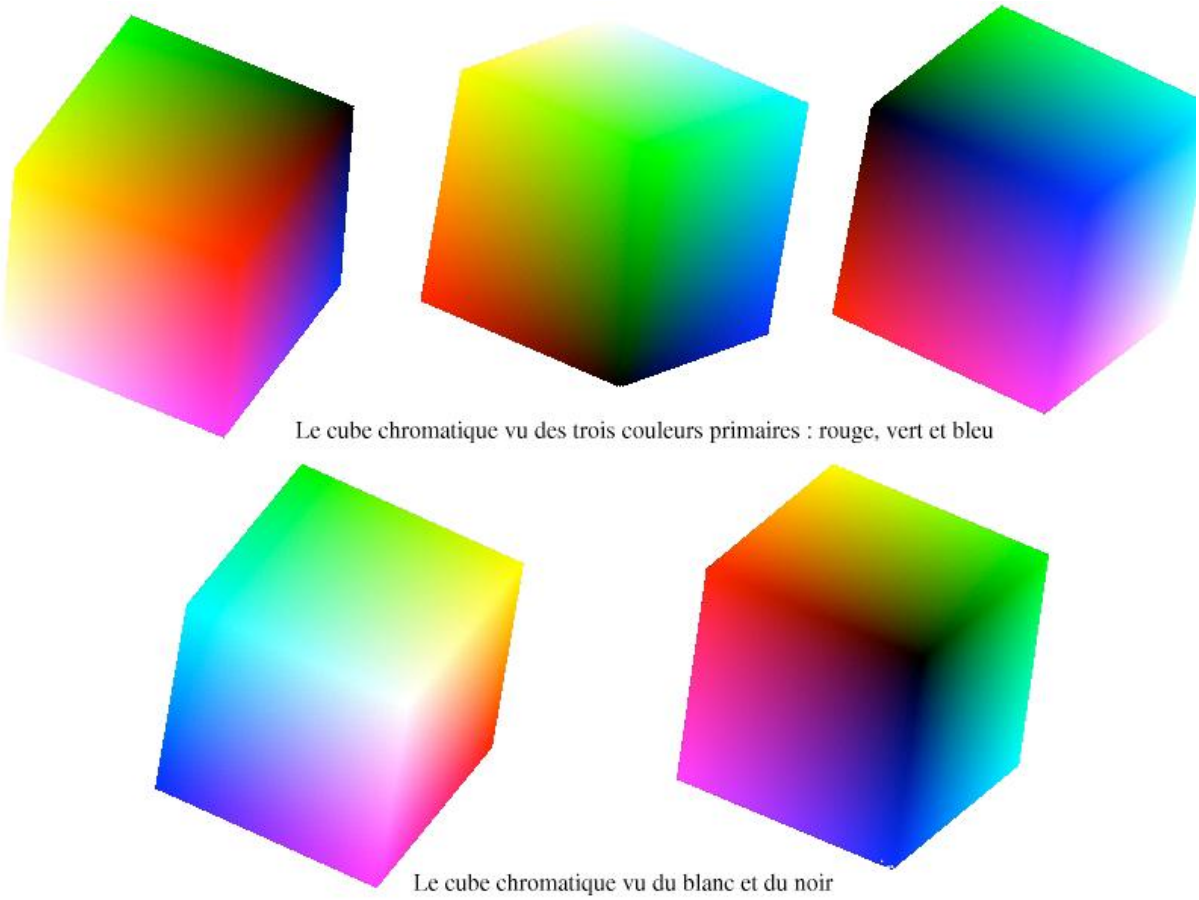


Figure 16 : Modèle RVB

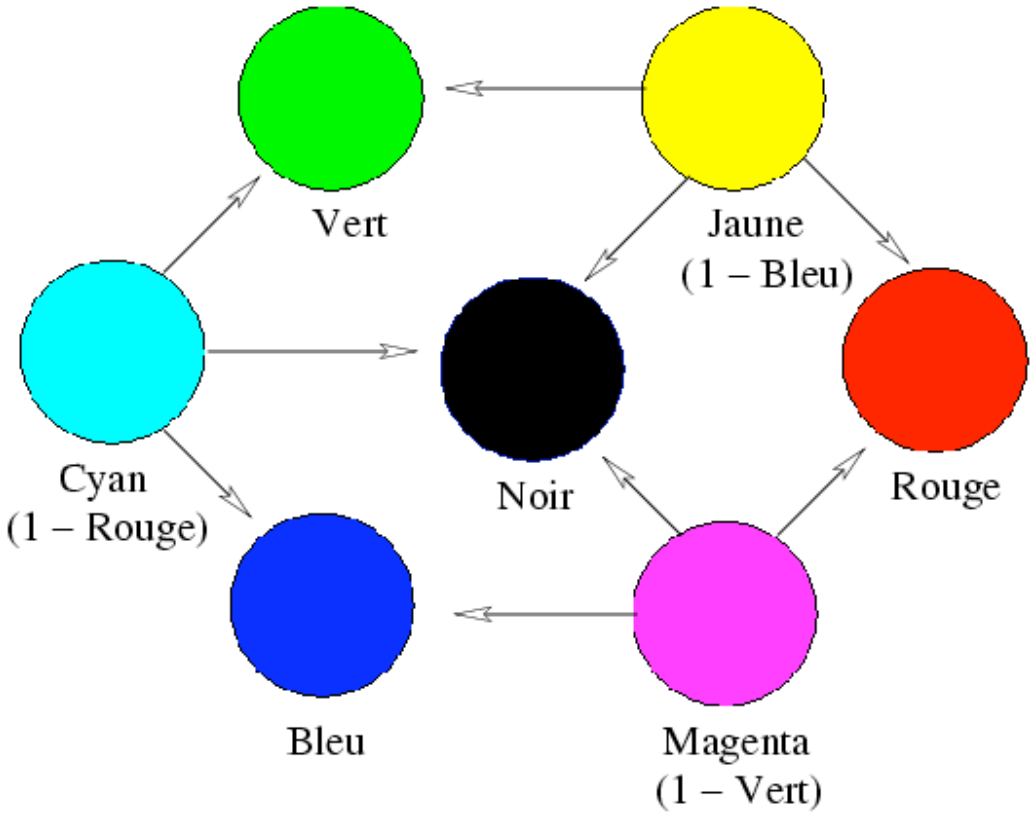


Figure 17 : Modèle CMJ

VI. Suppression de parties cachées :

1. Les Points

Le calcul des parties cachées se fait généralement après la projection en perspective. On a donc deux types d'information :

- Abscisse et ordonnée dans le plan de projection pseudo-profondeur vue dans le cours sur les projections.
- Les coordonnées des points dans le plan de projection ne sont pas arrondies pour la précision des calculs de parties cachées.

2. Les Faces

• sommets de la face avec 3 coordonnées

• équation du plan de la face

• coordonnées du cube englobant la face pour accélérer le clipping et les calculs de parties cachées .

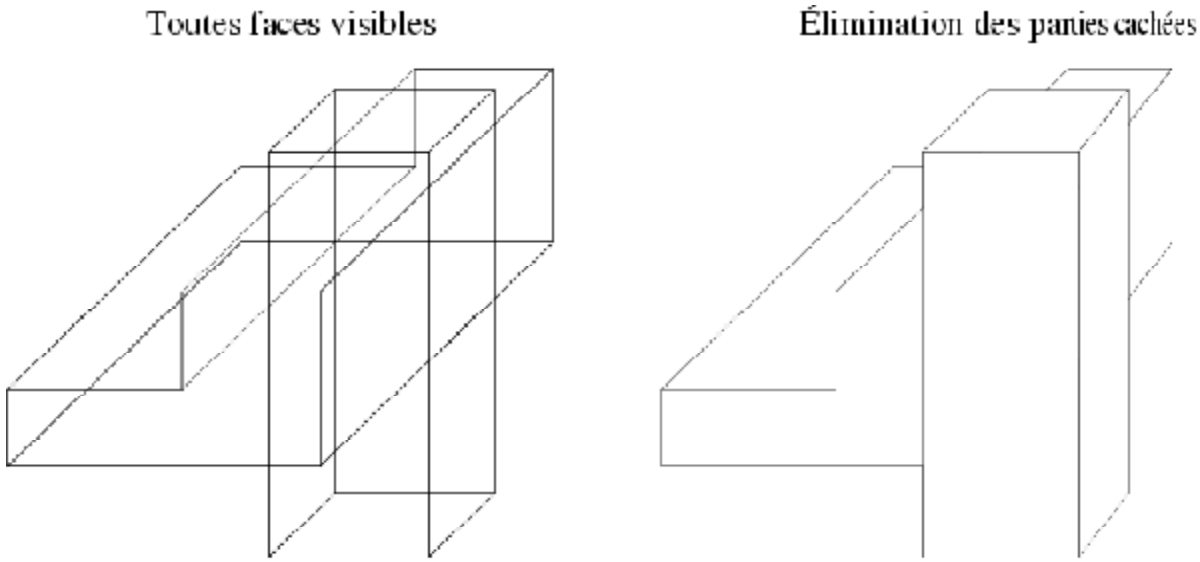


Figure 18 : Élimination de faces cachées

VII. Notion du clipping

» limiter le tracé d'une figure à une région déterminée .

» rechercher l'intersection entre des figures géométriques simples et des zones graphiques élémentaires .

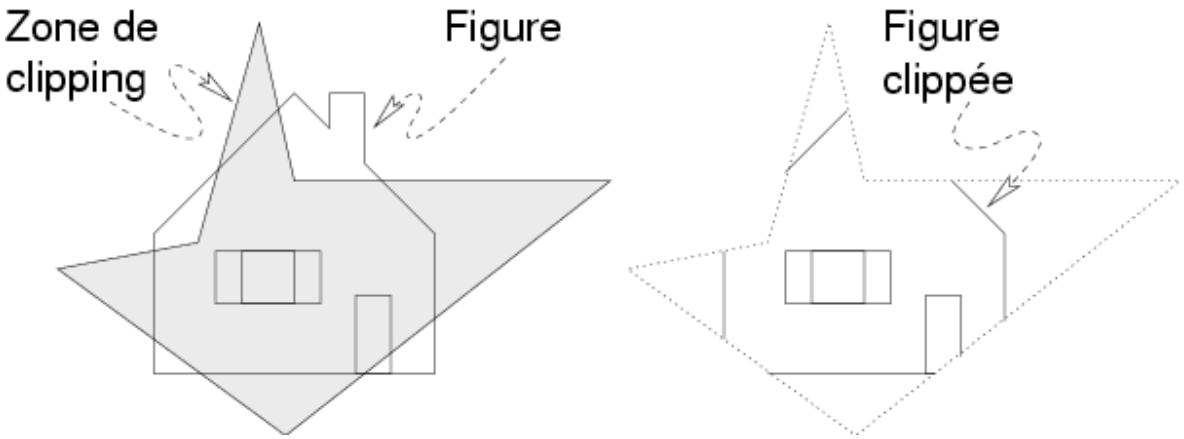


Figure 19 : Le clipping

VIII . Les composantes du rendu d'un volume éclairé :

» Sources lumineuses

Induisant des éclairements différents selon l'orientation des faces par rapport à ces sources (*fig1*)

» Modèle d'ombrage

fournissant un rendu lisse (*fig2*)

» Modèle d'illumination

Composantes émises ou réfléchies de la lumière (aspect brillant ou réfléchissant des objets) (*fig3*)

» Texture

Apparence d'un matériau tel que le bois ou la pierre (fig4)

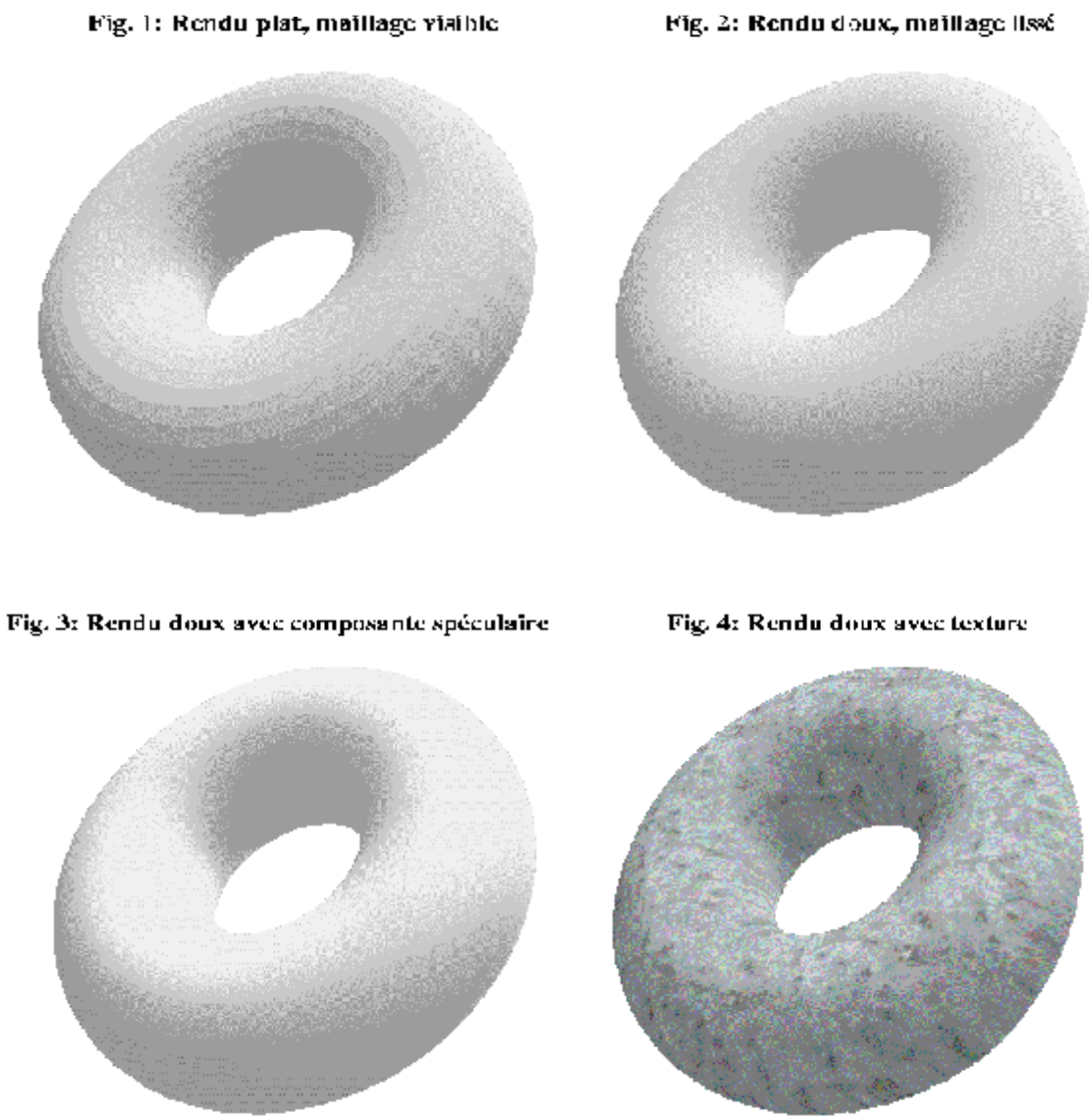


Figure 20 : Éclairement et rendu réaliste

VII. Conclusion

On retiendra tout au long de ce chapitre qu'un objet dans une scène 3D est défini par un ensemble de primitives graphiques (point, segment, triangle) et un contexte de visualisation .

De même ,OpenGL est une librairie graphique qui permet de pouvoir créer des applications graphique 3D ou 2D , c'est l'une des meilleur librairie graphique à ce jour.

Un autre grand interré de OpenGL c'est qu'il permet d'avoir un code propre et simple, pas trop compliqué car il est simple, facile pour débuter et il peut dessiner des choses très belles avec quelques efforts, enfin c'est une librairie très bien pour les amateurs mais aussi pour les professionnels.

I. Qu'est-ce qu'OpenGL

OpenGL est une librairie graphique 3D disponible sur de nombreuses plateformes (portabilité) qui devenu un standard en infographie.

C'est un langage procédural (environ 200 fonctions) qui permet de donner des ordres de tracé de primitives graphiques (segments, facettes, etc.) directement en 3D. C'est aussi une machine à états qui permet de définir un contexte de tracé : position de caméra, projection 2D, couleurs, lumières, matériaux... OpenGL se charge de faire les changements de repère, la projection à l'écran, le « clipping » (limites de visualisation), l'élimination des parties cachées, l'interpolation les couleurs, et de la « rasterisation » (tracer ligne à ligne) des faces pour en faire des pixels.

OpenGL s'appuie sur le hardware disponible (selon la carte graphique). Toutes les opérations de

base sont a priori accessibles sur toute machine, simplement elles iront plus ou moins vite selon qu'elles sont implémentées au niveau matériel ou logiciel.

Ce que ça fait :

1. sélection d'un point de vue.
2. tracé de polygones.
3. organisation des objets dans une scène.
4. effets de profondeur (luminosité décroissante, brouillard, flou de type photographie).
5. placage de texture.
6. éclairage des surfaces (en 3D).
7. lissage ou anti-crénelage (anti-aliasing) à la « rasterisation ».

Ce que ça ne fait pas:

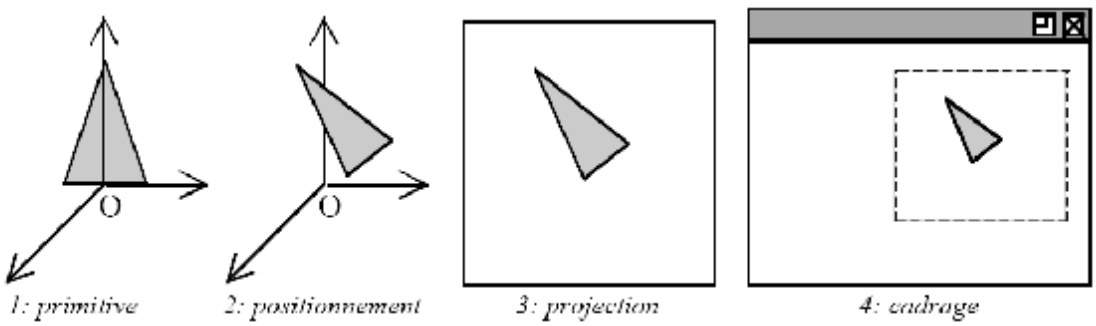
- ne gère pas le fenêtrage,
- ne gère pas les interruptions (interactions utilisateurs).

Une scène Virtuelle 3D n'est jamais mémorisée dans sa totalité et chaque primitive énoncée est immédiatement projetée « à l'écran » après un calcul de rendu plus ou moins complexe. Ce calcul se fait d'après un contexte prédéfini précisant la couleur, l'éclairage, le

point de vue, le type de projection etc… La simple modification de la scène ou du contexte de visualisation impose une reconstruction complète de la scène 3D car aucun résultat intermédiaire n'a été conservé.

Aussi, on retiendra tout au long de ce chapitre qu'un objet dans une scène 3D est défini par un ensemble de primitives graphiques (point, segment, triangle) et un contexte de visualisation. En simplifiant, le tracé d'une primitive déclenche immédiatement la séquence d'opérations suivantes :

1. définition de la primitive dans le repère de l'objet
2. transformation géométrique sur la primitive (positionnement de l'objet dans la scène)
3. projection de la primitive sur le plan image
4. cadrage dans la fenêtre de visualisation



Le plan de cet exposé ne respecte pas l'ordre des opérations effectuées par OpenGL. C'est simplement une manière d'aborder la construction d'une image. Nous parlerons de :

- GLUT : un système simple de fenêtrage
- Couleur et primitives de tracé (lignes et polygones)
- Transformations géométriques pour la construction d'une scène 3D
- Visualisation d'une scène 3D (z-buffer et projection 2D)
- Amélioration du rendu (ou rendu réaliste) : brouillard, éclairage, …
- Le cas des images 2D

Nous proposons de présenter ici un sous-ensemble des fonctionnalités d'OpenGL qui permettra une programmation déjà efficace. Certaines restrictions seront faites pour aller à l'essentiel.

La première concerne le mode d'affichage que nous limiterons au mode couleur RGB pour la totalité de ce cours.

I.1. Philosophie des identificateurs OpenGL

OpenGL a pris la précaution de redéfinir les types de base pour assurer la portabilité des applications quelque soit l'implémentation. Si la conversion entre les types de base du C et les types GL est bien assurée (par exemple, pour les arguments passés par valeur dans une fonction OpenGL lors d'un appel), il est fortement conseillé d'utiliser les types OpenGL pour les tableaux qui sont passés par adresse.

Par ailleurs, OpenGL s'est imposé certaines règles pour créer ses identificateurs : ceux-ci disposent presque systématiquement d'un préfixe et d'un suffixe :

- Préfixe : gl*Nom*
- Suffixe : précise parfois le nombre d'arguments et leur type. La même fonction est souvent disponible pour différents type d'arguments. Par exemple, la définition d'une couleur peut se faire avec les fonctions suivantes :

glColor4f(*rouge, vert, bleu, transparence*)
glColor3iv (*tableDeTroisEntiers*)

Suffixe	**type**	**taille**	**nature**
b	GLbyte	8	octet signé
s	GLshort	16	entier court
I	GLint	32	entier
f	GLfloat	32	flottant
d	GLdouble	64	flottant double précision
ub	GLubyte	8	octet non signé
us	GLushort	16	entier court non signé
ui	GLuint	32	entier long non signé
*type*v			adr d'un vecteur de ce *type*

I.2. La librairie GLUT :

OpenGL a été conçu pour être indépendant du gestionnaire de fenêtres qui est intimement lié ausystème d'exploitation. Il existe toutefois un système de fenêtrage « élémentaire » qui permet de développer des applications graphiques dans un cadre simple tout en garantissant une très bonne portabilité sur de très nombreuses plate-formes : openGL Utility Toolkit (GLUT).

Les fonctionnalités de cette bibliothèque permettent principalement de :

- créer et gérer plusieurs fenêtres d'affichage
- gérer les interruptions (click souris, touches clavier, …)
- disposer de menus déroulant
- connaître la valeur d'un certain nombre de paramètres systèmes,

Quelques fonctions supplémentaires permettent de créer simplement un certain nombre d'objets 3D (cube, sphère, tore, …)

Cette bibliothèque s'enrichie régulièrement d'outils simples et pratiques (on trouve maintenant sur les sites OpenGL des boutons, des affichages de répertoires, …) sans devenir un « monstre » dont la maîtrise demande une longue pratique.

La philosophie générale de ce système de fenêtrage est basée sur la «programmation événementielle » (ce que l'on pourra regretter …), ce qui impose une structuration assez particulière de l'application.

II. Construire des images avec OpenGL :

II.1 La syntaxe d'OpenGL :

La syntaxe d'OpenGL caractérise les constantes, types et fonctions de la manière suivante :

_ les constantes : **GL_CONSTANTE** (GL_COLOR_BUFFER_BIT par exemple);

_ les types : **GLtype** (GLbyte, GLint par exemple);

_ les fonctions : **glLaFonction** (glDraw, glMatrixMode par exemple).

À cela s'ajoute, dans la syntaxe des fonctions, la caractérisation du nombre et du type des arguments par un suffixe. Par exemple :

glVertex3f(1.0, 2.0, 3.0).

définit les coordonnées dans l'espace d'un sommet (vertex) en simple précision réelle. Pour définir les coordonnées du plan d'un sommet par des valeurs entières, on utilise :
glVertex2i(1, 2).

II.2 Le principe :

OpenGL dessine dans une image tampon. Cette image tampon peut être soit directement la mémoire vidéo de la fenêtre graphique, soit une image tampon intermédiaire permettant de faire du "double buffering".

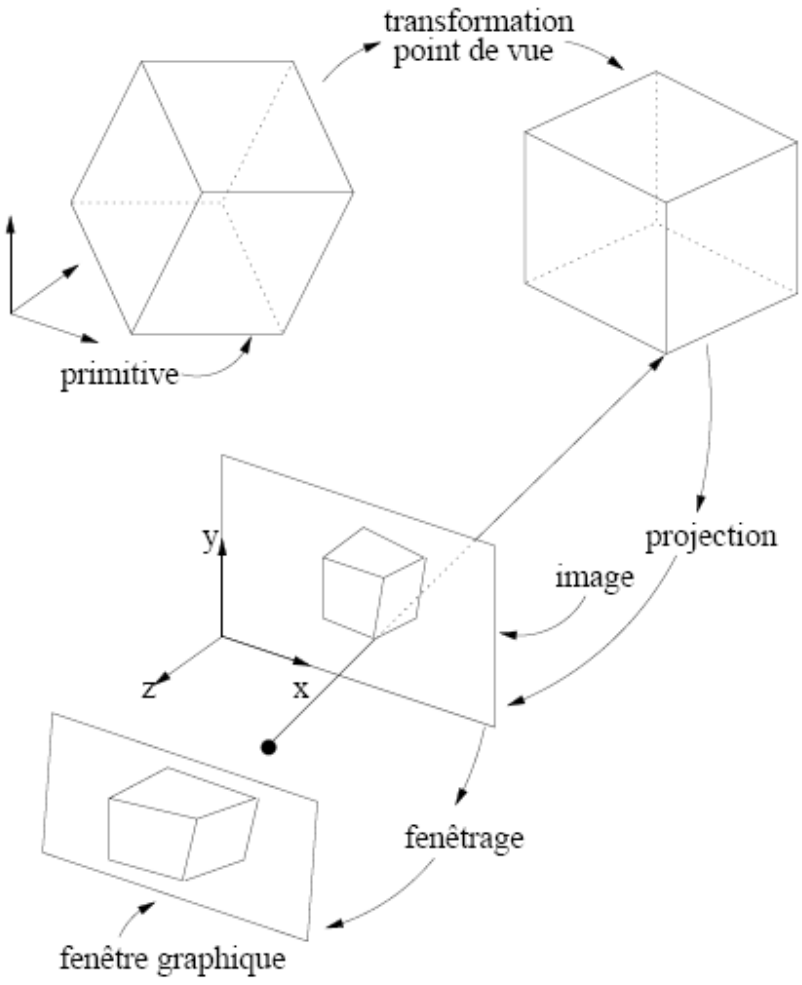


Figure 21: La construction d'une image

Les différentes étapes de la génération d'une image sont :

1. spécification des primitives à dessiner : les primitives sont définies dans un certain repère.
2. transformation *point de vue* : une transformation est appliquée aux primitives à dessiner. Cette transformation sert à fixer le point de vue des primitives ou, en d'autres termes, la position du plan image.
3. Projection : les primitives sont projetées sur le plan image suivant la projection spécifiée (orthographique, perspective).
4. Fenêtrage et numérisation : l'image obtenue est redimensionnée suivant les tailles de la fenêtre graphique et les primitives projetées sont numérisées sous la forme de pixels dans la mémoire vidéo.

Ces quatre étapes sont réalisées directement à l'appel d'une fonction de dessin d'OpenGL et ne nécessitent pas quatre appels spécifiques. En effet, les spécifications du point de vue, de la projection et du fenêtrage se font de manière indépendante de la spécification des primitives. Cela veut dire qu'à chaque appel d'une fonction de dessin, la transformation *point de vue* courante et la projection courante sont appliquées aux primitives. La figure 2 montre, par exemple, ce qui est effectué lors de l'appel d'une fonction de dessin d'une ligne polygonale.

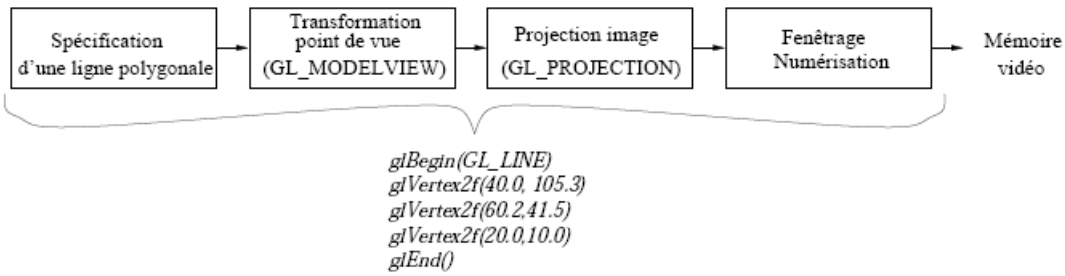


Figure 22: Le dessin d'une ligne polygonale avec OpenGL

Les paragraphes suivant explicitent les spécifications de ces différentes étapes.

II.3 Spécifier la transformation *point de vue* :

Les transformations sont représentées dans OpenGL sous la forme de matrices. Spécifier la transformation courante appliquée aux primitives avant la projection consiste donc à définir la matrice de cette transformation. OpenGL utilise les coordonnées homogènes pour

effectuer des transformations. La transformation *point de vue* est donc représentée par une matrice 4x4. De plus OpenGL stocke les matrices dans des piles. Ainsi, la transformation *point de vue* correspond, dans OpenGL, non pas à une matrice mais à une pile de matrice. De cette manière, OpenGL appliquera l'ensemble des transformations empilées aux primitives à dessiner avant de les projeter. Pour modifier la transformation *point de vue*, il faut tout d'abord sélectionner la pile des transformations *point de vue*, puis ensuite modifier la transformation en conséquence.

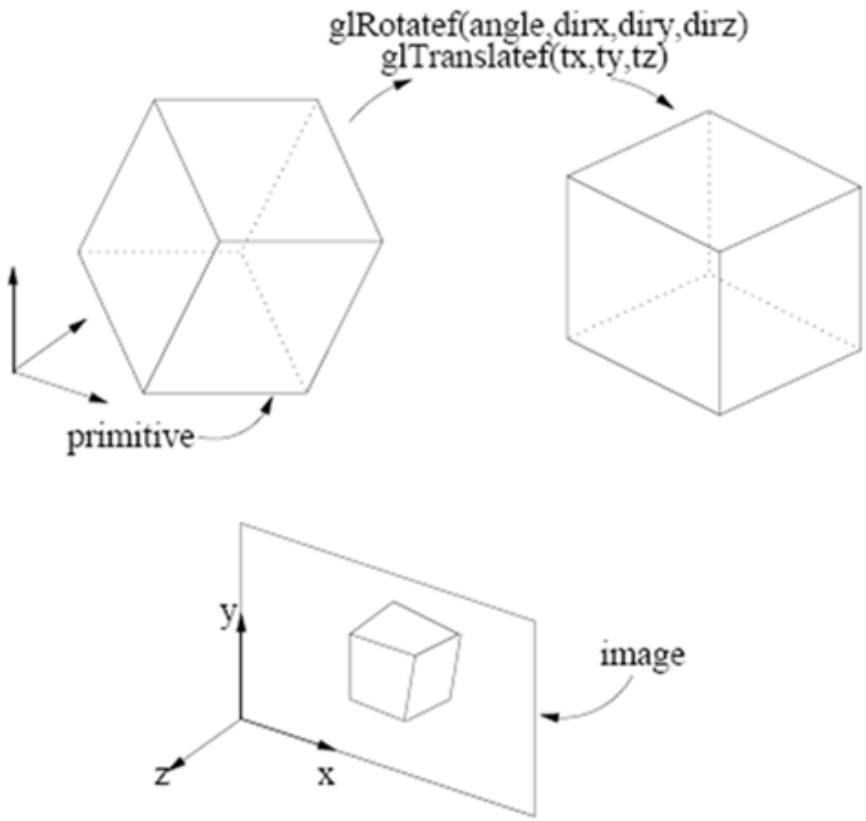


Figure 23 : La transformation point de vue

```
glMatrixMode(GL_MODELVIEW); /* la pile des transformations point */
/* de vue est selectionnee */
glLoadIdentity(); /* initialisation de la transformation */
```

glRotatef(angle, /* rotation de angle degres autour de */
dirx, diry, dirz); /* l'axe de direction (dirx, diry, dirz)*/
glTranslatef(tx, ty, tz); /* translation de vecteur (tx,ty,tz) */
Algorithme de modification de la transformation *point de vue* (voir figure 3).
Quelques remarques :
_ les rotations s'expriment en degrés.
_ Sans appel de la fonction **glLoadIdentity()**, la rotation et la translation définies ensuite s'ajoutent à la transformation courante.
_ Les modifications sont appliquées à la matrice en haut de pile alors que la transformation *point de vue* comprend l'ensemble des matrices empilées.

II.3.1 Ajouter une transformation pour le calcul du point de vue :

Pour ajouter une matrice à la pile courante (GL_MODELVIEW, GL_PROJECTION) ou en enlever une, on utilise les fonctions **glPushMatrix()** et **glPopMatrix()**.
Cela peut être utilisé pour appliquer une transformation supplémentaire à une primitive donnée (voir figure 4). Par exemple, l'algorithme suivant trace la projection d'un cube auquel est appliqué une rotation puis les transformations contenues dans la pile *point de vue*. Ensuite le même cube est projeté après avoir subi les transformations *point de vue* sans la rotation précédente :

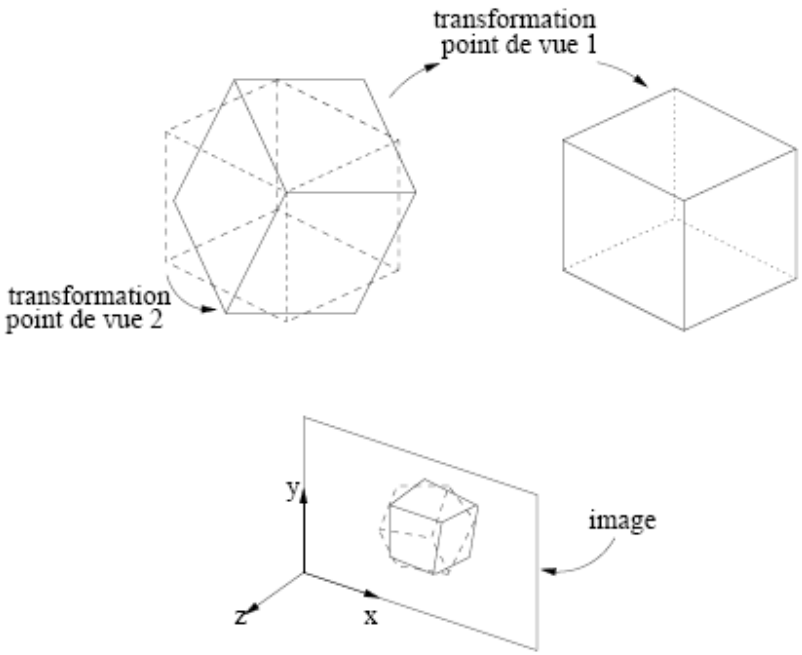


Figure 24: Empilement des matrices de transformation point de vue pour modifier position et orientation des primitives.

II.4 Transformation géométrique :

II.4.1 Variable d'état :

glMatrixMode(GLenum mode) sert à changer la valeur de la variable d'état **GL_MATRIX_MODE** (3 valeurs possibles **GL_MODELVIEW**, **GL_TEXTURE** et **GL_PROJECTION**).

II.4.2 Composition des transformations :

- quand **glMatrixMode()** est appelée, toutes les transformations suivantes affectent la matrice de ce mode en se composant les unes aux autres.
- Dans la suite, nous ne considérons que les compositions de transformations sur la matrice de modèle de vue.

II.4.3 Translation :

- **glLoadIdentity()** réinitialise la matrice courante à la matrice identité.
- **glTranslate(TYPE x, TYPE y, TYPE z)** multiplie la matrice.

II.4.3.1 Rotation :

glRotate(TYPE angle, TYPE x, TYPE y, TYPE z) multiplie la matrice courante par une matrice qui fait tourner un objet d'un angle *a* autour de l'axe passant par l'origine, de vecteur directeur (x,y,z).

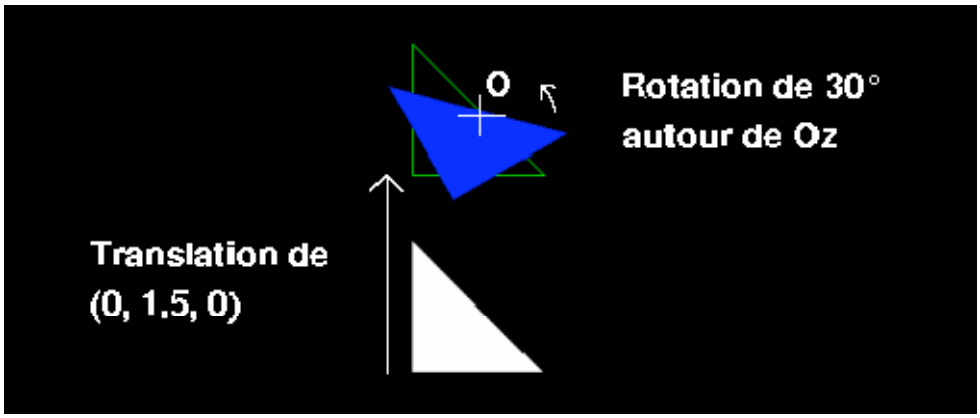


Figure 25 : Translation vertical du triangle d'origine

III. Types de projection :

Il existe deux types de projection telle que la projection orthographique et la projection perspective.

III.1 Projection Orthographique :

Pour définir une projection image de type orthographique, soit une projection suivant la direction orthogonale au plan image, il faut définir la direction de projection.

Dans OpenGL, la direction de projection est celle de l'axe des z du repère correspondant au point de vue (soit après la transformation *point de vue*).

volume de vue : parallélépipède rectangle

Propriétés :

- conserve le parallélisme quelle que soit l'orientation des droites.
- conserve les rapports de longueurs sur des droites parallèles.

• **glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)** définit :

- les dimensions des rectangles de clipping (proche et lointain).
- leurs positions sur *Oz* .

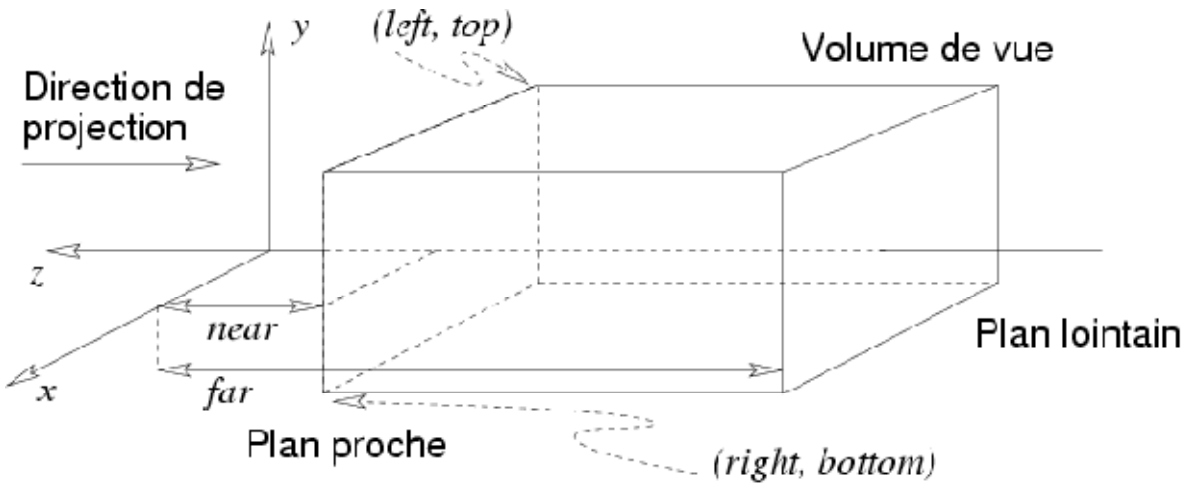


Figure 26 : Volume de vue en projection orthographique

III.2 Projection en perspective :

Pour définir une projection de type perspectif, il faut définir un centre de projection, une distance focale (distance du centre de projection au plan de projection) et une direction de projection. Dans OpenGL, la direction de projection est toujours suivant l'axe des z. Le centre de projection est à l'origine du repère courant et la distance focale est paramétrable.

volume de vue : tronc de pyramide

Propriétés :

- ne conserve pas le parallélisme : des droites parallèles convergent vers un point de fuite
- les objets lointains apparaissent plus petits que les objets proches.

• **glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)** définit :

- les dimensions du rectangle de clipping proche.
- les positions des rectangles de clipping proche et lointain sur *Oz* (distances positives).

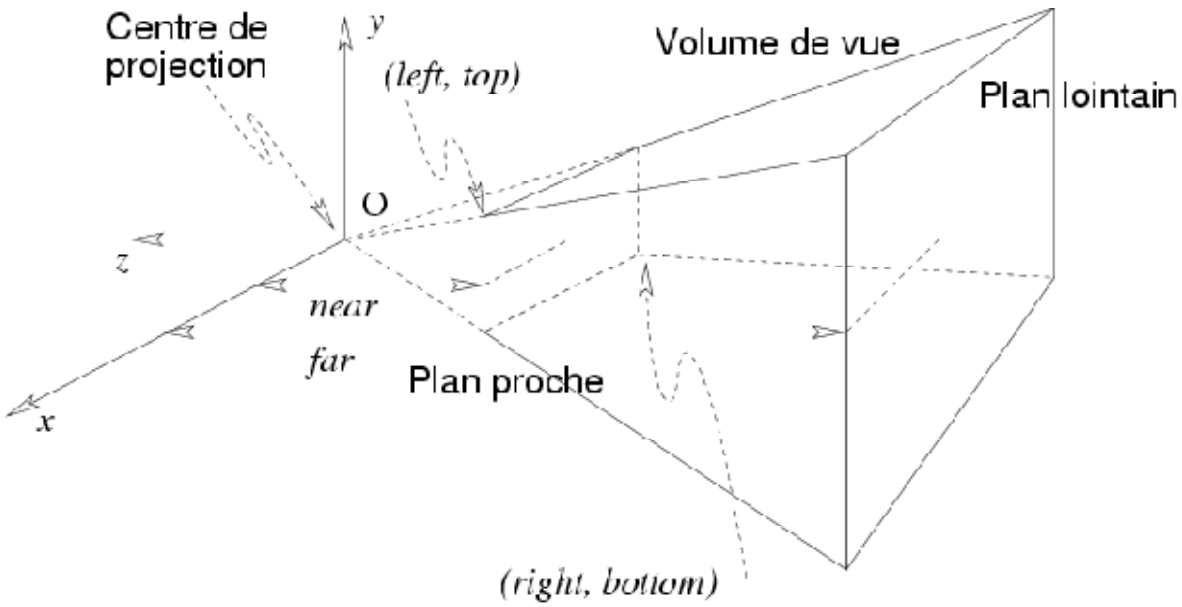


Figure 27 : Volume de vue en projection en perspective

IV. Texture :

IV.1 Plaquer des textures :

Les textures confèrent aux images de synthèse un réalisme indéniable. Le principe est d'utiliser, pour remplir un élément de surface de la scène observée, une texture (en général une image). Le plaquage de texture consiste donc à définir, pour les points texturés de l'image à synthétiser, quelles sont les points de références correspondants dans une image (2D ou 1D)

de texture (ou de quelle manière déterminer ces points de références), et comment utiliser ces points de références.
L'utilisation des textures dans OpenGL s'effectue suivant deux étapes principales :

1. Création de la texture : spécifier l'image source pour la texture et la méthode de plaquage à utiliser.
2. Plaquage de la texture : spécifier les points de la scène délimitant la *zone* de plaquage.

IV.2 Définir une texture :

La création d'une texture nécessite dans un premier temps une image source. Cette image source peut soit être crée à l'intérieur du programme, soit être lue dans un fichier. La définition d'une texture à partir de cette image source s'effectue ensuite en plusieurs étapes dont certaines découlent du besoin d'utiliser plusieurs textures de façon simultanée :
1. Identifier.
2. spécifier la texture courante.
3. spécifier le plaquage.
4. spécifier l'image de texture.
5. spécifier le mode de rendu.

Identifier : Pour pouvoir manipuler plusieurs textures, il est nécessaire de les identifier.
OpenGL utilise pour cela des entiers. Le choix des entiers peut être fait soit directement, soit à l'aide de la fonction *glGenTextures*. Cette dernière gère les identifiants de textures et fournit des entiers qui ne sont pas encore utilisés.

Spécifier la texture courante : Il s'agit de spécifier la texture (1D ou 2D) sur laquelle s'appliqueront les opérations ultérieures. Cela se fait par l'identifiant de la texture.

Spécifier le plaquage : Il s'agit de spécifier la méthode à utiliser pour plaquer la texture courante. Cela se fait à l'aide la fonction *glTexParameter*. Cette dernière permet de plaquer les

textures de différentes manières suivant la fonction de transfert choisie entre l'image destination d'une texture et son image source.

Spécifier l'image de texture : Pour spécifier l'image de texture, il est possible de définir directement l'image concernée (soit un tableau de pixels) ou bien de définir ou de calculer une mipmap (l'image à différents niveaux de résolution) à partir de cette image.

Spécifier le mode de rendu : Enfin, la dernière fonction concerne le mode de rendu à utiliser : soit la texture uniquement, soit la combinaison de la texture et d'un rendu.

IV .3 Afficher et animer des images OpenGL avec GLUT :

Les fonctions qui ont été présentées permettent de construire une image en effectuant des dessins dans une mémoire :
vidéo ou tampon.
Ces fonctions ne réalisent par contre pas la liaison entre la fenêtre graphique et la mémoire de dessin. Cela nécessite un environnement gérant les interconnexions avec le matériel. La libraire GLUT permet cela, et gère en particulier le clavier, la souris et les fenêtres graphiques. Nous présentons ici l'architecture générale d'un programme GLUT et quelques exemples démonstratifs.

IV.3.1 Architecture générale d'un programme GLUT :

Un programme GLUT est généralement constitué des parties suivantes (la liste est non-exhaustive) :
1. Une fonction d'initialisation : cette fonction permet d'initialiser différentes valeurs : couleurs, taille d'un point, etc.
2. Une fonction d'affichage : c'est la fonction appelée par GLUT pour rafraîchir la fenêtre graphique.
3. Une fonction de fenêtrage : c'est la fonction appelée lorsque la fenêtre graphique est modifiée.

4. Une fonction de gestion du clavier : c'est la fonction appelée lorsque l'utilisateur appuie sur une touche du clavier.
5. Des fonctions de gestion des événements de la souris : c'est les fonctions appelées lorsque l'état de la souris est modifié.
6. Une partie principale : dans cette partie diverses initialisations sont réalisées, l'association des fonctions précédentes aux événements correspondants est effectuée et enfin, la boucle principale est lancée.

V. Notion de graphe de scène :

Les graphes de scènes sont des modèles de descriptions de scènes complexes très utilisés en réalité virtuelle modélisés par plusieurs langages.
nous considèrerons les primitives d'OpenGL (graphes de scènes procéduraux) et l'implémentation de graphes de scènes en modèle objet.

V.1 Description d'une grue par un graphe de scène :

La grue se compose :
- D'une cabine : cube centré à l'origine dont les arêtes sont alignées sur les axes.
- d'un bras B1 articulé sur la cabine : segment aligné sur l'axe des abscisses dont une des extrémités est l'origine.
- d'un bras B2 articulé sur le bras B1 : segment aligné sur l'axe des abscisses dont une des extrémités est l'origine.

La solidarité entre les pièces mécaniques fournit un structuration implicite pour la description de la grue par un graphe de scène.

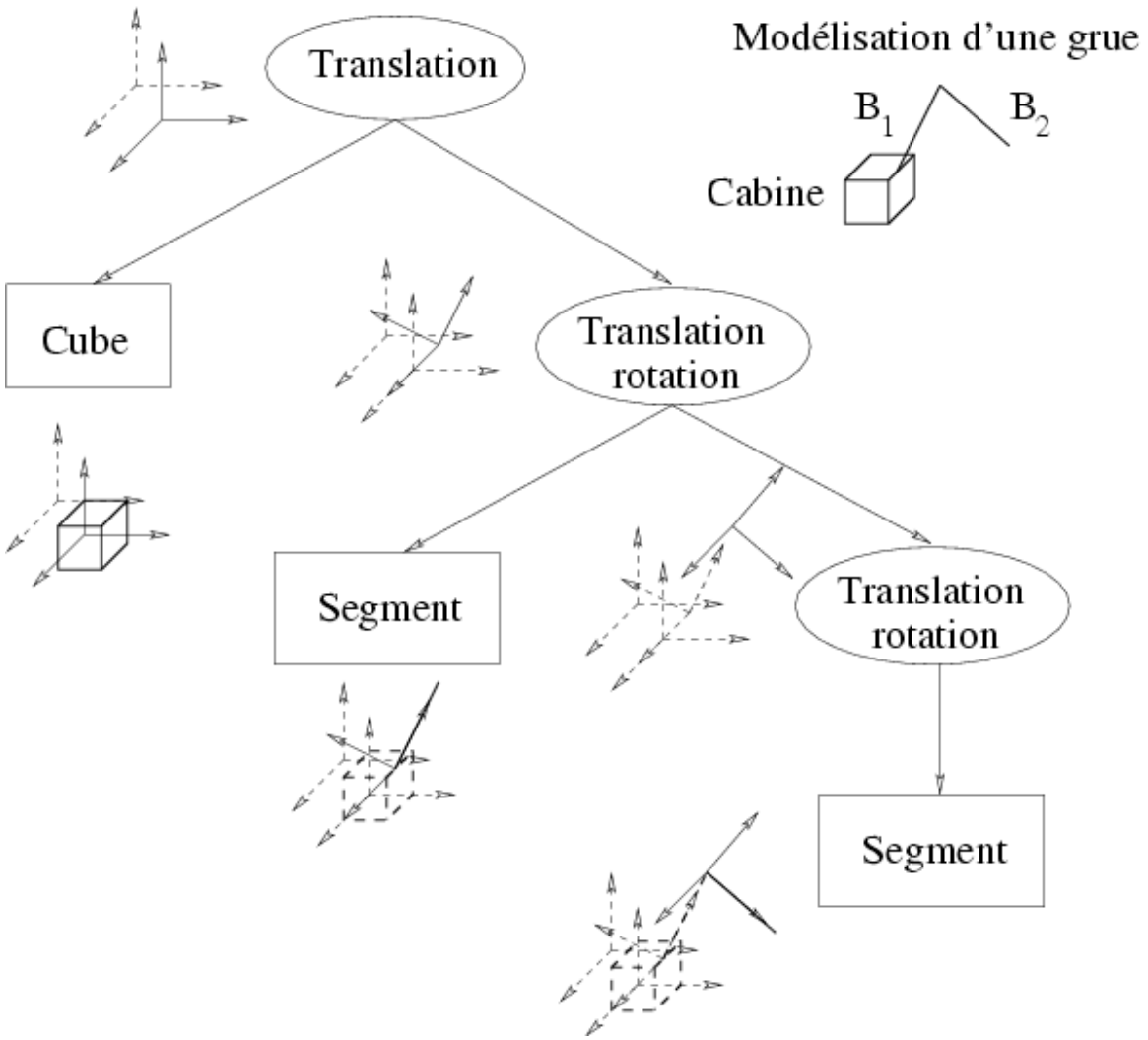


Figure 28 : Exemple de graphe de scène

V.1.1 Forme générale d'un graphe de scène :

- feuilles : objets graphiques.
- nœuds : intermédiaires des transformations.

V.1.2 Structure de graphe de scène :

Deux structures principales pour les graphes de scènes :

- arbre : chaque nœud n'a qu'un seul ancêtre.
- la transformation résultante s'obtient par composition des transformations sur le chemin menant de la racine au nœud courant .
- graphe acyclique orienté : permettent de partager des objets géométriques ou des sous-scènes sans les redécrire.
- chaque chemin de la racine au nœud courant fournit une transformation permettant de générer une des sous-scènes décrite par ce nœud.

V.1.3 Implémentation des graphes de scène :

• Implémentation procédurale :

- parcours dynamique de l'arbre de scène : appels successifs de transformations puis de primitives

Graphiques.

- descente dans l'arbre en rétablissant les conditions d'appel : instructions de sauvegarde et restauration du contexte graphique.
- Implémentation orientée objet :
- définition statique de l'arbre : structure de donnée hiérarchique avec un mécanisme d'héritage.
- analyse et parcours de la structure puis génération des images (librairie graphique) par le visualiseur.

V.2 Avantages d'une représentation hiérarchique en animation :

- Toute transformation sur une des pièces se répercute sur l'ensemble des pièces qui en sont Solidaires.
- un mouvement de la cabine, va entraîner un mouvement correspondant des deux bras
- Tous les tracés se font dans le repère canonique.
- de nombreuses librairies fournissent des primitives dont la position se définit par rapport au repère canonique.
- La structuration combine des impératifs géométriques, fonctionnels et graphiques.

VI. Conclusion :

On a constaté qu'OpenGL est une bibliothèque de fonctions pour pouvoir gérer l'affichage graphique .

En outre , Il faut savoir que OpenGL n'affiche pas directement l'image mais utilise soit les librairies de Windows soit une autre librairie liée directement à OpenGL qui s'appelle Glut, grâce à celle-ci vous obtiendrez un code 100% portable car Glut propose un tas de fonctions permettant de gérer le fenêtrage, les événements enfin tout ce que OpenGL ne peut pas gérer pour qu'il soit portable . Ainsi en utilisant OpenGL et Glut vous n'aurez qu'à changer les fichiers à inclure pour rendre votre code portable .

Il faut bien savoir que OpenGL est une librairie graphique à l'origine de nombreux jeux vidéo et souvent les plus connus comme Half-Life, Soldier of Fortune et bien d'autres .

Introduction:

But du projet:

Le but du projet est de:

- Montrer les fonctionnalités et les différentes bibliothèques d'OpenGL à travers une application réalisée en C, et se familiariser avec la programmation événementielle et graphique.

- Ainsi, nous avons créés des scènes 3D que nous pouvons manipuler à l'aide de rotations, translations, et nous y avons rajouté diverses fonctionnalités comme les lumières, la texture, les filtres, la profondeur, le reflet et l'ombre.
  Créer une Scène 3D avec possibilité de manipulation des objets et déplacement de la caméra en traitant les collisions.

Plan:
Dans ce chapitre nous allons vous parler du travail réalisé qui consiste en une modélisation d'une scène 3D avec des robots (inspirés de films).
Nous allons commencer par expliquer la construction de la scène :

- L'environnement
- La caméra

Puis nous allons détailler la construction de chaque robot :

- Les graphes de scène.
- Les animations.

1. Construction de la scène 3D :

1.1 L'environnement:

L'environnement réalisé est composé en deux parties:
la SkyBox qui est une technique couramment utilisée pour les jeux vidéos et qui consiste en un plaquage d'images (généralement d'un ciel et de reliefs), ici on a opté pour deux lunes et d'un ciel étoilé sur un cube assez grand pour contenir le reste des objets graphiques.

Avant

Droite

Arrière

Gauche

Haut

Bas

Exemple de code pour le placage d'une image :

```
makeSkyBox() {

mySkyBox = glGenLists(1);

glNewList(mySkyBox, GL_COMPILE);

  glDisable(GL_LIGHTING);
  glEnable(GL_TEXTURE_2D);
  glDepthMask(GL_FALSE);

  glColor3ub(255, 255, 255);

// faces perpendiculaires à l'axe z
// z négatif
// skyBoxTexDown correspond à l'image « Bas » à plaquer
```

```
        glBindTexture(GL_TEXTURE_2D, skyBoxTexDown);
        setTexParameters();

        //debut du placage
        // les coordonnées (1,0), (1,1) …. correspondent aux
//coordonnées d'un carré de taille 1
        glBegin(GL_QUADS);
          glTexCoord2i(1, 0); glVertex3d(-SKYBOX_SIDE,
     -SKYBOX_SIDE, -SKYBOX_SIDE);
          glTexCoord2i(1, 1); glVertex3d(SKYBOX_SIDE,
     -SKYBOX_SIDE, -SKYBOX_SIDE);
          glTexCoord2i(0, 1); glVertex3d(SKYBOX_SIDE,
     SKYBOX_SIDE, -SKYBOX_SIDE);
          glTexCoord2i(0, 0); glVertex3d(-SKYBOX_SIDE,
     SKYBOX_SIDE, -SKYBOX_SIDE);
        glEnd();

        // refaire la même chose pour les autres cotés
        glBegin(GL_QUADS);
        …
        …
```

Le sol: Cette technique consiste non pas à plaquer une image sur un carré, mais à répéter ce motif (sol métallique) tout au long de ce carré.

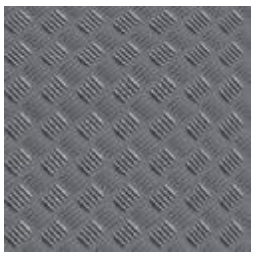


Image de la texture du sol

Code du placage du sol :

//On utilisera une boucle qui fera appel a cette fonction pour plaquer une à une la //surface du sol

```
void makeFloor(double width, double length) {

      myFloor = glGenLists(1);

      glNewList(myFloor, GL_COMPILE);

      glEnable(GL_TEXTURE_2D);

      glColor3ub(255, 255, 255);
      glBindTexture(GL_TEXTURE_2D, floorTex);
      setTexParameters();
```

```
        glBegin(GL_QUADS);
             glTexCoord2i(0, 0);
             glVertex3d(-width/2.0, -length/2.0, 0.0);
             glTexCoord2i((int)width, 0);
        glVertex3d(width/2.0, -length/2.0, 0.0);
             glTexCoord2i((int)width, (int)length);
        glVertex3d(width/2.0, length/2.0, 0.0);
             glTexCoord2i(0, (int)length);
             glVertex3d(-width/2.0, length/2.0, 0.0);
        glEnd();

        lDisable(GL_TEXTURE_2D);

glEndList();
}
```

Le building: Formé de 4 tours reliées entre elle par des passerelles, la réalisation du building est assez simple, puisqu'on se sert de 4 parallélépipède (identiques) sur lesquels on applique des textures. Les passerelles sont composer d'un demi cylindre et d'un sol sur lequel on applique aussi une texture

Texture des tours du building

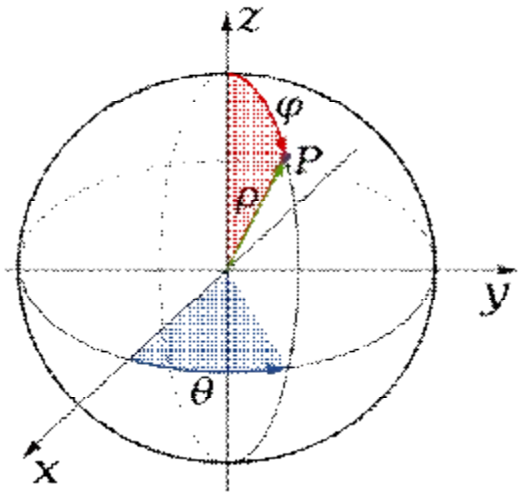
Texture des passerelles du building

1.2 La caméra

Il y deux niveau niveau de déplacement de la caméra :
Déplacement via la souris :

Le déplacement de la souris est reconnu grâce à la fonction de gestion d'événement **`glutPassiveMotionFunc`**

En utilisant les coordonnées sphériques :



Repère en coordonnées sphériques

Le curseur de la souris est toujours centré au milieu de l'écran, un mouvement de la souris engendre la modification des angles **Θ** et Φ qui se traduit en une rotation de l'angle de vue initial grâce au passage d'une transformation angulaire à une transformation linéaire:

$$\begin{cases} x &= \rho\cos\theta\sin\phi \\ y &= \rho\sin\theta\sin\phi \\ z &= \rho\cos\phi \end{cases}$$

```
// fonction de capture des mouvements de la souris
void windowPassiveMotion(int x, int y) {

    if(freeCamBool) {

        if(!(x == WINDOW_WIDTH/2 && y == WINDOW_HEIGHT/2)) { // si la souris
n'est pas centrée

            cursorCenteredBool = FALSE;
```

```
            // calcul du déplacement relatif
            x -= WINDOW_WIDTH/2;
            y -= WINDOW_HEIGHT/2;

            if((yPrev+y)*camAngularVelocityCoef > -PI/2.0 &&
(yPrev+y)*camAngularVelocityCoef < PI/2.0) // pour limiter l'angle en z
                yPrev += y;
            xPrev += x;
            // formule
            centerX = eyeX + distEyeCenter * cos(camAngularVelocityCoef * xPrev) *
cos(camAngularVelocityCoef * yPrev);
            centerY = eyeY + distEyeCenter * -sin(camAngularVelocityCoef * xPrev) *
cos(camAngularVelocityCoef * yPrev);
            centerZ = eyeZ + distEyeCenter * -sin(camAngularVelocityCoef * yPrev);
        }
        glutPostRedisplay();
    }
}
```

Déplacement avec les touches clavier UP, DOWN, LEFT RIGHT :

L'enfoncement de l'une des touches citées ci-dessus est reconnu grâce à la fonction de gestion d'événement **glutKeyboardFonc.**

le déplacement est directement traduit en une translation.

Code de déplacement de la caméra :

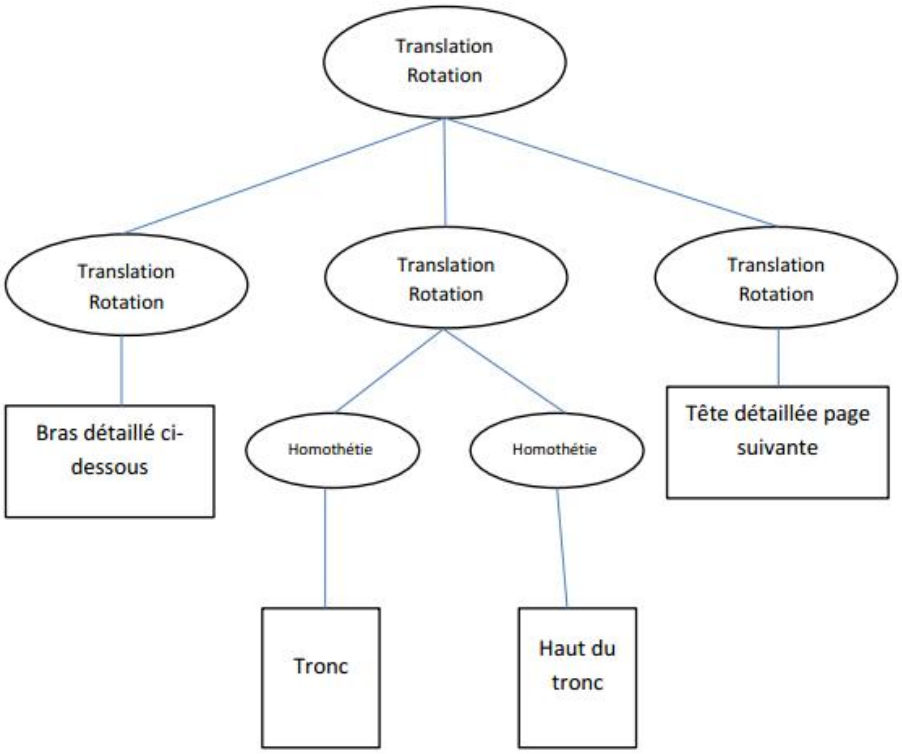
```
void windowSpecialKey(int key, int x, int y) {

    switch (key) {

        case GLUT_KEY_UP:
            if(freeCamBool) {
                camMoveForwardBool = TRUE;
                if(!camMoveBool) {
                    camMoveBool = TRUE;
                    glutTimerFunc(15, &camMove, 0);
                }
            }
            break;

        case GLUT_KEY_DOWN:
        ...
```

2. Construction des robots:

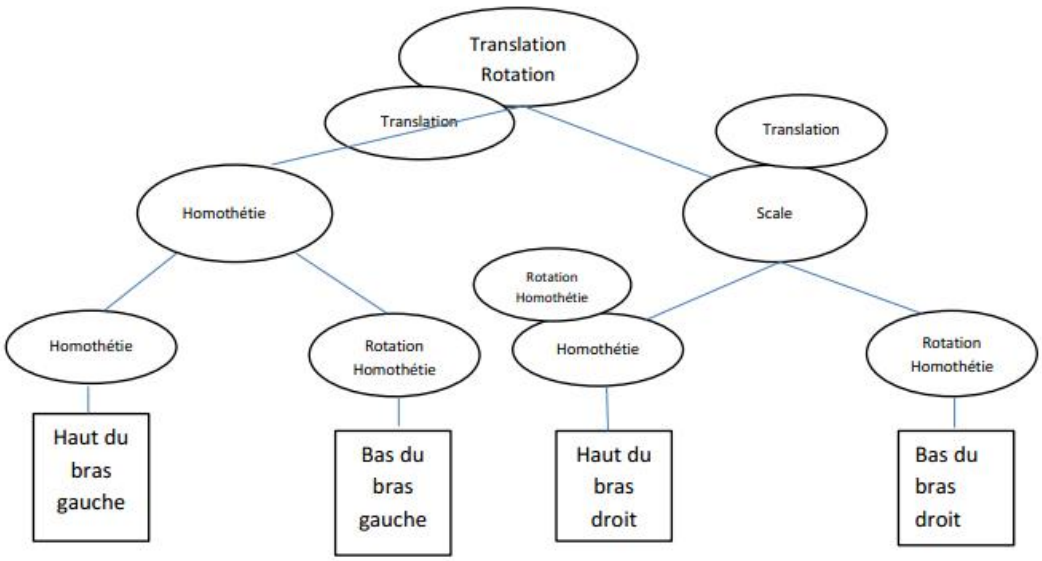
2.1 Les graphes de scènes:

Notre projet comporte plusieurs robots, cependant nous avons voulu modéliser qu'un seul Robot « Eve » vu la complexité de sa construction et de ses mouvements.

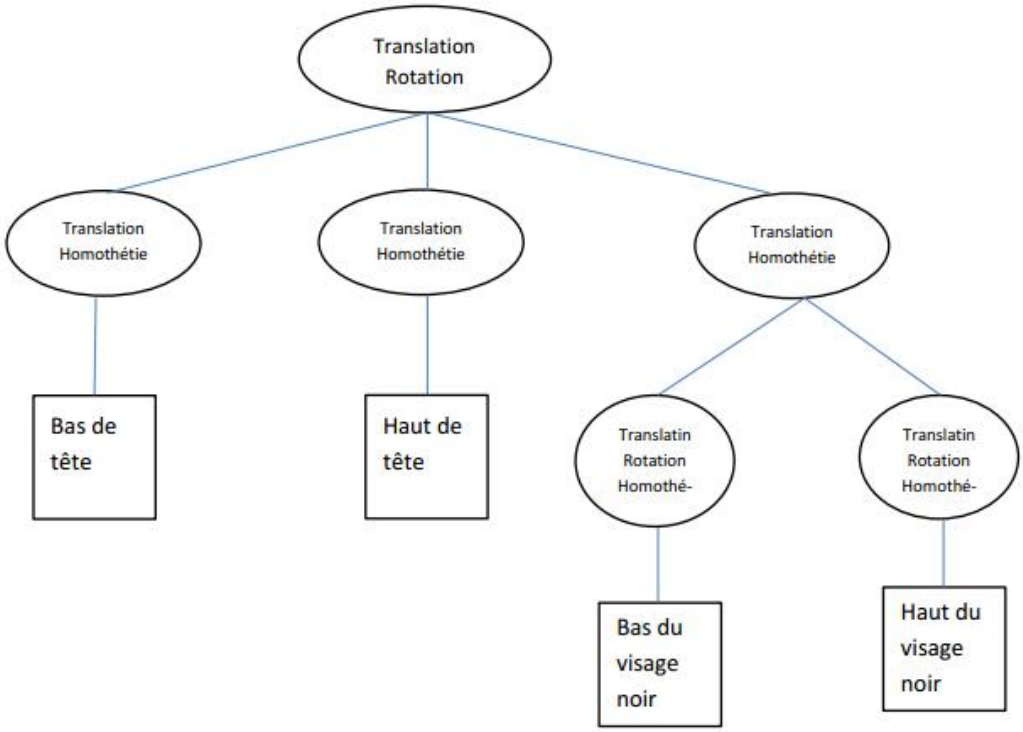
Graphe de scène du robot « Eve »



Détails du bras :



Détail de la tête :



2.2 **Les animations:**

Nous avons souhaité enrichir notre projet en réunissant plusieurs modèles de robot.
Deux types de déplacement sont utilisés suivant les robots :
- la marche pour les robots bipèdes (Humanoid, Clank) ;
- le déplacement semblable aux véhicules pour les robots qui se déplacent en roulant (Wall-E
et R2d2) et Eve.

Déplacement commun à tous les robots

Nous avons choisi un modèle unique pour tous les robots, qui est inspiré du mode de déplacement pour les jeux de courses et de voitures.
Nous avons attribué 4 degrés de liberté :

- deux translations pour avancer/reculer ;
- deux rotations pour tourner à droite/à gauche.

Ce schéma modélise le déplacement de type « véhicule ».

Déplacement des robots bipèdes

Ces robots nécessitent d'implémenter un modèle réaliste de mouvement des jambes.
Pour simuler ce mouvement lors de la marche, nous avons créé une fonction (walkHumanoid) qui répète un « motif de déplacement élémentaire ».

Cette animation fait intervenir trois rotations : à la hanche, aux genoux et aux chevilles. Elle est composée de différentes étapes. Chaque étape se lance à la fin de la précédente et déclenche une micro-animation. Le cycle se répète en boucle.

Dans un premier temps, la jambe bascule ; ensuite, le pied et le genou se plient. On recommence cette opération à chaque appel de la fonction.

L'ensemble des animations est traité par la fonction robotsMotion.
Elle prend en paramètre le numéro du robot qui a été sélectionné dans la scène.

Des fonctions annexes sont créées pour la gestion des animations automatiques pour chaque robot (wallEAnimation, ClankAnimation…).

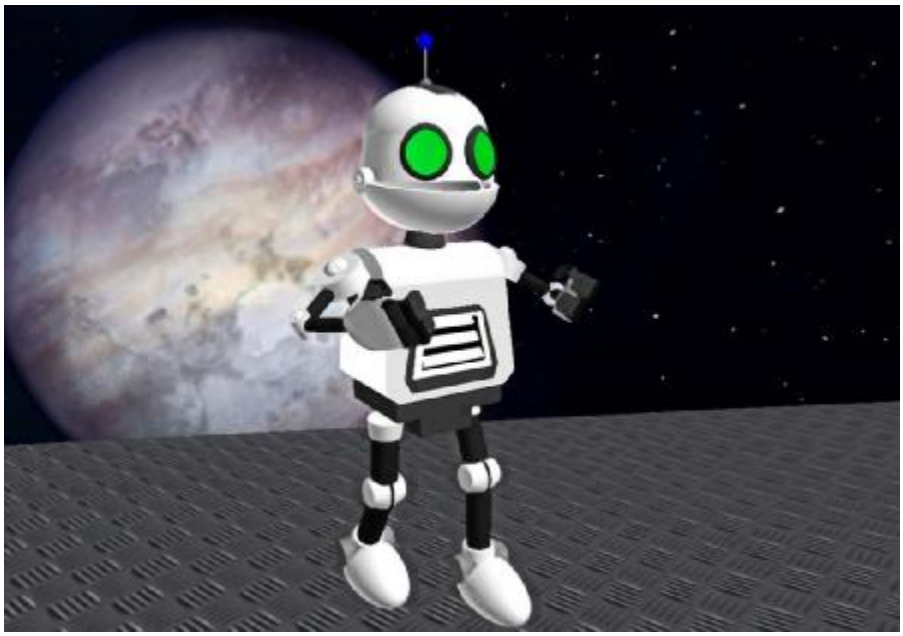
On ajoute des variables globales comme angles de rotations que l'on place à chaque articulation des robots, puis on les fait varier avec une oscillation entre deux valeurs (positive et négative) de façon à créer un mouvement de va-et-vient.

Gestion des collisions

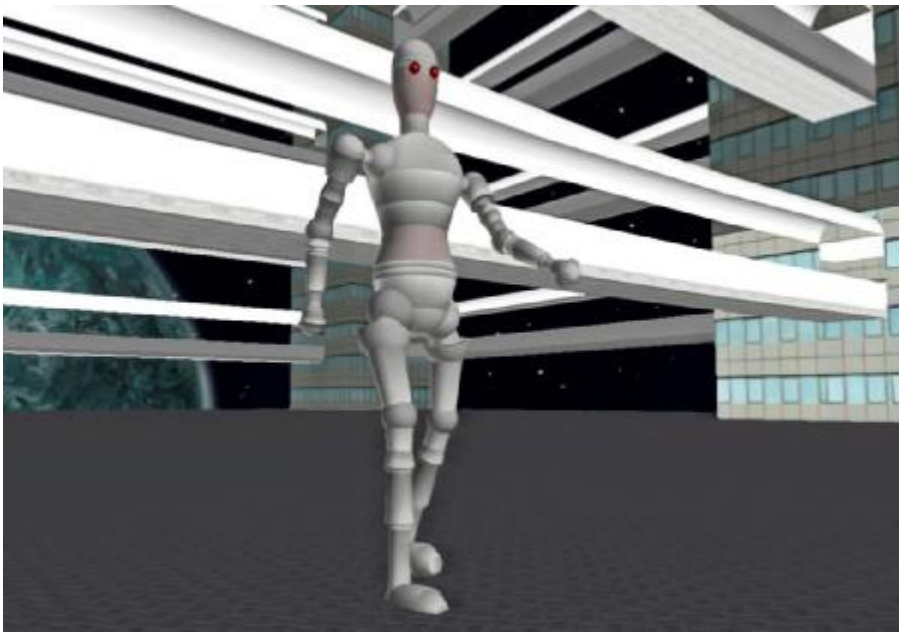
Notre sol est un carré de 100x100. Nous avons créé un tableau à 2 dimensions de 1000x1000 pour partitionner la surface. Le tableau est initialisé à 0, et on ajoute des 1 dans les cases qui correspondent à un objet (zone non accessible). À chaque pas de déplacement d'un robot, on teste la position suivante : si elle correspond à un 1 dans le tableau, on ne fait pas le déplacement.

Textures

Pour les textures, nous utilisons une fonction de la librairie SOIL qui permet de charger des fichiers de plusieurs formats (jpg, bmp, png…) en renvoyant un identifiant de texture.



Robot Clank en mouvement

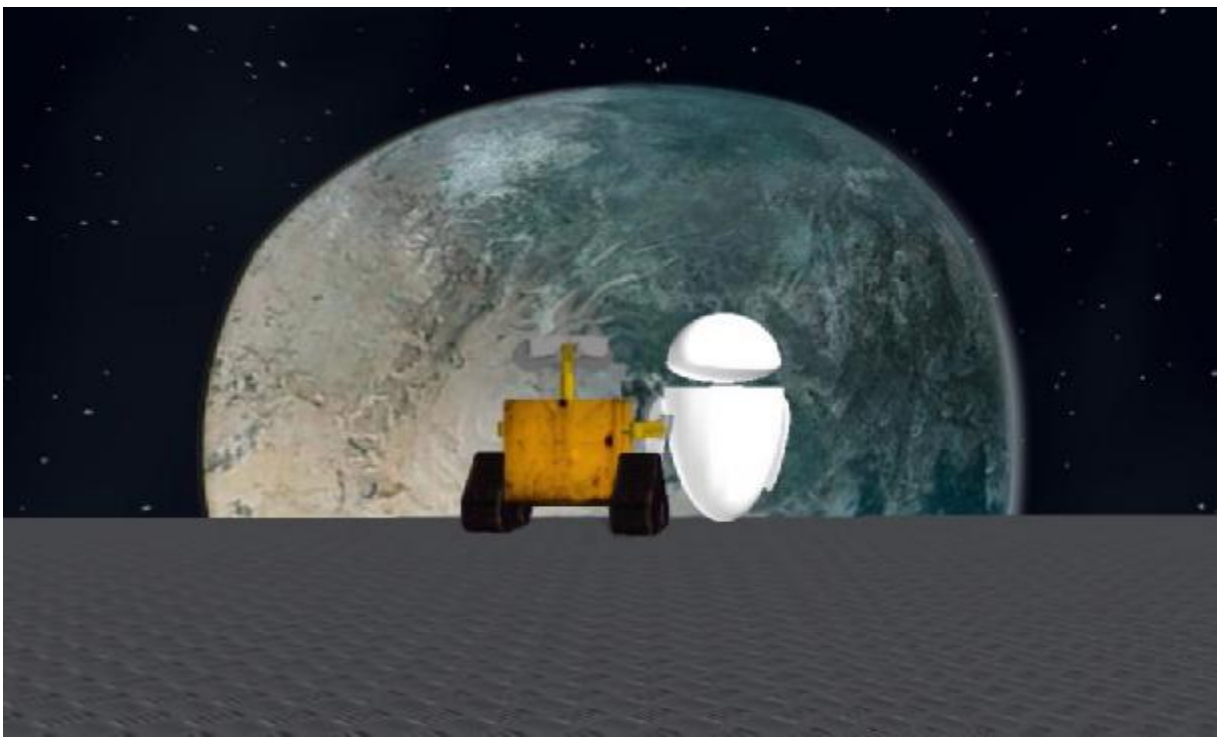


Robot humanoïde en mouvement

Notice d'utilisation

Touches	Évènements
	• Déplacer la caméra (cette option est seulement disponible en mode caméra libre).
	• Verrouiller/déverrouiller la caméra. • Cette touche associée aux touches 1, 2, 3, 4, 5 permet de basculer d'un robot à un autre. La caméra est placée derrière le robot et reste focalisée sur ce dernier.
	• Contrôler Wall-E.
	• Contrôler Eve.
	• Contrôler R2D2.
	• Contrôler Humanoid.
	• Contrôler Clank.
	• Faire avancer le Robot.
	• Orienter le Robot vers la gauche.
	• Orienter le Robot vers la droite.
	• Faire reculer le Robot.
	• Activer/Désactiver le mode animation automatique pour le robot Wall-E.
	• Activer/Désactiver le mode animation automatique pour le robot Clank.
	• Activer/Désactiver le mode animation automatique pour le robot Eve.
	• Activer/Désactiver le mode animation automatique pour le robot R2D2.

	• Activer/Désactiver le mode animation automatique pour le robot Humanoid.
ESPACE	• Sauter



Compétences acquises :

La modélisation des objets 3D: apprendre à utiliser et à créer des objets 3D comme la sphère, le cylindre, le cube et à intégrer des objets 3D complexe , ensuite faire subir ses objets divers transformations géométriques telles que la rotation, la translation et le changement d'échelle.

Les différents modes de rendus : appliquer sur nos objets 3D le mode point, le mode fil de fer, ou le mode rempli.

La création et le positionnement des lumières : introduire les différents types de lumières (ambiante, diffuse ou spéculaire) avec la possibilité de faire varier les paramètres (couleur, atténuation, ...).

Les textures et ses différents filtres : coller une texture à un objet et lui appliquer ensuite un des filtres disponibles pour permettre le lissage de la texture. Aussi La gestion de la transparence (blending): pour dessiner et prendre en compte le paramètre alpha (de transparence) de la couleur d'un objet.

Ajout d'effets spéciaux : comme le reflet, l'ombre, le brouillard… .. et d'autre tests comme : le test de profondeur, et le changement d'échelle.

Sélection et ajout d'un ensemble d'objets dans une scène 3D avec possibilité de manipuler ses objets : les déplacer, les sélectionner, les supprimer …

Déplacement dans la scène crée : déplacement de la camera avec les flèches ou la souris, en traitant la détection des collisions.

Conclusion générale :

Tout d'abord, nous avons vu que la 3D joue un rôle central dans le domaine de la visualisation. Associée avec les fonctions communicantes d'Internet (Web3D), elle devient une composante essentielle de la stratégie Internet de nombreuses entreprises. Elle est également utilisée pour des applications de type Réalité Virtuelle (temps réel), en y ajoutant la notion d'immersion (le manipulateur a l'impression d'être à l'intérieur même de l'image virtuelle), et pour des domaines d'activités variés (Architecture, marketing, formation, travail collaboratif…)

Cette technologie est celle qui a le plus évolué ces 10 dernières années, et elle connaît une révolution depuis 3 ans avec l'arrivée des shaders. Son évolution est liée à la œurse effrénée aux technologies que se livrent les grands industriels.

Dans cette optique, les programmeurs de jeux entrent en scène en exploitant les capacités des dernières cartes 3D afin de pouvoir obtenir des effets toujours plus époustouflants que ceux du dernier jeu concurrent. C'est ainsi que certains noms marquent la communauté 3D comme John Carmack qui a été le précurseur de ce que l'on appelle maintenant la vue subjective. Nous verrons que ce concept, révolutionnaire à l'époque, est sur le point de bouleverser le web et son affichage 2D.

En effet, les développements des gros industriels de la 3D s'organisent des communautés de passionnés de 3D, présents surtouts sur le vieux continent. Ce sont des démomakers, ils développent dans leurs coins des scènes cinématiques temps réel et se regroupent pour la plupart dans des concours pour s'affronter, chacun montrant leur démos.

Le cinéma se met également à faire des films en 3D, ce qui a pour effet

d'accélérer l'évolution des techniques et technologies du photo-réalisme. L'évolution de l'informatique est telle qu'il suffit d'attendre quelques mois avant que l'implémentation de ces nouvelles techniques soit envisageable dans les animations temps réel.

Enfin nous avons fini ce projet par présenter plusieurs domaines d'applications qui connaissent une véritable révolution grâce au domaine de la 3D temps réel.

Problèmes rencontrés :

La modélisation de ce projet nous apparaissait au premier abord plus simple qu'il ne l'est vraiment. Nous avons essayé de rendre la scène le plus réaliste possible mais nous nous sommes confrontés à des problèmes d'éclairages et de transparences. Le chargement de certaines textures ne se faisant pas toujours correctement, il n'était pas non plus facile de trouver les textures adaptées.
Cependant ce projet nous a permis de bien comprendre le principe de la modélisation OpenGL .

L'utilisation des textures nous est aussi plus familière et permet d'obtenir des rendus plus réalistes.

Perspectives :

Grâce à ce projet, nous pensons avoir acquis les principes de bases de la programmation graphique. Cependant, certains aspects, comme le déplacement dans une scène, conjoint à l'utilisation de la caméra sont des points que nous aurions aimé approfondir.

Bien que de nombreuses améliorations soient encore possibles, le projet est arrivé `a un résultat intéressant, qui rempli les objectifs fixés. Réaliser soi-même un sujet et innovant, puis poursuivre son projet dans toutes les différentes étapes est quelque chose

de très motivant et le travail engendré est proportionnel a cette motivation. Apr`es avoir passé un temps important sur le projet, l'arrivée `à un résultat unique est assez enthousiasmant sur le plan personnel.

Références Bibliographiques

Informatique graphique :

• E. Angel. *Interactive computer graphics, a topdown approach using OpenGL* (third edition).
Addison Wesley, 2003

• J. D. Foley, A. van Dam, S. K. Feiner & J. F.
Hughes. *Computer Graphics: Principles and Practice in C* (2nd edition). Addison Wesley, 1995

• A. Watt & M.Watt. *Advanced animation and rendering techniques: theory and practice*. ACM
Press & Addison Wesley, 1992

• *Mathématiques pour l'Informatique Graphique*. Notes du cours donné par J. Calisti en 4ème année Informatique de l'IFIPS

• *Infographie et applications.* T. Liebling et H.
Rothlisberger. Editions Masson.

OpenGL :

Le livre rouge : OpenGL Programming Guide

Le livre bleu : OpenGL Reference Manual

Programming with OpenGL: Advanced Rendering,
Copyright ©1997 by Tom McReynolds and David Blythe.
SIGGRAPH `97 Course .

http://fly.srk.fer.hr/~unreal/theredbook/

http://www.gamedev.net/download/redbook.pdf

http://www.rush3d.com/reference/opengl-bluebook-1.0/

http://www.sgi.com/software/opengl/advanced97/notes/
node84.html