الجمهورية الجزائرية الديمقراطية الشعبية

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي و البحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة أبي بكر بلقايد– تلمسان

Université Aboubakr Belkaïd-Tlemcen

كلية التكنلوجيا

Faculté de Technologie

Département de Génie Electrique et Electronique (GEE)

Filière : Electronique

MASTER INSTRUMENTATIONS

PROJET DE FIN D'ETUDES

Présenté par : **Melle BELABBES FATIMA**

Intitulé du Sujet

Conception d'un outil de visualisation en temps réel d'objets 3D déformables en utilisant les librairies Qt et OpenGL

Soutenu en 2018, devant le jury composé de :

Mr YACOUBI Boumediene	Grade MAA	Univ. Tlemcen	Président
Mr HADJ ABDELKADER Amine	Grade MCA	Univ. Tlemcen	Encadreur
Mr IRID Sidi Mohammed El Hadj	Grade MCB	Univ. Tlemcen	Examinateur
Mme *HANDOUZI Wahida*	GradeMCB	Univ. Tlemcen	Examinatrice

Année Universitaire 2017-2018

Remerciements

Ma reconnaissance et mes remerciements pour Toi seigneur, ALLAH LE TOUT PUISSANT, pour la grâce et la miséricorde que Tu m'as accordées pour finaliser ce travail.

Tous mes remerciements à Mr HADJ ABDELKADER Amine qui a bien voulu m'encadrer dans ce travail.

Je tiens vivement à exprimer ma profonde reconnaissance aux Professeurs Mr YACOUBI Boumediene, Mr IRID Sidi Mohammed El Hadj et Mme HANDOUZI Wahida qui m'ont fait l'honneur de bien vouloir juger ce travail.

Mes remerciements vont à tous ceux de près ou de loin ont contribué à ce travail.

.

Dédicaces

je dédie ce travail À tout le monde

Résumé

Avec l'accroissement de la technologie la 3d devient omniprésente dans tout ce qui nous entoure (le cinéma, l'industrie, les jeux vidéo, l'enseignement, la santé etc..)

Cette large diffusion de la 3d est accompagnée par une forte concurrence entre les sociétés pour développer des modeleurs et des moteurs de rendu correspondent au format de fichier adopté par la société pour stoker les maillages.

Parmi les formats les plus connus et aussi utilisé comme des formats de fichiers d'échanges entre logiciels on trouve le format obj cette dernière permet de stocker les informations relatives à la géométrie et aux matériaux.

Dans ce mémoire nous nous sommes intéressées à la conception d'un outil qui permet de charger des fichiers obj et de visualiser en temps réel des objets 3D.

Pour le développement de cette application on a utilisé les librairies QT , Opengl pour assurer la portabilité de l'application et le langage de programmation c++ qui est très utilisable pour le développement des applications temps réel .

Mots-clés : c++,Qt ,Qt creator,opengl,fichier obj,loader,chargeur,3D,infographie,rendu, maillage.

Abstract:

The current technological development makes the 3D objects and scenes omnipresent in our daily life, such as in cinema, industry, video games, teaching, health, …

This large spread of the 3D technology brings many challenges and a competition among companies on developing modelers and rendering engines matching the 3D files encoding available on the market and adopted by the different content creators for meshes storage.

Among the most known formats we find the obj format, widely used for content exchange between different softwares and very useful for storing the information about geometry and and materials.

In this dissertation, we were interested in the design of a visualization tool, in order to load the obj files display the corresponding 3D forms in real-time, hence taking account of their continuous deformation.

This application was developed using the standard cross-platforms Qt and Open GL libraries, in order to ensure the application transport-ability. It was indeed necessary to use the C++ language for the development since it allows to take account of the real-time constraints .

Keywords: C++, Qt, Qt Creator, Open GL, obj files, loader, 3D, rendering, mesh

ملخص :

مع التطور التكنولوجي الحاصل أصبحت تقنية الأبعاد الثلاثة متواجدة في كل ما يحيط بنا (السنما، الصناعة، ألعاب الفيديو، التعليم، و الصحة،..... إلخ)

هذا الإنتشار الواسع ترافق معه ظهور منافسة شديدة بين الشركات و ذلك لتطوير منمذجات و محركات عرض تتوافق مع صيغة الملفات المعتمدة من طرف كل شركة لتخزين النماذج الثلاثية الأبعاد.

من بين صيغ الملفات الأكثر شهرة و التي تستعمل كصيغة لتبادل المحتوى بين برامج مختلفة نجد صيغة Obj التي تسمح بتخزين المعلومات الخاصة بالهندسة و كذا المواد.

في مذكرة التخرج هاته كان اهتمامنا منصب حول تطوير برنامج يسمح بتحميل ملفات ذات الصيغة Obj وعرض الأجسام الثلاثية الأبعاد في الوقت الحقيقي.

لتطوير هذا التطبيق استخدمنا مكتبتي Opengl و Qt لضمان اشتغال البرنامج على أنظمة تشغيل مختلفة كما قمنا باستعمال لغة C++ و التي تعتبر اللغة الأكثر استعمالا لتطوير تطبيقات الزمن الحقيق.

Table des matières

CHAPITRE 03:APPLICATION

Liste des illustrations

INTRODUCTION GÉNÉRALE

Introduction générale :

L'infographie tridimensionnelle ou bien la 3D est une discipline qui ne date que d'une trentaine d'années est aujourd'hui une branche majeur de l'informatique.
La différence primordiale entre la 2D et la 3D c'est l'existence de la profondeur ou bien l'ajout d'une troisième dimension ce qu'on appelle l'axe Z qui permet une représentation des objets en perspective, cette représentation donne un effet très réaliste, proche de l'espace dans lequel nous vivons.

La 3D comme discipline se subdivise essentiellement en deux sous branches :

La modélisation de ce que l'on veut visualiser ou représenter soit en utilisant la méthode polygonale qui permet la conception de formes 3D à partir de triangles, polygones et quadrilatères. soit en utilisant La méthode surfacique qui permet la création de formes 3D à partir de courbes.

Une fois que l'étape de modélisation est terminée vient l'étape de rendu qui permet de visualiser ce que l'on a modélisé.

L'émergence de la 3d dans d'innombrables domaines tels que l'architecture, l'industrie, les jeux vidéo, la présentation des phénomènes médicaux donne naissance à un ensemble de logiciel de modélisation et de visualisation comme Blender et Maya etc.. ,dans ce projet de fin d'étude nous avons développé notre propre chargeur (loader) en utilisant les bibliothèque Qt et Opengl, ce loader permet le chargement des fichier obj et la visualisation des objet 3d en temps réel.

Cet outil doit être facile d'utilisation, à faible consommation de ressources (mémoire, CPU, communication, etc.) et transportable sur de multiples plateformes, afin d'être utile à un maximum d'applications faisant intervenir des objets 3D continuellement déformables et dont la manipulation pourra être gérée par d'autres programmes parallèles. Notre application servira ainsi comme un outil de visualisation basique permettant de suivre l'évolution des formes 3D dans le temps et sera utile à des fins d'analyse et de débogage. L'application principale est la visualisation d'organes humains sur un simulateur chirurgical ou une plateforme d'apprentissage de l'anatomie humaine.

Pour décrire l’implémentation de cet outil de visualisation nous présentons un manuscrit structuré en trois chapitres.

Le premier chapitre présente les notions nécessaires à la compréhension de la 3d, le second chapitre et entièrement consacré à l’explication des outils utilisés lors de l’étape de développement, En chapitre 3 on a expliqué en détails les différentes étapes de conception de l’outil de visualisation partant de la préparation de l’environnement jusqu'à la phase d’exécution

En fin on termine le manuscrit par une conclusion générale et des perspectives

CHAPITRE 1 :
GÉNÉRALITÉ SUR LES MODÈLES 3D

1.1. Introduction :

L'infographie tridimensionnelle ou bien la 3d est omniprésente dans tout ce qui nous entoure (la médecine, les jeux vidéo, l'industrie….), Cette discipline a changé la face de l'informatique en lui ajoutant une troisième dimension ce qui permet de modéliser et visualiser avec plus de réalisme soit le monde virtuel ou le monde réel.
Dans ce chapitre on va essayer de mettre en évidence les concepts les plus utilisables dans ce domaine ainsi de parler de l'étape de modélisation où on montre de quoi se compose un objet et les différents types d'un maillage puis on essaye d'expliquer les principes mathématiques qui permettent de manipuler un objet 3d dans l'espace.

La deuxième partie consiste à mettre en lumière le processus de rendu, et on termine le chapitre par une explication détaillée du contenu du fichier obj.

1.2. Qu'est ce que l'infographie 3D ? [1]

3D (l'abrégé de L'infographie tridimensionnelle) est une branche de l'informatique qui s'intéresse à la création d'images numériques de synthèse. Elle est basée sur un espace virtuel, défini par trois axes orthogonaux X, Y et Z, la 3D rassemble les techniques issues de la CAO (conception assistée par l'ordinateur) qui Permettent la représentation d'objets en perspective sur un moniteur d'ordinateur.
Les images obtenues en utilisant l'infographie 3D peuvent facilement être distinguées des images obtenues à l'aide d'autres moyens (dessin, infographie 2D), par un niveau de réalisme élevé dans la reproduction des ombres, de l'éclairage, des reflets, des matériaux etc…..La 3D se décompose essentiellement en deux étapes :

- Modéliser ce que l'on veut visualiser ou représenter. Cette étape est appelée **modélisation.**
- Effectuer la visualisation de ce que l'on a modélisé. Cette étape est appelée **rendu.**

1.3. La modélisation des objets 3d : [3]

Dans l'univers de l'informatique graphique 3D, la modélisation 3D est la première étape de l'infographie 3D. Il s'agit d'une technique de synthèse d'images 3D qui consiste à modéliser, c'est-à-dire à représenter en trois dimensions les éléments que l'on souhaite visualiser.

Il existe deux grandes techniques de modélisation : **la polygonale** et **la surfacique**.
La première permet la conception de formes 3D à partir de triangles, polygones et quadrilatères. La seconde permet quant à elle la création de formes 3D à partir de courbes.
La modélisation 3D repose essentiellement sur des lois mathématiques complexes, qui sont appliquées à des contextes précis de la physique et en particulier à l'optique.

1.3.1. Définition d'un maillage : [4] [6]

Un maillage est le nom donné à tout objet modélisé dans un espace tridimensionnel, on peut le définir aussi comme une partition de l'espace ou d'un domaine en cellules appelées éléments. Ou bien un ensemble de points sommets reliés entre eux par des arêtes.
Les éléments qui forment un maillage sont souvent des triangles ou des quadrilatères en deux dimensions, et des tétraèdres, des cubes ou des hexaèdres en trois dimensions (voire figure 1.1).

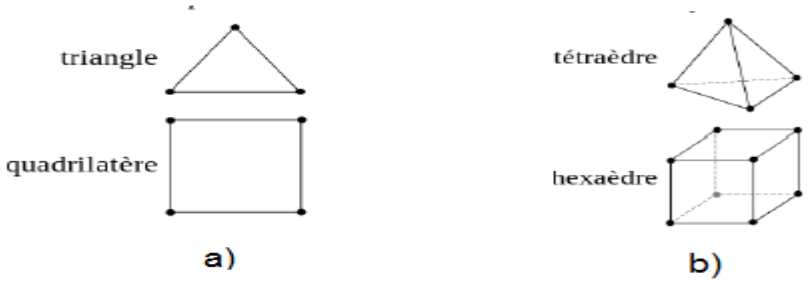


Figure 1.1:Éléments d'un maillage 2D(a) et 3D (b)

1.3.2 Composition d'un maillage :

Tout un maillage de base est constitué de trois structures principales (voire figure 1.2) :

- **Un vertice (sommet) :** les vertices pluriel du vertex, ou encore sommet sont la base du rendu 3D polygonal, on peut les considérer comme des points 3D ou bien une unité à laquelle l'ordinateur associe des informations sur la position (p(x ; y ; z)), la couleur, etc.
- **Une arête (edge)** *:* C'est un segment qui relie deux vertices.

- **Une face :** est un polygone reliant 3 vertices ou plus. Chaque face est entourée d'edges refermées et a obligatoirement le même nombre de vertices et d'edges, à chaque face est associée un vecteur normal perpendiculaire au plan auquel appartient la face dont le but de spécifier l'orientation de cette dernière.

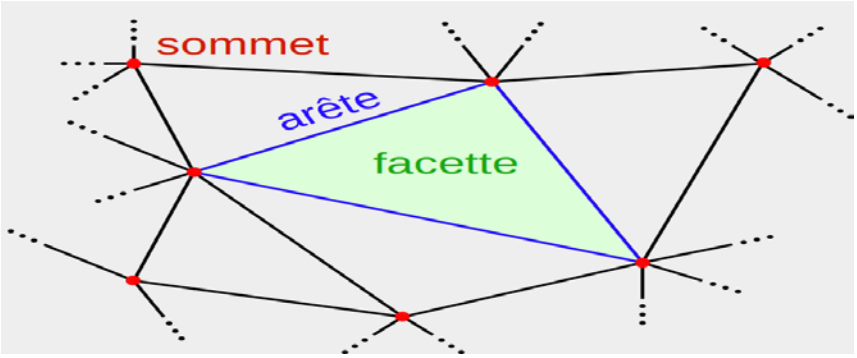


Figure 1.2:composition d'un maillage

1.3.3.Les types d'un maillage :

- **Maillage triangulaire et quadrangulaire :** [5]

On parle d'un maillage triangulaire si tous les polyèdres qu'il forment sont des triangles.

Si tous les polyèdres sont des quadrangles, le maillage est dit quadrangulaire (voir la figure 1.3).

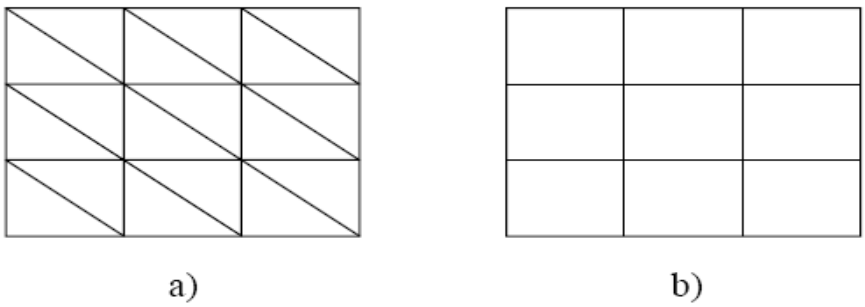
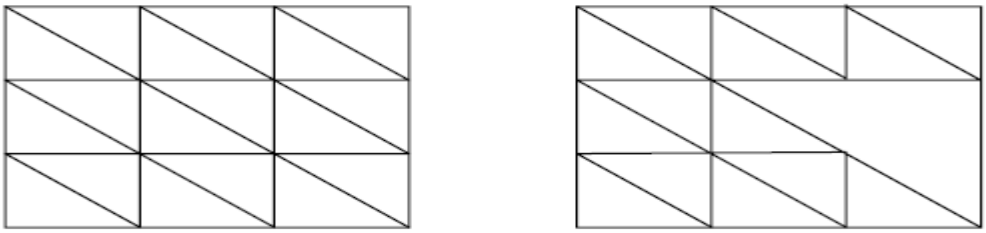


Figure 1.3:maillages 2D triangulaires (a) et quadrangulaires (b)

- **Maillage conforme et non conforme**

-Un maillage est dit conforme (le terme anglais est ‘manifold’) si aucun des sommets qui le composent n’est situé sur une arête du même maillage (voir figure 1. 4)



a) maillage conforme b) maillage non conforme

Figure 1.4:conformité des maillages

- **Maillage structuré :**[7][8]

Le type de classement le plus fréquemment utilisé est basé sur la propriété structuré ou non structuré des maillages qui se rapporte à la nature de la connectivité entre les éléments.

Dans un maillage structuré, chaque nœud est entouré d'exactement du même nombre de nœuds (voire figure 1.5).

Le problème des maillages structures est qu'ils ne permettent pas de mailler en un seul bloc des géométries complexes.

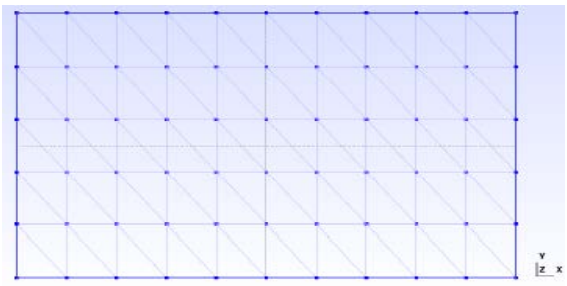


Figure 1.5:maillage triangulaire structuré

- **le maillage non structuré** [9]**:**

  Un maillage non-structuré est constitué d'un ensemble de Cellules (faces) qui représentent les même géométries souvent triangulaires en deux dimensions et tétraèdres en trois dimensions mais sont de dimensions différentes (voire la figure 1.6).

Les maillages non structurés peuvent être générés par les méthodes suivantes : algorithme de subdivision quadtree-octree, triangulation de Delaunay et méthodes d'avances de front

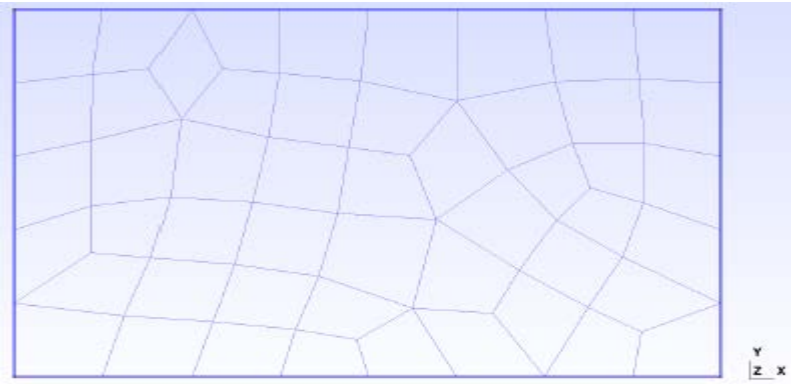


Figure 1.6:Maillage quadrangle non structuré

- **Maillages hybrides :**

  Des maillages hybrides, peu développés à l'heure actuelle, sont susceptibles de connaître un essor. Ils consistent à définir plusieurs maillages réguliers reliés entre eux par un maillage irrégulier .les maillages hybrides sont très complexes, et difficiles à mettre en œuvre.

1.4. Manipulation des objets 3D :

Modéliser un maillage (objet) sans le pouvoir manipuler (changer sa forme et sa taille, le faire bouger, pivoter) n'a aucun sens dans le domaine de l'infographie tridimensionnelle, pour cette raison on va aborder quelques principes mathématiques qui permettent un ensemble de transformation sur un maillage.

1.4.1. Les transformation sur un plan (2D) : [11]

On premier temps on va étudier les différentes transformations sur un plan en utilisant les coordonnées cartésiennes pour faciliter la compréhension.

1.4.1.1. Translation :

Une translation est simplement l'addition de deux vecteurs, un des vecteurs est le vecteur translaté et l'autre est le vecteur de translation. Une translation lorsqu'elle est appliquée à tous les points d'une forme permet de déplacer la forme sans la modifier (voire figure1. 7).

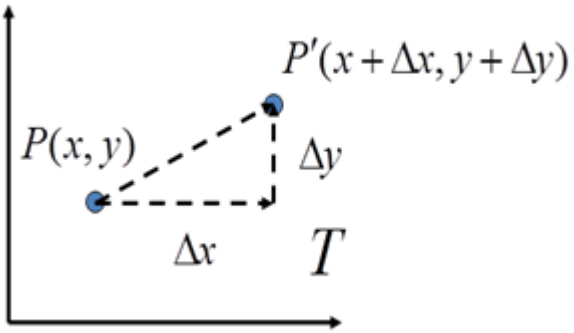


Figure 1.7:la translation

Soit P(x, y) un point du plan, T(dx, dy) le vecteur de translation. Le point P'(x', y') transformé du point P par la translation de vecteur T est défini par :

$x' = x + \Delta x$ $y' = y + \Delta y$.

Autrement dit : P' = P + T

1.4.1.2. L'homothétie :

L'homothétie ou bien Le changement d'échelle (agrandissements ou rétrécissements) consiste à associer à un point P(x ,y), un point P'(x' ,y') (voire figure 1. 8) tel que :

x'= x × sx ; y'= y × sy

Autrement dit : P' = S.P ie $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

sx , sy sont appelés des scalaires. Ce sont des réels qui contrôlent l'agrandissement ou le rétrécissement de l'objet

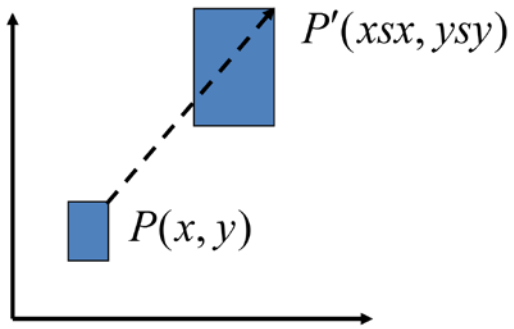


Figure 1.8:L'homothétie

1.4.1.3 La rotation :

La rotation est l'une des manipulations de base en infographie 3D. Elle consiste à faire pivoter un objet (maillage) dans l'espace, autour d'un axe donné.

On considère pour l'instant les rotations par rapport à l'origine (voire figure 1.9)

.

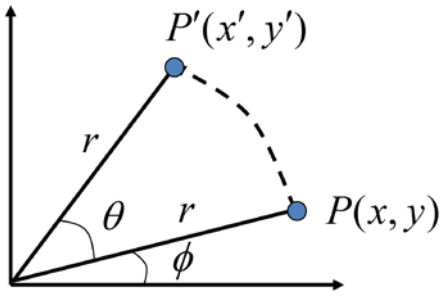


Figure 1.9:la rotation

$$P:\begin{cases} x = r\cos\phi \\ y = r\sin\phi \end{cases}$$

$$P':\begin{cases} x' = r\cos(\phi+\theta) \\ \quad = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ \quad = x\cos\theta - y\sin\theta \\ y' = r\sin(\phi+\theta) \\ \quad = r\cos\phi\sin\theta + r\sin\phi\cos\theta \\ \quad = x\sin\theta + y\cos\theta \end{cases}$$

Donc $P' = RP$ ie : $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

1.4.2. Les systèmes de coordonnées homogènes :

Contrairement à la rotation (P' = R.P) et au changement d'échelle (P' = S.P) , la translation en coordonnées cartésiennes ne peut pas être effectuée à l'aide d'une multiplication matricielle(P' = T+P) .La solution sera de remplacer les coordonnées cartésiennes par des coordonnées homogènes.

L'utilisation des coordonnées homogène consiste a ajouté une quatrième coordonnée w à chaque point p(x,y,z). Donc un point est représenté par le quadruplet (x, y, z, w).

Deux quadruplets peuvent représenter le même point si l'un est un multiple de l'autre, en autant que ce point n'est pas l'origine.

Par ex., (x, y, z, w) et (x/λ, y/λ, z/λ, w/λ) représentent le même point et si λ= w ≠0, (x/w, y/w, z/w) désignent les coordonnées cartésiennes du point homogène. Ces quadruplets de points représentent des points dans l'espace à 4 dimensions.

Toutefois, notre but est de représenter des points 3D. Pour ce faire, nous allons normaliser ces points homogènes 4D; ils auront la forme suivante: (x, y, z, 1).

1.4.3 .Transformation d'un objet dans l'espace (3D) en utilisant les coordonnées homogènes: [10]

1.4.3.1. Translation 3D :

L'utilisation des coordonnées homogènes permet de représenter la translation sous la forme d'une multiplication matricielle.

La forme générale de translation est :

$$P' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T*P$$

1.4.3.2. L'homothétie dans un espace 3D :

Un changement d'échelle 3D modifie la taille d'un objet par rapport à l'origine.

L'opération de changement d'échelle a la forme suivante :

$$P' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = S*P$$

1.4.3.3. La Rotation dans un espace 3D :

En 3D, une rotation fait tourner un ensemble de points (ou d'objets) d'un angle θ autour d'un axe de rotation.

L'axe autour du quel s'effectue la rotation 3D peut être une droite quelconque dans l'espace. Il sera alors nécessaire de décomposer la rotation en trois composantes : des rotations en x, y et z, ce qui donnera lieu à trois matrices de rotation différentes :

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rz(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

-Forme générale de rotation (rotation autour de l'axe x) :

$$P' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = Rx * P$$

1.5.le rendu 3D :

C'est une fois que l'étape de modélisation 3D est réalisée que vient le moment de créer le rendu, ce processus permet de passer d'une scène modélisée en 3d à une image 2D où en associant à chaque pixel une couleur et une intensité lumineuse.

Le processus de rendu fait appel à différentes techniques de projection et d'illumination plus au moins élaborées dont le but d'avoir une scène réaliste.

1.5.1. La projection :

La projection d'une scène 3d sur un écran nécessite de définir :

- Un volume de visualisation dans l'espace réel où ont été décrits les différents objets constituant la scène, le volume de visualisation est limité par un plan de coupage avant et un plant de coupage arrière (voire la figure 1.10) .
- La projection sur le plan de projection : il existe différents types de projection selon l'objectif fixé, les plus utilisé sont la projection perspective où le volume

de visualisation est un tronc de pyramide et la projection parallèle où le volume de visualisation est un prisme.

- Une zone d'affichage sur le périphérique considéré : le résultat de projection est affiché sur le périphérique d'affichage.

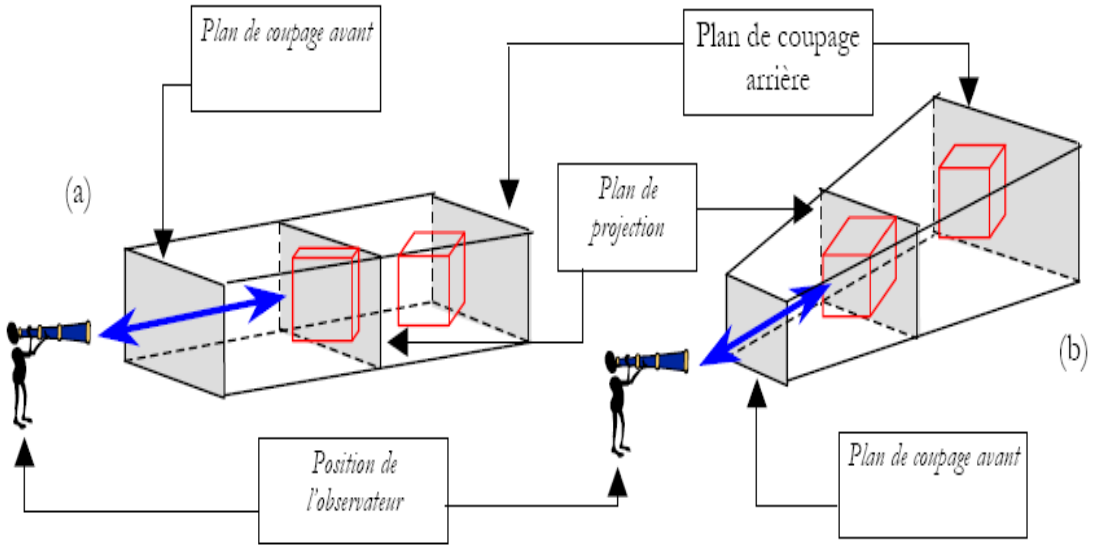


Figure 1.10:définition de volume de visualisation et plan de projection

(a)projection parallèle, (b) projection perspective

1.5.1.1 .projection parallèle : [16][17]

Dans une projection "parallèle" les projecteurs sont parallèles entre eux et le point de fuite placée à l'infini (voire la figure 1.11(a)).

Les lignes parallèles en 3D demeurent parallèles après la projection donc ce type de projection est moins réaliste mais il fournit des mesures exactes.

Pour obtenir le projeté d'un point M de coordonnées (x ,y ,z) (voire la figure 1.11 (b) , il suffit de négliger la coordonnée z : Le projeté de M notée Mp aura pour coordonnées :

xp=x, yp=y , zp=0.

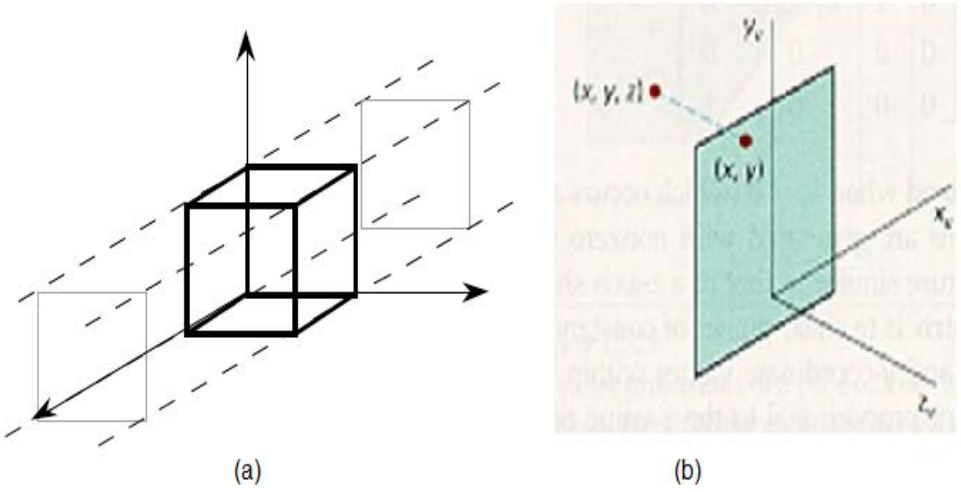


Figure 1.11:la projection parallèle

1.5.1.2. Projection perspective :

La projection en perspective se rapproche du système visuel humain. Cela permet d'atteindre un certain niveau de réalisme.

L'effet visuel d'une projection en perspective est de diminuer la taille de formes selon leur distance vers le point de fuite.

Ceci peut être décrit par une matrice 4 x 4 en coordonnées homogènes.

Pour une point de fuite sur l'axe z à position –d.

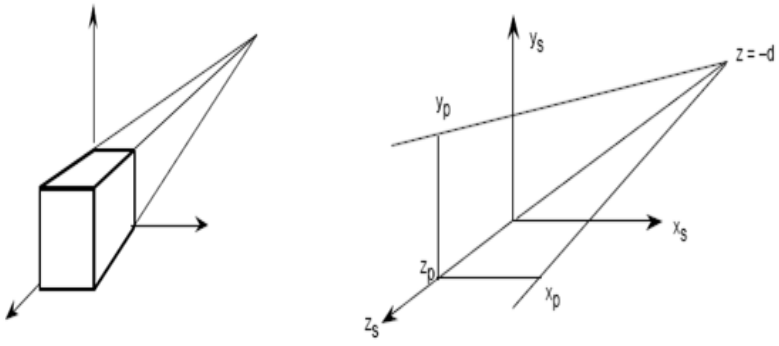


Figure 1.12:la projection perspective

$$\frac{xp}{d+zp} = \frac{xs}{d} \Rightarrow xp = \left(\frac{d+zp}{d}\right) xs = \left(1 + \frac{zp}{d}\right) xs \qquad , \quad \frac{xp}{w} = xs \Rightarrow w = 1 + \frac{zp}{d}$$

$$\begin{bmatrix} xp/w \\ yp/w \\ zp \\ w \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{bmatrix} xs \\ ys \\ zs \\ 1 \end{bmatrix}$$

1.5.2. l'Illumination : [18][19][20]

L'illumination est un processus d'intérêt majeur Pour augmenter le réalisme du rendu, il modélise le parcours des rayons de lumière quittant une source lumineuse jusqu'à la surface d'un objet pour déterminer son apparence visuelle (couleur, profondeur, ….)

On distingue deux types d'illuminations:

Illumination locale:

Considère l'apport direct des sources de lumière sur les objets de la scène

Illumination globale:

Considère non seulement l'apport direct des sources de lumière mais aussi celui de lumières inter-réfléchies par les autres objets de la scène.

1.5.3 .Les modèles d'illuminations :

En tenant compte seulement les interactions entre le surface d'objet et les sources lumineuses sans prendre en considération les interactions entre objets , trois modèles d'illumination simple peuvent être distingués .

1.5.3.1. Lumière ambiante :

C'est le modèle le plus simple il consiste à prendre en compte une source lumineuse non Ponctuelle d'intensité **Ia** qui émet de manière constante dans toutes les directions et sur toutes les surfaces, pour un point P de la surface, l'intensité lumineuse est : **Ip = Ka × Ia**; qui est constante où **Ka** est un facteur qui détermine la quantité de lumière ambiante réfléchie par la surface et est fonction des propriétés matérielles de la surface ($0 \leq Ka \leq 1$)

1.5.3.2. Réflexion diffuse :

La réflexion diffuse est du à une source lumineuse ponctuelle qui émet de manière constante dans toutes les directions. Dans ce type de réflexion le rayon incident se réfléchie dans un grand nombre de directions et son énergie est redistribuée dans une multitude de rayons réfléchis.

L'intensité en un point de surface ne dépend que de l'angle entre la normale à la surface et la direction du point de la source lumineuse, donc quelque soit la direction de l'observateur il va recevoir la même quantité de lumière, car la radiance sortante est la même pour toutes les directions.

Ce type de réflexion régit par la loi de Lambert :

$$IP = kd \; I_L \cos\theta$$

$$= kd \; I_L \; (Np \; . \; Lp)$$

Donc Plus l'angle construit entre la normale et la direction de la source de lumière incidente est petit, plus l'intensité de la lumière est forte.

I_L : est l'intensité de la source lumineuse ponctuelle

kd : le coefficient de réflexion diffuse de la surface ($0 \leq kd \leq 1$).

Np : la normale à la surface au point P.

Lp : la direction du point P à la source lumineuse.

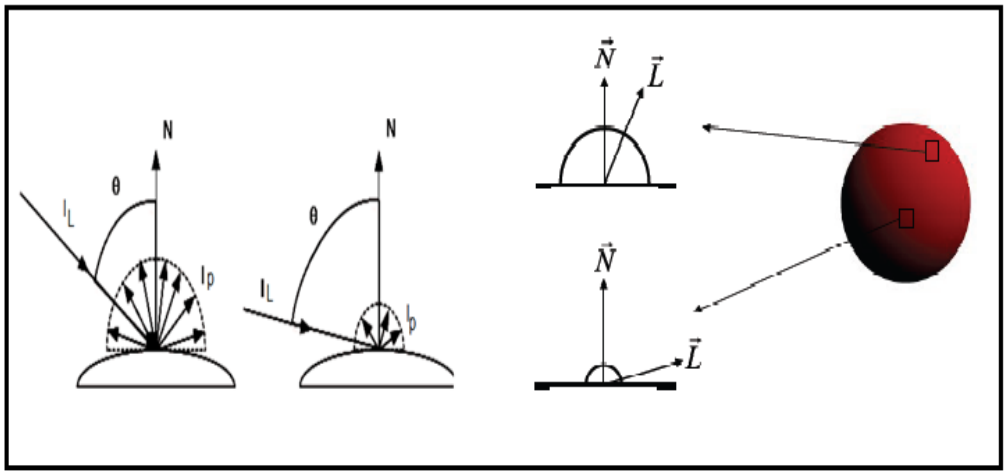


Figure 1.13:le modèle de diffusion

1.5.3.3. Le modèle spéculaire :

La réflexion purement spéculaire est celle engendrée par un miroir ou toute surface parfaitement lisse (métaux, verres, plastiques transparents, liquides transparents) pour ce type de réflexion le rayon incident donne naissance à un rayon réfléchi unique , Les deux rayons sont symétrique par rapport à la normale (voir la figure 1.14 (a)) ,Dans le cas idéal l'énergie du rayon incident se retrouve totalement dans le rayon réfléchi, en pratique une partie de l'énergie peut être absorbée, diffusée ou réfractée au niveau de l'interface.

La réflexion spéculaire apparaît sur des surfaces brillantes sous la forme d'une région de haute intensité que l'on appelle highlight (voir figure 1.14 (b))

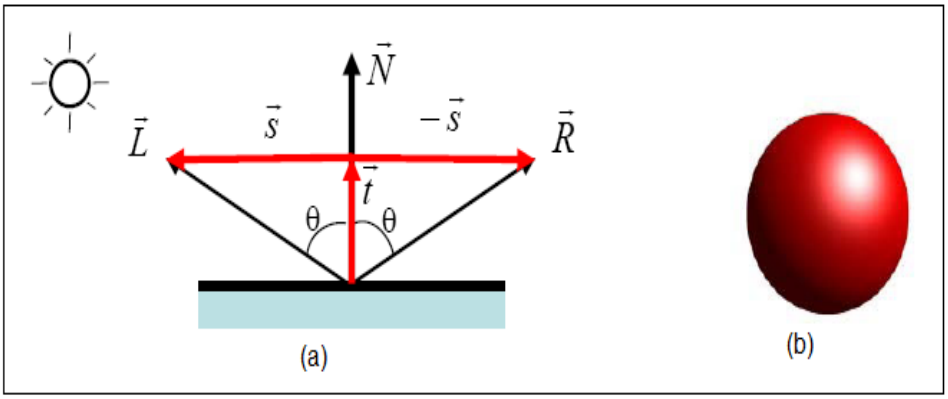


Figure 1.14:réflexion spéculaire pour un miroir parfait

1.5.4. Méthode du lancer de rayons :

.

Le lancé de rayon (ou ray-tracing) est la méthode de rendu la plus simple pour générer une image réaliste elle utilise les lois de l'optique géométrique. La méthode consiste à suivre le trajet inverse (à partir de l'observateur vers les sources de lumière Pour éviter de traiter tous les rayons inutiles issus d'une source lumineuse) des rayons lumineux afin de calculer les propriétés géométriques et lumineuses de la scène.

L'algorithme de ray-tracing a pour but de déterminer l'image pixel par pixel en lançant un rayon du centre de projection vers la scène `à travers chaque pixel comme illustre sur la figure 1.15 Si le rayon coupe un objet de la scène, on calcule la couleur du pixel d'après les sources lumineuses et les propriétés du matériau associé à l'objet en utilisant

un des modèles de réflexion, Si le rayon ne coupe aucun objet alors le pixel prend la couleur de fond (couleur par défaut, texture...).

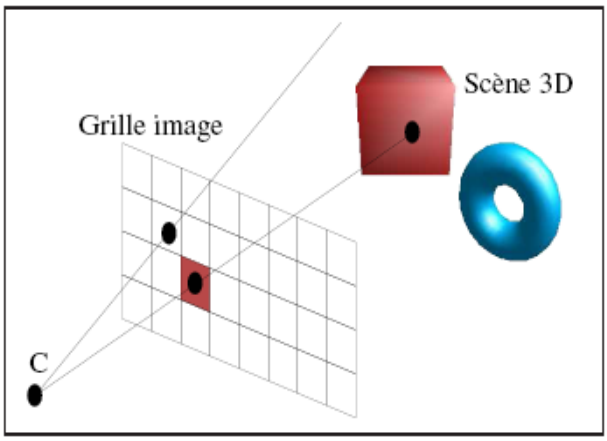


Figure 1.15:Le lancer de rayons

1.6. Le format «obj »:

Il existe aujourd'hui de nombreux formats de fichiers pour stocker les différentes données relatives à un maillage 3D cette diversité est principalement due au fait que chaque logiciel possède ses propres formats selon la manière qu'il a utilisé pendant l'opération de stockage, quelques formats ont réussi à s'imposer et à devenir des références, pouvant ainsi être utilisés avec de nombreux logiciels de modélisation et de visualisation.

Parmi les formats les plus courants et aussi utilisés comme des formats de fichiers d'échanges entre logiciels on trouve le format stl , X3D, COLLADA et le format obj,c'est cet dernier qu'on va étudier en détail dans ce qui suit en raison que le chargeur (loader) qu'on va développer dans ce projet de fin d'étude se base sur un fichier .obj pour visualiser un objet 3d

1.6.1. Quel format 3D pour quelle finalité ? [12]

1.6.1.1. Le format .X3D (extensible 3D) :

Le format X3D est un format ouvert de fichier 3D basé sur le XML, créé par le consortium Web3D dans le but de supplanter le VRML (Virtuality Reality Modeling Language).

Le format X3D reprend l'ensemble des spécificités du format VRML (couleur, texture, transparence, etc.) mais a été repensé de manière a être adapté au web et notamment dans le but de pouvoir être lancé depuis un navigateur grâce notamment à OpenGL.

1.6.1.2. Le format .DAE (collaborative design activity, OU COLLADA) :

Le format COLLADA est un format ouvert basé sur le XML et très utilisé pour l'échange de données entre différents logiciels. Le format COLLADA enregistre les informations relatives non seulement à la géométrie, les matériaux, la texture ou encore l'animation, mais il est également l'un des seuls formats qui prend en compte la cinématique et la physique des modèles.

1.6.1.3. Le format STL :

Le format de fichier STL est un format utilisé dans les logiciels de stéréolithographie (STL pour **ST**ereo-**L**ithography). Ce format a été développé par la société 3D Systems. Il est largement utilisé pour faire du prototypage rapide et de la fabrication assistée par ordinateur. Le format de fichier STL ne décrit que la géométrie de surface d'un objet en 3 dimensions.

1.6.1.4. Le format .obj : [2]

L'extension de fichier OBJ est connue en tant que fichier d'objet Wavefront 3D qui a été développé par la société Wavefront Technologies. Structuré en ASCII, il contient la description des données géométriques d'un modèle 3D (sommets, coordonnées de texture, couleur). Le format OBJ ne peut cependant pas supporter toutes les informations comme celles relatives à l'animation du modèle 3D. Le .obj est donc privilégié pour un travail en local ou pour des exportations d'images non animées.

Avec le développement de l'impression 3D, le format OBJ connaît un regain d'énergie notamment parce qu'il prend en compte l'affichage des couleurs, ce que le format STL, principalement utilisé dans ce domaine, ne permet pas.

Les fichiers au format OBJ peuvent être ouverts avec Autodesk Maya 2013, Blender, et MeshLab dans Microsoft Windows, Mac OS et Linux.

1.7. La structure du format OBJ: [13][14]

L'encodage avec Le format obj est un peut particulier et peut prendre différentes formes selon l'objet encodé (couleur, texture,..) et la qualité de visualisation qu'on veut atteindre.

Le format obj se divise en deux fichiers : **un fichier .OBJ** qui donne toutes les informations sur la géométrie (les sommets et les faces), et **un fichier .mtl** qui contient les données sur les matériaux utilisés.

1.7.1. Structure du fichier obj :

Le fichier obj représente les informations ligne par ligne , On peut le décomposer de cette manière (voir figure 1.16) :

- Ligne de commentaire
- Indication du fichier .MTL
- Définition des sommets
- Attribution des faces.
- Regroupement

```
# Blender3D v249 OBJ File: untitled.blend
# www.blender3d.org
mtllib cube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 0.748573 0.750412
vt 0.749279 0.501284
vt 0.999110 0.501077
vt 0.999455 0.750380
vt 0.250471 0.500702
vt 0.249682 0.749677
vt 0.001085 0.750380
vt 0.001517 0.499994
vt 0.499422 0.500239
vt 0.500149 0.750166
vt 0.748355 0.998230
vt 0.500193 0.998728
vt 0.498993 0.250415
vt 0.748953 0.250920
vn 0.000000 0.000000 -1.000000
vn -1.000000 -0.000000 -0.000000
vn -0.000000 -0.000000 1.000000
vn -0.000001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
usemtl Material_ray.png
s off
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
f 5/1/6 6/10/6 2/9/6
f 5/1/7 8/11/7 6/10/7
f 8/11/7 7/12/7 6/10/7
f 1/2/8 2/9/8 3/13/8
f 1/2/8 3/13/8 4/14/8
```

Figure 1.16:exemple d’un fichier .obj

A- **. ligne de commentaire :**

Le signe # : Un commentaire dans un fichier .obj indique le programme avec lequel le fichier a été écrit comme peut aussi indiquer le nombre des vertex, des textures, vecteurs normaux et des faces.

B- **Indication du fichier .MTL :**

mtllib mon_fichier.mtl : cette ligne permet de déterminer où se trouve le fichier .mtl à charger.
Usemtl ***material_name*** : indique le matériau à utiliser pour ce qui suit.

C- **Les sommets :** les sommets sont décrits en fonction des coordonnés de position les coordonnés de texture et les coordonnés du vecteur normal.

- Les coordonnées de position : chaque ligne qui décrit une position commence par la lettre " **v**" comme vertex et se note comme ceci : v X Y Z où X Y Z représentent les coordonnées (exemple `v 1.000000 -1.000000 -1.000000`)
- Les coordonnées de texture : chaque ligne décrivant la texture commence par "`vt`" Et ce note comme ceci **`vt u v`** où les coordonnés **`u v`** définissent comment chaque texture s'appliquer sur chaque facette de l'Object (exemple `vt 0.749279 0.501284`)
- Les coordonnées du vecteur normal : chaque ligne décrivant un vecteur normal commence par " vn" et se note comme ceci vn X Y Z, le vecteur normal permet de spécifier l'orientation de face ce qui est important lors de l'opération de l'éclairage et le calcul de propagation de la lumière (exemple vn -1.000000 -0.000000 -0.000000)

D- **Attribution des faces** :
les faces définies selon le type de maillage utilisé (triangulaire, quadrangulaire etc…...)
,
Chaque ligne décrivant une face indexée sous la forme suivante :

`f V1/T1/N1 V2/T2/N2 V3/T3/N3` (si la face est triangulaire)

`f V1/T1/N1 V2/T2/N2 V3/T3/N3 V4/T4/N4` (si la face est quadrangulaire)

où les V1, V2, V3, V4 sont les numéros des positions ; les T1, T2, T3, T4 sont les numéros des coordonnées de texture ; et les N1, N2, N3, N4 sont les numéros des coordonnées des normales.

Exemple : `f 5/1/1 1/2/1 4/3/1` présente une face triangulaire dont le premier sommet est défini par la position n°5, les coordonnées de texture n°1 et le normale n°1 ; et ainsi de suite pour les autres sommets.

E- Regroupement :

Il existe quatre déclarations dans le fichier .obj qui aident à manipuler les groupes d'éléments:

- Les déclarations de nom de groupe qui sont utilisées pour organiser des collections d'éléments et simplifier la manipulation des données.

Syntaxe : g *group_name*
group_name Spécifie le nom du groupe pour les éléments qui le suivent.

Voici un fichier obj qui représente un cube dont chacun de ses faces appartient à un groupe spécifié

```
 v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices
g front cube
f 1 2 3 4
g back cube
f 8 7 6 5
g right cube
f 4 3 7 8
g top cube
f 5 1 4 8
g left cube
f 5 6 2 1
g bottom cube
f 2 6 7 3
# 6 elements
```

- Groupe de lissage (Smoothing group) :

  Permet d'identifier les éléments sur lesquels les normales doivent être interpolées afin de donner à ces éléments un aspect lisse.

  Syntaxe : s *group_number*

  group_number est le numéro du groupe de lissage.si vous préférez de ne pas utiliser un groupe de lissage affecter le une valeur de 0 ou off.

  Voici un fichier. obj qui représente deux carrés adjacents qui partagent la même arête.

  Les carrés sont placés dans un groupe de lissage.

```
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
v 4.000000 0.000000 -1.255298
v 4.000000 2.000000 -1.255298
# 6 vertices
g all
s 1
f 1 2 3 4
f 4 3 5 6
```

- Nom de l'objet

  Lorsque plusieurs objets cohabitent dans le même fichier, la section définissant l'objet est définie par **o** ***object_name*** où ***object_name*** est le nom d'objet défini par l'utilisateur

1.7.2 .structure du fichier .MTL : [15]

Le format MTL (Material Template Library, ou Material Library file) est un standard défini par Wavefront Technologies en complément du format OBJ. Le fichier .mtl, est un fichier au format ASCII (texte) qui contient la définition d'un ou de plusieurs matériaux d'un objet 3D objet. voici un exemple d'un fichier .mtl

```
1# Blender MTL File: 'None'
 2 # Material Count: 2
 3
 4 newmtl Material
 5 Ns 96.078431
 6 Ka 1.000000 1.000000 1.000000
 7 Kd 0.640000 0.640000 0.640000
 8 Ks 0.500000 0.500000 0.500000
 9 Ke 0.000000 0.000000 0.000000
10 Ni 1.000000
11 d 1.000000
12 illum 2
13
14 newmtl Material.001
15 Ns 96.078431
16 Ka 1.000000 1.000000 1.000000
17 Kd 0.640000 0.640000 0.640000
18 Ks 0.500000 0.500000 0.500000
19 Ke 0.000000 0.000000 0.000000
20 Ni 1.000000
21 d 1.000000
22 illum 2
23 map_Kd  icon.png
```

Un objet est souvent composé de plusieurs partie chacune est associée à une matière (notion de groupe), le fichier mtl contient la définition de ces matériaux Chacune de ses définitions incluent la coloration, la texture et les paramètres de réflexion optique et d'autres informations :

- Le "#" spécifie une ligne de commentaires
- "newmtl" indique le début d'un nouveau matériel

- "Ka" nous donne la couleur ambiante (la couleur de l'objet sans lumière directe), RVB entre 0 (Min) et 1 (Max)
- "Kd" est utilisé pour la couleur diffuse (la couleur de l'objet sous lumière blanche)
- "Ks" pour la couleur spéculaire (specular)
- "Ke" pour la couleur émissive (emissive)
- "Ni" pour la densité optique
- "Ns" l'exposant spéculaire est multiplié par la valeur de texture
- "d" pour la transparence il est entre 0 et 1 (aucune transparence)
- "illum" pour les paramètres de lumières, spécifie le modèle d'éclairage à utiliser dans le matériau, Les modèles d'illumination sont des équations mathématiques qui représentent les effets d'éclairage et d'ombrage, le numéro d'illumination prend une valeur comprise entre 0 et 10 .
- "map_kd" spécifie la carte de texture utilisée pour la lumière diffuse
- "map_ka" spécifie la carte de texture utilisée pour la lumière ambiante

1.8. Conclusion :

La modélisation comme première partie de l'infographie 3d est une étape très complexe nécessite à comprendre les éléments constituant un maillage et les différentes techniques pour présenter un objet 3d et ainsi un bagage dans le domaine mathématique.

Les données relatives à un maillage sont stockées sous Différents formats de fichier parmi eux le format obj qui est réussi à s'imposer et à devenir un format d'échange entre logiciel à cause de sa manière de stocker les informations relative à la géométrie et celle à l'affichage des couleurs.

Visualiser ce qui est stocké et modélisé avec un grand réalisme n'est atteignable que par la maîtrise des différentes branches liées au rendu tel que les projections, la physique de la lumière, l'optique géométrique et les algorithmes de rendu comme le lancer du rayon

Dans le chapitre qui se suit on va présenter les outils informatiques qui permettent de visualiser le contenu d'un objet 3d avec un rendu plus réalisme.

CHAPITRE 2 :
LES OUTILS DE DÉVELOPPEMENTS

2.1. Introduction :

Après l'étude théorique qui a été faite dans le premier chapitre concernant la modélisation d'un objet, leur format de stockage et ainsi l'explication des différentes phénomènes physiques et mathématiques qui interviennent lors de l'étape de rendu , ce chapitre va être consacré à présenter les outils informatiques qui seront utilisés dans l'étape de développement d'une application qui permet de visualiser en temps réel une scène 3d , manipuler des objets et atteindre un rendu réaliste.

La portabilité de l'ensemble du code de l'application est un critère important dans ce projet ce qui impose d'utiliser des bibliothèques multiplateforme comme Qt et opengl et qui seront étudiées en détail dans ce qui suit.

2.2. Les principales bibliothèques multiplateformes : [21]

Chaque système d'exploitation propose au moins une bibliothèque qui permet de créer des fenêtres, mais cette bibliothèque ne fonctionne que pour le système d'exploitation pour lequel elle a été créée ce qui pose un grand problème pour les développeurs.
Dans ce cadre Plusieurs bibliothèques multiplateforme ont été développées dont le but de s'affranchir de ce handicap et d'offrir aux développeurs plus de liberté et de souplesse dans la programmation .Voici quelques-unes des principales bibliothèques multiplateforme utilisé par les développeurs selon leurs besoin :

• .NET (prononcez « Dot Net ») : développé par Microsoft pour succéder à la vieillissante API Win32. On l'utilise souvent en langage C#. On peut néanmoins utiliser .NET dans une multitude d'autres langages dont le C++.

• GTK+ : une des plus importantes bibliothèques utilisées sous Linux. Elle est portable, c'est à-dire utilisable sous Linux, Mac OS X et Windows.
GTK+ est utilisable en C mais il en existe une version C++ appelée GTKmm .

• wxWidgets : une bibliothèque objet très complète, comparable à Qt. wxWidgets n'est pas beaucoup plus compliquée que Qt. C'est la bibliothèque utilisée pour réaliser la GUI (*Graphical User Interface)* de l'IDE Code::Blocks.

• FLTK : contrairement à toutes les bibliothèques « poids lourds » précédentes, FLTK se veut Légère. C'est une petite bibliothèque dédiée uniquement à la création d'interfaces graphiques Multiplateforme.

• Qt très utilisée sous Linux, en particulier dans l'environnement de bureau KDE.

2.3. Les modules du framework QT :

Qt est une **bibliothèque** multiplateforme pour créer des GUI elle est écrite en C++ et elle est à la base conçue pour être utilisée en C++. Mais aujourd'hui il est possible de l'utiliser avec d'autres langages comme Java, Python, Ruby, JavaScript, BASIC, Ada 2005, C#, Pascal, Lua, Perl etc.

Qt est constituée d'un ensemble de bibliothèques ce qui nous conduit de parler d'un framework au lieu d'une bibliothèque, chaque bibliothèque est appelées « modules ». On peut y trouver entre autres ces fonctionnalités :

• **Module GUI** : c'est toute la partie création de fenêtres.

• **Module OpenGL** : Qt peut ouvrir une fenêtre contenant de la 3D gérée par OpenGL ,on va parler en détaille sur ce module dans les titres qui suivent .

• **Module réseau :** Qt fournit une batterie d'outils pour accéder au réseau, que ce soit pour créer un logiciel de Chat, un client FTP, un client *BitTorrent*, un lecteur de flux RSS…

• **Module XML :** c'est un moyen très pratique d'échanger des données à partir de fichiers structurés à l'aide de balises, comme le XHTML.

• **Module SQL :** permet d'accéder aux bases de données (MySQL, Oracle, PostgreSQL…).

2.4. Qt et les couches graphiques :

Le framework Qt est conçu de telle sorte que les applications développées soient compatibles avec les différents systèmes d'exploitations tel que Windows, Linux, Mac OS.

Le Framework s'appuie sur la couche graphique de chaque système d'exploitation comme win32 GDI pour Windows, X11 pour Linux et Carbon/ Cocoa pour Mac OS(voir figure 2.1).

Qt est indépendant du système et ne demande qu'une simple recompilation pour pouvoir être adapté, L'API Qt est la même sur tous les systèmes.

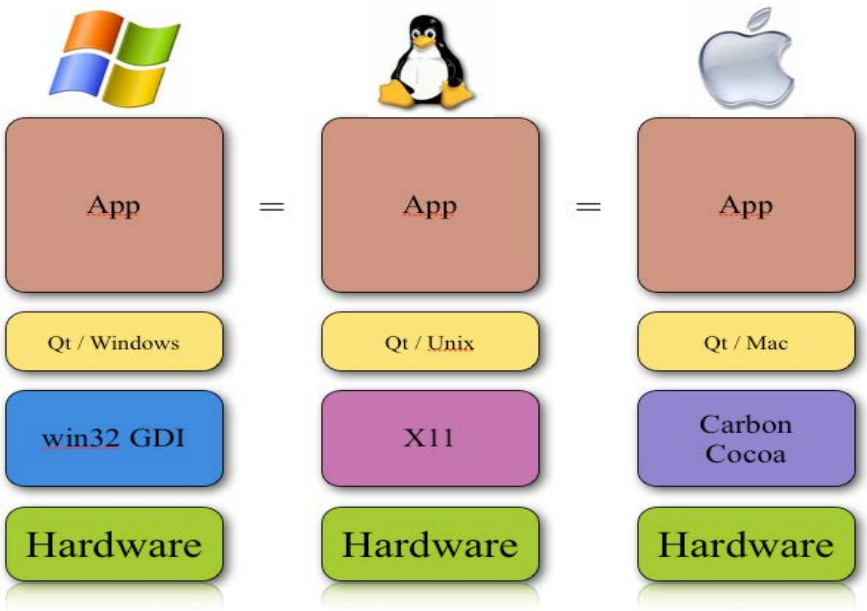


Figure 2.1:l'architecture d'une application sous Qt

2.5. Les outils de développement dans Qt :

Qt a mis à la disposition des développeurs un ensemble d'outils facilitant le développement D'applications comme :

2.5.1.Qt Designer :[22]

Qt Designer est un builder d'interfaces graphiques qui fournit au développeur la capacité de créer ses propres interfaces à l'aide de glisser-déplacer des composants graphiques tels que des bouton etc.

Ce même logiciel va ensuite générer un fichier «.ui» basé sur la norme XML. Ce fichier est utilisé par le compilateur Qt lors de la compilation du fichier «.pro». De plus, la génération des fichiers sources et des fichiers d'en-têtes est automatisée par Qt Designer à l'aide du compilateur intégré User Interface Compiler. Donc la génération du code source à l'aide de Qt Designer est propre et Les fichiers sources sont réutilisables très facilement.

2.5.2. Qt Linguist :[23]

Qt intègre son propre système de traduction « Qt Linguist » qui permet de simplifier la vie des développeurs en traduisant l'interface de leur application dont le but d'assurer l'internationalisation.

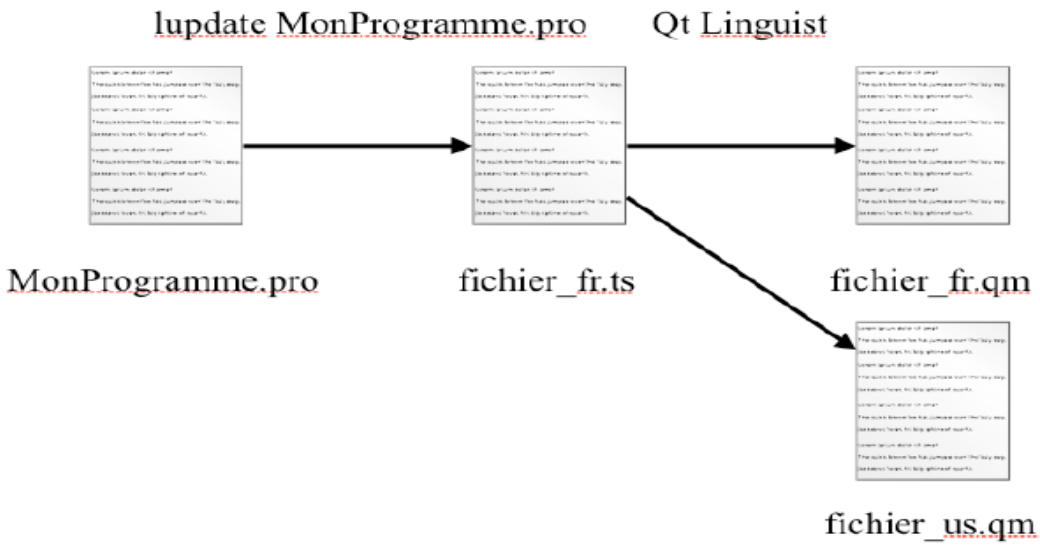
Dans leur code source, les développeurs entrent des chaînes de caractères dans leur propre langue précédées de la méthode tr().

Le chef de projet déclare les fichiers de traduction (un pour chaque langue) dans le fichier de projet.

L'utilitaire **lupdate** parcourt les sources à la recherche de chaînes à traduire et synchronise les fichiers de traduction avec les sources. Les fichiers de traductions sont des fichiers xml portant l'extension *.ts*.

Qt Linguist permet d'ouvrir les fichiers «.ts» et donne la possibilité de traduire ses fichiers dans la langue de son choix. Par exemple si le développer à généré un fichier mon_app_francais.ts qui contient les mots tels que bonjour, il pourra traduire ce mot en hello et générer le fichier mon_app_anglais.ts pour sa traduction en anglais.

Une fois les fichiers «.ts» créés il faut générer les fichiers «.qm» qui sont utilisable par l'application Qt afin de traduire l'interface. Ces fichiers s'obtiennent en convertissant les fichiers «.ts» à l'aide de Qt Linguist comme indiqué ci dessous



2.5.3.Qt assistant :[24]

Il s'agit d'un programme permettant de consulter la documentation de Qt en mode hors ligne.

Qt assistant a les avantages suivants :

- recherche rapide de mots clés, de texte complet, présence d'un index et de marque-pages .
- indexation et recherche de plusieurs documents d'aide simultanément ;

- accueille une documentation locale ou fournit une aide en ligne dans votre application.

2.5.4. Qt Creator : [25]

C'est un IDE (Environnement de Développement Intégré) qui contient un ensemble d'outils qui offre aux développeurs la possibilité de concevoir des applications et des interfaces utilisateurs , son éditeur de texte se caractérise par le complètement automatique , l'indentation et la coloration syntaxique, etc. ce fameux IDE intègre un ensemble d'outils comme Qt Designer et Qt Assistant Qt Linguist pour le support de l'internationalisation.

Dans Qt creator un kit spécifie les compilateurs et les autres outils nécessaires pour créer une application et l'exécuter sur une plate-forme particulière.

La détection des compilateurs enregistrés dans le système d'exploitation se fait d'une manière automatique exemple (MSVC, MinGW**)** comme vous pouvez installer d'autres compilateurs supplémentaires.

2 .6 . les principaux Widget dans Qt :[26]

Le fait de pouvoir créer une classe de base, réutilisée par des sous-classes filles, qui ont elles mêmes leurs propres sous-classes filles, cela donne à une bibliothèque comme Qt une puissance infinie.

- **QObject** : est la classe de base de tous les objets sous Qt. toutes les autres classes héritent de cette classe (voir la figure 2.2 permet de mieux visualiser la hiérarchie des classes)
- **QWidget :** Dans une fenêtre, tout est considéré comme un widget et tous les widgets héritent de QWidgt qui contient énormément de propriétés comme la hauteur, la largeur, la police de caractères etc…
- **QAbstructButton :** est une classe abstruct on ne peut pas créer d'objet a partir d'elle
- **Les widget intérateurs :** comme **RdioButton, CheckBox** etc.. .
- **Les widgets conteneurs :** Certains widgets ont été créés spécialement pour pouvoir en contenir d'autres :

  -QFrame : un widget pouvant avoir une bordure.

  - QGroupBox : un widget adapté à la gestion des groupes de cases à cocher et de boutons radio.

-QTabWidget : un widget gérant plusieurs pages d'onglets.

- **Les widgets afficheurs :** Parmi les widgets afficheurs, on compte principalement

  Qlabel : le plus important, un widget permettant d'afficher du texte ou une image

  QprogressBar : permet d'indiquer à l'utilisateur l'avancement des opérations.

- **Widget des champs :** Voici les principaux widgets qui permettent de saisir des données :
  - QLineEdit: champ de texte à une seule ligne ;
  - QTextEdit: champ de texte à plusieurs lignes pouvant afficher du texte mis en forme
  - QComboBox: liste déroulante.

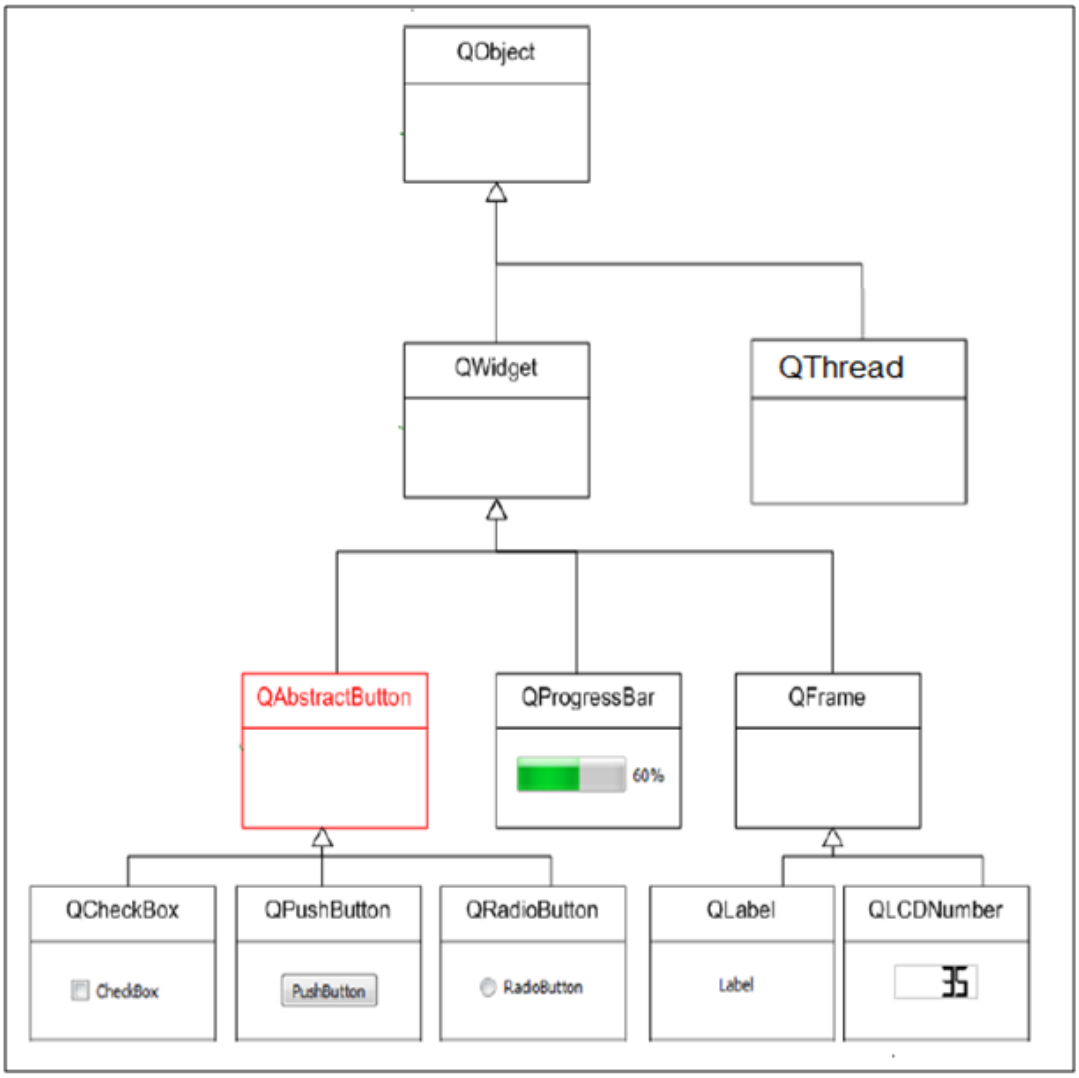


Figure 2.2:Arbre d'héritage dans Qt

.

2.7. Les signaux et les slots :

Une application de type GUI réagit à partir d'évènements, c'est ce qu'on appelle Dans Qt « signaux et de slots » , ce mécanisme permet de faire communiquer les objets entre eux sans les liés ce qui permet une réutilisation du code par la suite.

• Un signal : c'est un message envoyé par un widget lorsqu'un évènement se produit.

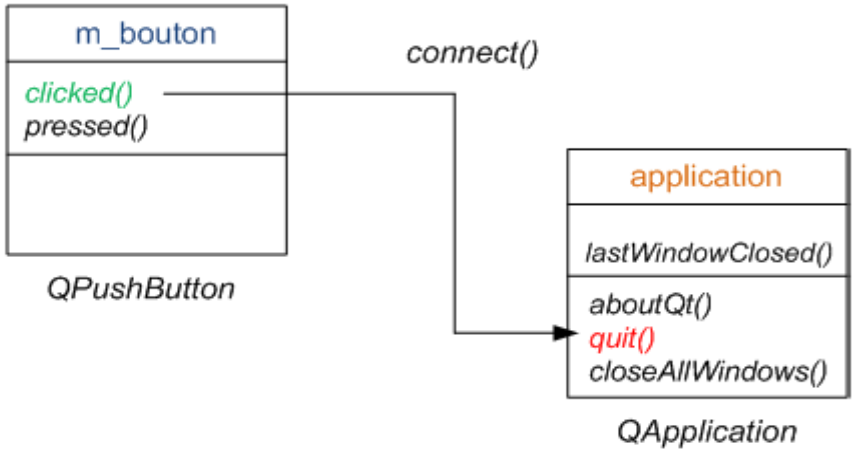
• Un slot : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.

Les signaux et les slots sont considérés par Qt comme des éléments d'une classe à part entière, en plus des attributs et des méthodes.

On peut Connecter un signal à un autre et créé des cascades d'appels, on peut supprimer un signale (fonction «disconnect»). La connexion d'un signal à un slot se fait à l'aide de la commande «connect».

Voici un simple exemple qui explique la connexion d'un signal à un slot :

L'exemple consiste à provoquer l'arrêt d'une application à travers un clic bouton comme illustré ci-dessous



La méthode **connect() doit** contenir :

•un pointeur vers l'objet qui émet le signal (m_bouton)

• le nom du signal que l'on souhaite intercepter (clicked)

• un pointeur vers l'objet qui contient le slot récepteur (application)

• le nom du slot qui doit s'exécuter lorsque le signal se produit (quit())

```
QObject::connect (m_bouton, SIGNAL(clicked()), qApp, SLOT(quit()));
```

2.8 Qu'est-ce qu'OpenGL?

OpenGL (Open Graphics Library) est une bibliothèque graphique très complète 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.

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. A charge à l'utilisateur ou à d'autres bibliothèques spécialisées de construire des objets plus complexes `a l'aide de ces primitives.

Plusieurs bibliothèques sont développées à partir d'OpenGL afin d'apporter des fonctionnalités qui ne sont pas disponibles dans la bibliothèque OpenGL elle même comme GLM,GLUT etc…

2.9. L'architecture Opengl :[27]

La figure 2.3 représente une version simplifiée de l'architecture OpenGL.

Le processus de rendu OpenGL est le suivant :

Une fois que l'application qui utilise OpenGL a envoyé les informations nécessaires à l'affichage des éléments de la scène, elle demande un rendu de chaque élément.

OpenGL passe alors par les phases suivantes :

– une phase de calculs préliminaires pour les lumières et les matériaux (Transformation and Lighting),

– une phase de calcul de la forme des éléments de la scène projetée dans l'espace image,

– une phase dite de rastérisation, qui effectue une opération de tramage de l'image finale. Cela a pour effet de représenter les éléments de la scène en pixels.

– une phase d'application de textures aux éléments de la scène,

– une phase de calcul de la valeur (couleur et profondeur) des pixels de la phase de rastérisation,

– enfin la phase d'écriture dans le frame buffer, qui correspond à une copie de l'image finale de rendu.

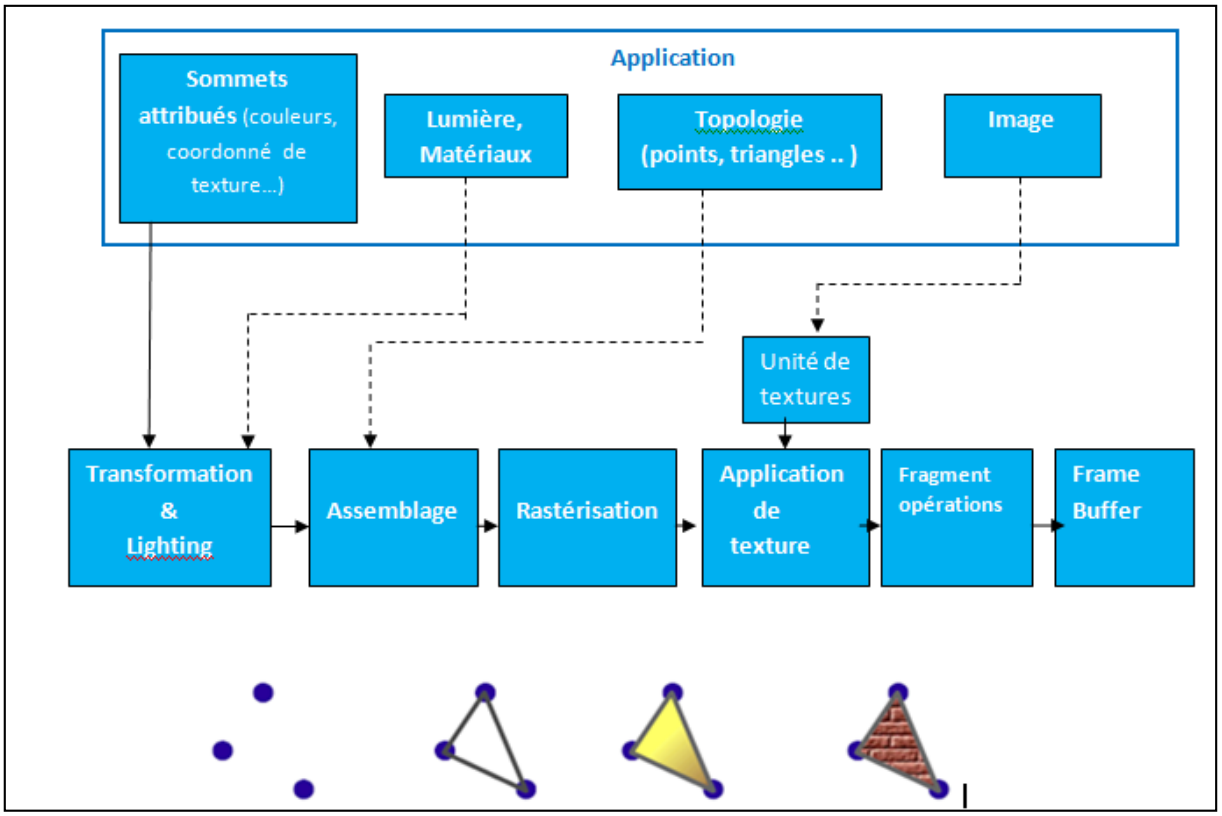


Figure 2.3: Architecture Opengl

2.10.Shading Language :

A partir de la version 2.0 d'OpenGL, apparaît une modification de l'architecture (voir la figure 2.4), qui rend celle-ci beaucoup plus modulaire.

Cette évolution est en fait conjointe de celle des architectures de cartes accélératrices graphiques (GPU = Graphical Processing Units),qui deviennent de plus en plus programmables.

Les phases de Transformation and Lightning et d'application de texture peuvent alors être remplacées par des phases entièrement programmables: **les Vertex shaders et les Fragment shaders**.

Le Vertex shader est exécuté au début de la phase de rendu d'un élément donné de la scène, et une seule fois pour chaque élément. Le Fragment shader est exécuté pour chaque pixel qui compose un élément, et a accès aux résultats du Vertex shader.

Les Vertex et Fragment shaders utilisent un langage spécifique, le Shading Langage, relativement proche du langage C.

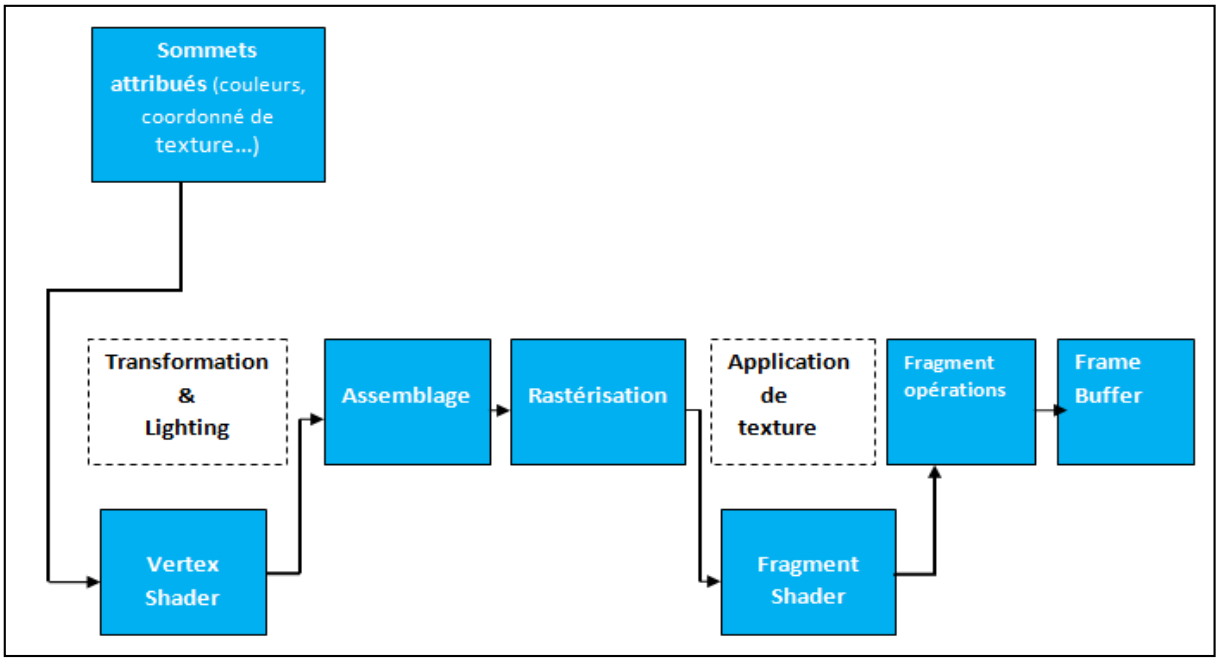


Figure 2.4:L'architecture OpenGL avec les shaders

2.11. Opengl une Machine à états : [32]

OpenGL est une machine à états,les états sont représentés par des variables d'états.

La couleur est par exemple une variable d'état. Une fois fixée, tous les objets seront dessinés dans cette couleur jusqu'à ce qu'on change cet état.

La notion d'état courant d' OpenGL concerne notamment les transformations, les motifs de lignes et de polygones, les modes de dessin, les positions et les caractéristiques des lumières et les propriétés des objets dessinés. Des fonctions comme glEnable () ou glDisable () permettent au programmeur d'activer ou de désactiver certaines possibilités offertes par OpenGL). Par exemple, glEnable (GL_ALPHA_TEST) permet d'activer la réalisation par OpenGL des tests de transparence des objets. Avec OpenGL, on peut enregistrer sur une pile puis restaurer depuis cette pile les valeurs d'un état à l'aide des instructions de type glPushxxx () et glPopxxx (). On peut également prendre connaissance d'un état courant à l'aide d'instructions de type glGet{xxx).

2.12. Spécification d'une primitive géométrique : [29]

Pour définir une primitive 3D, il est nécessaire :

- de préciser le type de primitive que l'on souhaite construire.
- d'énumérer les points qui permettent de construire cette primitive.

La définition d'une primitive doit toujours se faire entre 2 fonctions particulières, **glBegin** et **glEnd**. Elle consiste alors à énumérer une liste de points qui vont être considérés comme les sommets de la primitive à construire.

-Un sommet est défini par la fonction **glVertex*(…) .**

La syntaxe des 2 fonctions glBegin et glEnd est la suivante :

glBegin(GLenum mode);

glEnd();

où **mode** spécifie le type de primitive qui doit être construite. Parmi les valeurs possibles de ce paramètre, en voici quelques unes :

- **GL_ LINES** : définition d'une suite de segments, chaque segment étant définie par un couple de points successifs ;
- **GL_TRIANGLES** : définition d'une suite de triangles, chaque triangle étant défini par trois points successifs ;
- **GL_ QUADS** : définition d'une suite de quadrilatères ;
- **GL_POLYGON** : définition d'un polygone construit avec tous les points qui suivent.

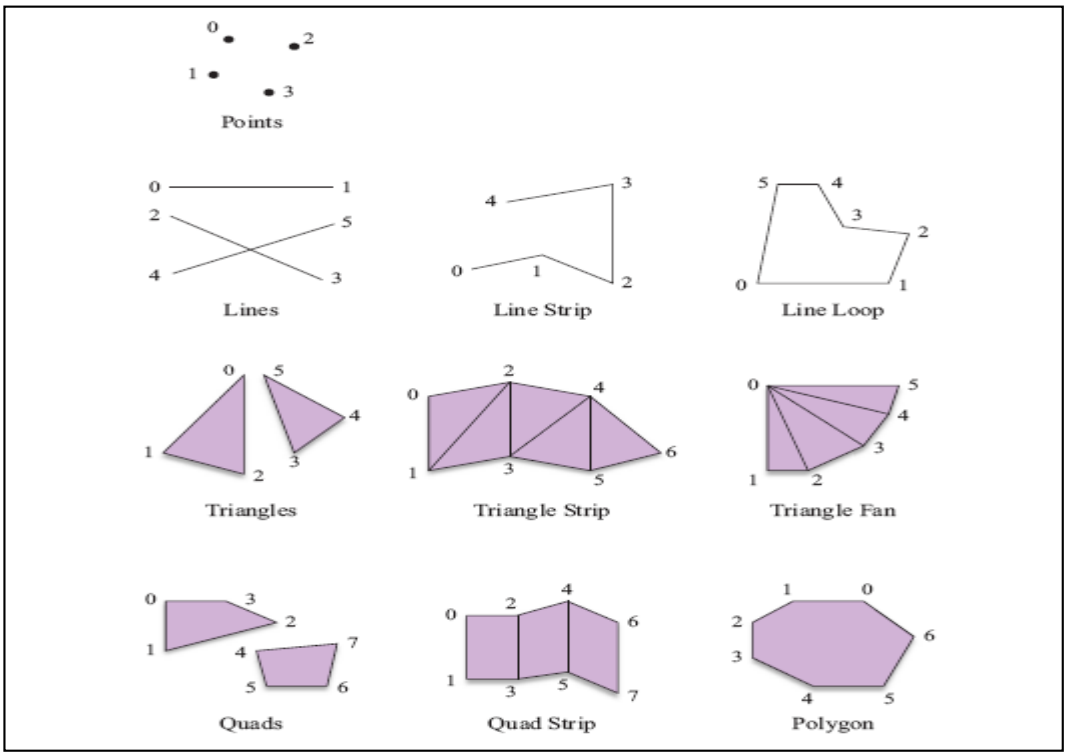


Figure 2.5:les Primitives graphiques permises par OpenGL

2.13. L'éclairage en OpenGL :[28]

2.13.1. Activer l'éclairage :

Deux opérations sont, au minimum, requises pour que l'on puisse éclairer les objets d'une scène :

- activer l'éclairage : ceci se fait par l'appel de la fonction suivante:

glEnable(GL_LIGHTING).

- Désactiver l'éclairage :ceci se fait par de la fonction suivant :

glDisable(GL_LIGHTING).

Activer une ou plusieurs sources de lumière : sous OpenGL chaque source est identifiée par une constante symbolique de la forme **GL_LIGHTn**, avec n ∈ [O,…, 7].

Chaque source peut être activée par un appel de la forme **glEnable(GL_LIGHTn)** ou désactivée par un appel de la forme **glDisable(GL_LIGHTn).**

2.13.2. Rendu lissé ou non lissé :

Par défaut le rendu est effectué en mode lissé, c'est à dire que la valeur de couleur en tout point d'une facette est interpolée depuis la valeur de couleur aux sommets de cette facette.

Le mode de rendu peut être modifié par l'intermédiaire de la fonction **gl_Shade_Model**, qui peut prendre l'une des deux valeurs suivantes comme unique paramètre :

GL _SMOOTH : le rendu doit être effectué en mode lissé (valeur par défaut) ;

GL _FLAT : le rendu doit être fait en mode **Lambert** , c'est à dire avec une intensité constante sur toute la surface de la facette.

2.13.3 Spécifier le matériau des objets :

La couleur d'un objet est précisée par l'intermédiaire d'un appel à la fonction **glColor3f**. Lorsque l'éclairage est activé, les appels à cette fonction sont ignorés, les caractéristiques liées au calcul de la couleur devant être spécifiées par l'intermédiaire d'un matériau.

Sous OpenGL , chaque objet est issu d'un matériau, dont les différentes composantes permettent de spécifier les différents coefficients de réflexion de l'objet (réflexions ambiante, diffuse et spéculaire).

2.13.3.1. Réflexion ambiante et diffuse :

La définition des propriétés de réflexion ambiante et diffuse d'un objet se font par l'intermédiaire de la fonction suivante :

glMaterialfv(GLenum face, GLenum propriete, float *valeurs)

avec:

-face : permet de préciser si le matériau s'applique sur les faces visibles (**GL FRONT**), ou invisibles (**GL BACK**) ou les deux (**GL FRONT AND BACK**) .

-propriété : permet de préciser les propriétés de réflexion du matériau ; les valeurs possibles, pour cette partie, sont :

GL_AMBIENT : la valeur des coefficients ambiants ;

GL_ DIFFUSE : la valeur des coefficients diffus ;

GL_ AMBIENT_AND_DIFFUSE : spécifie la (même) valeur pour les coefficients ambiants et diffus.

valeurs : représente un tableau de 4 valeurs réelles, comprises entre 0.0 et 1.0 :
une valeur pour chacune des 3 primaires R(rouge), V(vert) et B(bleu) et une pour le canal alpha (lié à la notion de transparence).

2.13.3.2. Réflexion spéculaire et brillance :

Sous OpenGL il est possible de gérer la brillance d'un objet, d'une part en précisant les coefficients spéculaires utilisés pour chaque composante R, V, B et alpha et, d'autre part, en précisant un indice de brillance « **s** » , compris entre 0.0 et 128.0, qui précise la largeur du cône spéculaire .

Par défaut, les valeurs des coefficients spéculaires associés à un matériau sont (0.0; 0.0; 0.0; 1.0), le coefficient « s » valant 0.0 . Pour affecter des valeurs aux coefficients, il faut utiliser la même fonction que pour les coefficients diffus et ambiant : **glMaterialfv**, en utilisant la constante symbolique **GL_SPECULAR** pour le paramètre propriété.

-La valeur de « s » doit être fixée par une fonction légèrement différente :

glMaterialf (GLenum face, GLenum propriete, float valeur)

Avec**:**

propriété : la constante symbolique **GL_SHININESS .**
valeur : un réel dans [0.0; 128.0].

2.14. Les Textures

Pour modifier l'apparence d'un objet, il est relativement simple de plaquer une image à la surface des triangles. L'objectif est de modifier la couleur des pixels qui sont utilisés pour dessiner l'objet.

Une texture est une « image », c'est à dire une grille de pixels. Les pixels de la texture sont appelés texels (pour les différencier des pixels de l'écran graphique).
Chaque texel est localisé dans la texture par ses coordonnées **s** et **t**.
Toute l'image de la texture est d'écrite par s ∈ [0; 1] et t ∈[0; 1]

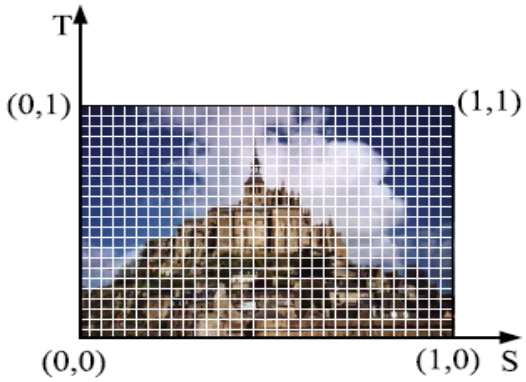


Figure 2.6:grille de pixel

2.14.1. Texturer les objets : [30][31]

Le plaquage de texture Consiste à associer à chaque point affiché un texel ,
L'association se fait simplement en indiquant les coordonnées **(s; t)** du texel voulu.
En OpenGL, l'association se fait seulement aux sommets.
Donc pour réaliser le plaquage de la texture sur la face il faut faire correspondre à chaque vertex (sommet) composant notre face une coordonnée sur la texture.
On définit les coordonnées du vertex par la fonction **glVertex3d(x,y,z),** et pour définir les coordonnées de texture on utilise la fonction **glTexCoord2d (s,t).**
voici la méthode de plaquage d'une image sur un polygone 3d (figure 2.7) en utilisant opengl

```
glBegin(GL POLYGON) ;
glTexCoord2f ( 0 , 0 ) ;
glVertex3fv (V0 ) ;
glTexCoord2f ( 1 , 0 ) ;
glVertex3 f (V1 ) ;
glTexCoord2f ( 1 , 1 ) ;
glVertex3 f (V2 ) ;
glTexCoord2f ( 0 , 1 ) ;
glVertex3 f (V3 ) ;
glEnd ( ) ;
```

.

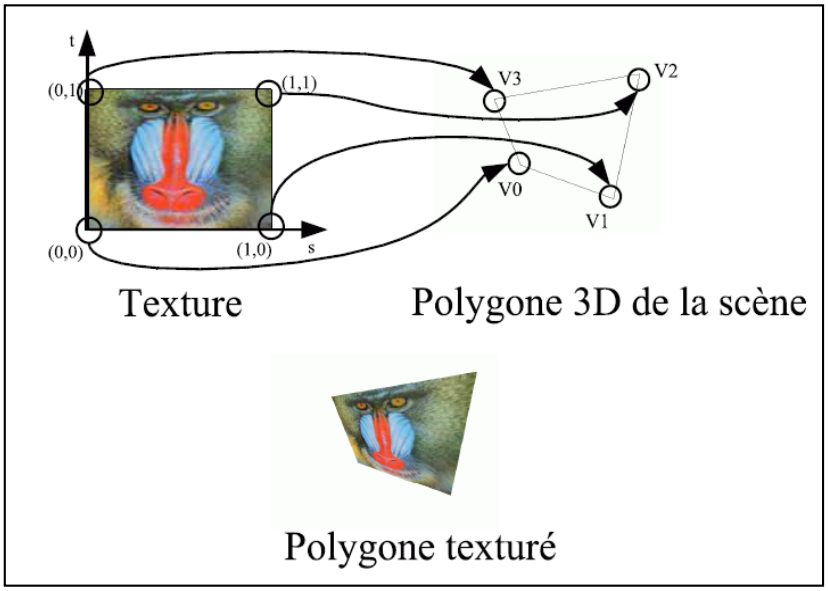


Figure 2.7:le Plaquage de texture

2.14.2. Cas d'un triangle :

Dans les jeux vidéo, la primitive de base la plus utilisée est le triangle. Le plaquage de texture fonctionne de la même manière en ne choisissant que 3 points sur la texture, points qui ne sont donc pas forcément des coins de l'image.

```
glBindTexture(GL_TEXTURE_2D, texture2);
glBegin(GL_TRIANGLES);
glTexCoord2d(0,0);
glVertex3d(1,1,-1);
glTexCoord2d(1,0);
glVertex3d(-1,1,-1);
glTexCoord2d(0.5,1);
glVertex3d(0,0,1);
glEnd();
```

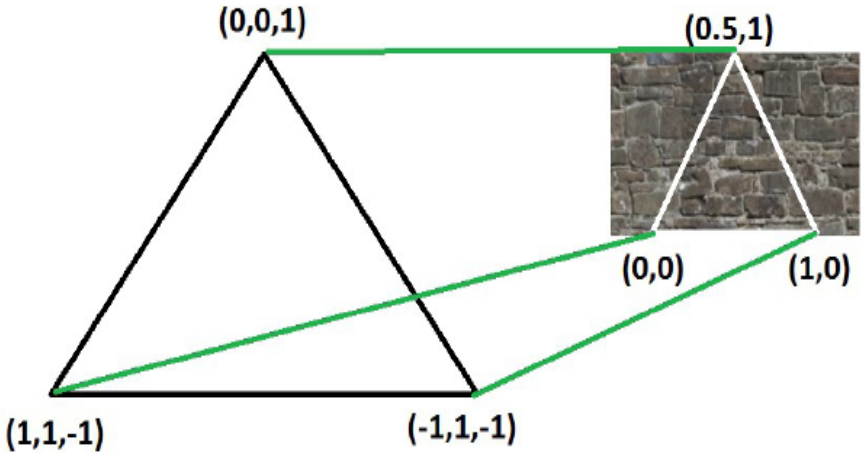


Figure 2.8:Le plaquage de texture cas d'un triangle

2.15. Conclusion :

L'étape de développement d'une application nécessite une bonne connaissance des outils et des langages de programmations utilisés, Pour cette raison on a abordé dans ce chapitre une étude du framework QT : les différents modules qu'il construisent, les outils de développements de ce framework et les variétés des widget du Qt creator qui permettent de créer une interface utilisateur.

Dans la deuxième partie de ce chapitre on a décrit l'architecture d'opengl comme on a présenté les fonctions d'opengl qui permettent de visualiser une scène 3d ainsi les techniques utilisés pour atteindre un niveau de réalisme plus élevé comme l'éclairage et le plaquage de texture.

CHAPITRE 3 : APPLICATION

3.1 .introduction :

Après avoir établi une étude sur les notions générales de la 3d et l'explication du framework Qt et la bibliothèque opengl, ce chapitre va être consacré à l'implémentation d'un outil de visualisation en temps réel d'objet 3d

3.2 Le choix des outils de développement QT :

3.2.1. Choix de Qt :

On a choisi le Framework QT(version gratuite 5.10.0) de la société Trolltech pour développer notre application , pour les raisons suivantes :

Qt permet de créer des applications compatibles avec les principales plates formes disponibles sur le marché à partir d'un unique code source ce qui permet d'économiser en temps, en effort et en argent.

D'un point de vu technique QT permet de développer des interfaces indépendantes de la configuration matérielle et logicielle.

L'un des objectifs de notre application est d'afficher un rendu opengl dans une fenêtre Qt et ça est offert par Qt en faisant une simple configuration dans notre programme ce qui permet d'appeler le module opengl intégré dans Qt .

Plus loin de l'aspect graphique Qt permet de gérer le multi htreading , les connexions réseaux, les connections aux bases de données SQL etc.

3.2.2. IDE :

L'environnement de développement intégré (IDE) utilisés lors de la programmation de notre application est Qtcreator du Qt.
QtCreator joue le rôle d'un IDE permettant de développer en C++ et d'autres langages de programmation et au même temps permet de créer des projets, les éditer, les débugger et de consulter la documentation de Qt.

3.2.3. OpengL :

OpenGL est une API (Application Programming Interface) multi-plateforme pour la conception d'applications générant des images 3D et 2D, l'utilisation de cette librairie est totalement gratuite et elle met à la disposition son propre code source ce qui facilite

l’apprentissage aux développeurs. Opengl est très rapide en termes d’exécution, il exploite aujourd'hui toute la puissance des cartes graphiques du marché.

3.2.4. Choix du langage :

Notre choix en terme de langage de programmation est orienté vers le C++, car c'est le langage le plus utilisé quand il s'agit d'utiliser OpenGl . Le C++ est un langage de programmation orienté objet classé parmi les meilleurs langages qui assurent le développement des applications temps réel.

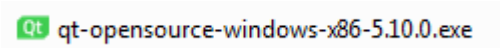
3.3. Préparation de l’environnement de développement :

3.3.1. L’installation de Qt version 5.10.0

La librairie Qt a subit de nombreux rebondissement de licence elle est actuellement disponible sous forme communautaire et commerciale, vous pouvez télécharger Qt à partir du site officiel www.qt.io

Dans ce projet on a installé la version gratuite de Qt , le site www.qt.io vous propose deux mode pour le téléchargement et le choix des outils et versions :

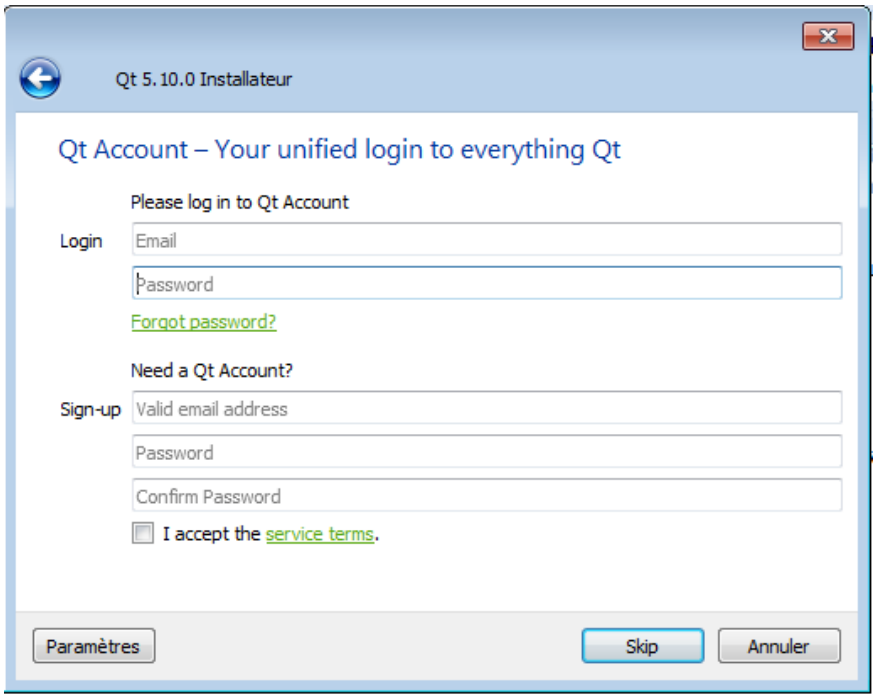
- la version en ligne, qui contient uniquement l'installateur. Toutes les outils seront installé après téléchargement en ligne elle nécessite un bon débit de connexion internet .
- la version hors ligne, qui contient l'installateur et une seule version de Qt ce mode permet une installation de Qt sans avoir de connexion internet.

Apres le téléchargement de Qt vous disposez donc sur votre ordinateur un installateur (taille 2 GO environ) correspond au fichier 

Vous pouvez lancer l’installation en double cliquant sur le fichier

Une première page apparaitre permet de rappeler les différentes licences utilisables avec Qt, cliquer sur suivant si vous voulez continuer avec une version open source (voire la figure ci dessous)

Pour installer Qt, il est t nécessaire de créer un compte. Si vous en avez déjà créé un sur le site qt.io , vous pouvez l'utiliser directement. Sinon vous devez créer un compte en remplissant le formulaire comme il est indiqué sous la figure suivante :

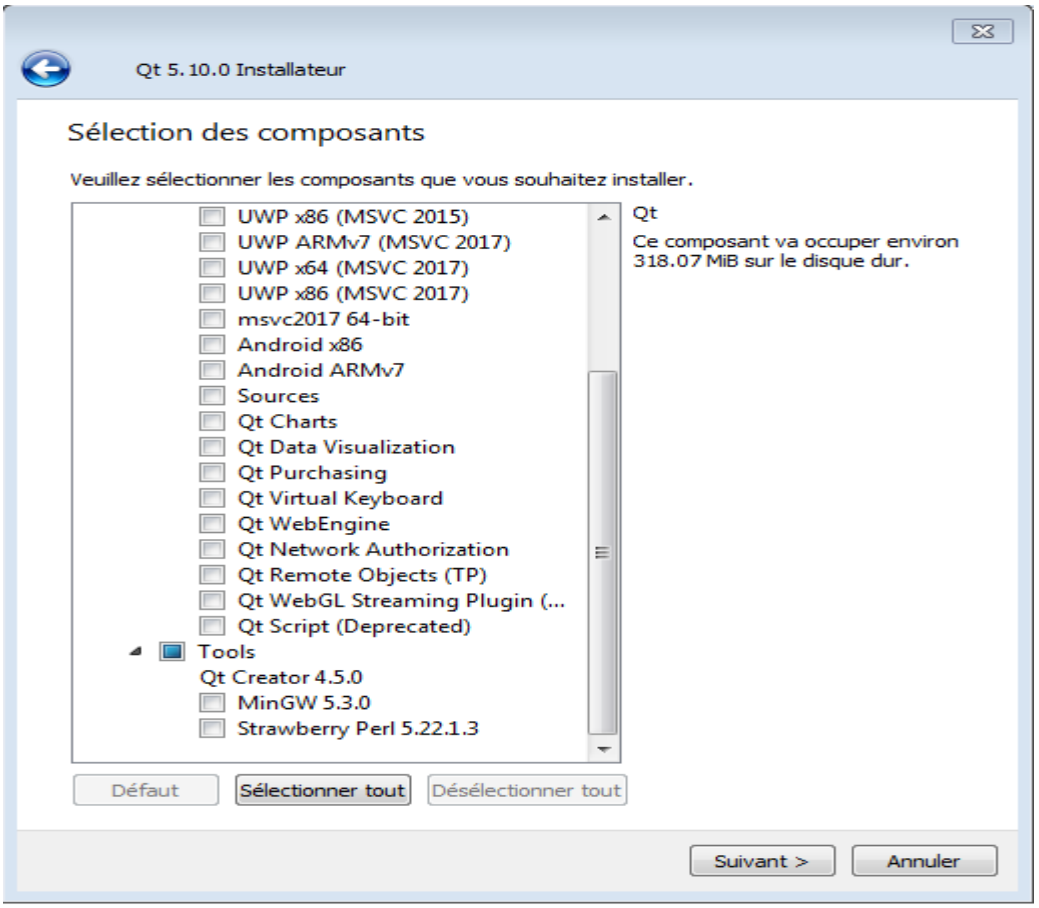


Puis cliquez sur Suivant.

Une page apparaitre seulement pour vous dire bienvenu cliquer sur suivant pour continuer.

Dans la page qui se suit indiquer le chemin d'installation de Qt puis cliquer sur suivant.

La fenêtre ci-dessous s'affiche .



Cette fenêtre vous permet de sélectionner la liste des outils à installer.

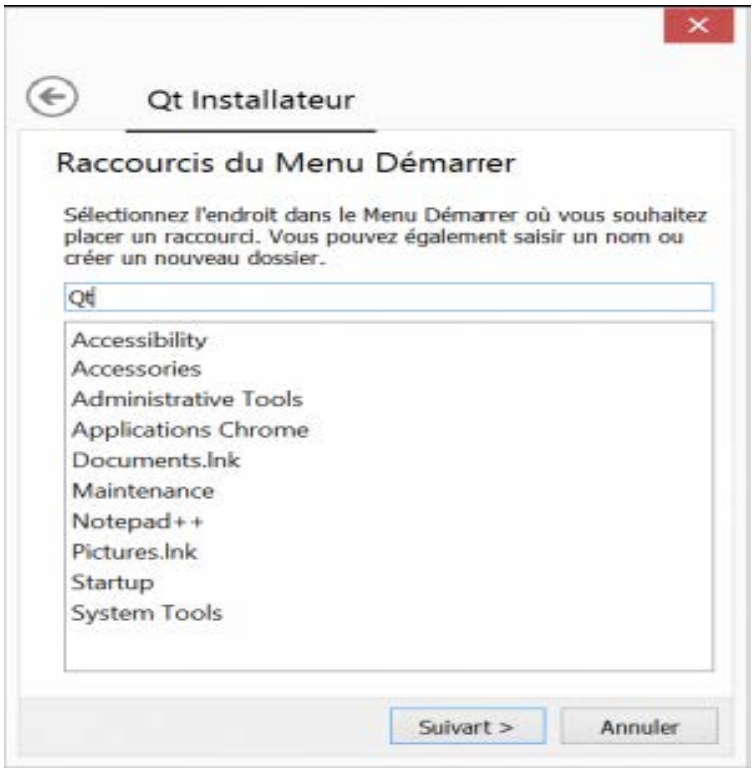
Pour programmer en C++ avec Qt, il faut installer au moins:

L'IDE Qt Creator , et un compilateur compatible avec votre système d'exploitation et la version de Qt installée. Dans ce projet on a choisi comme compilateur MinGW 5.3.0 Compatible avec windows7 32bit.

Si vous êtes débutants en Qt et vous ne savez pas quels outils sélectionner cliquer sur sélectionner tout puis sur suivant, lors de l'installation des messages de non compatibilité de quelques outils avec votre système s'affichent dans ce cas la vous pouvez cliquer sur ignorer pour continuer l'installation.

Apres l'étape précédente, une autre page vous permet de valider les licences utilisateurs. Cliquez sur suivant pour continuer.

La page suivante permet de choisir le répertoire dans le menu Démarrer.



Cliquer sur suivant pour continuer.

Une autre fenêtre s'ouvre pour vous informer que l'installation est prête à être démarrée donc cliquer sur installer pour commencer l'installation.

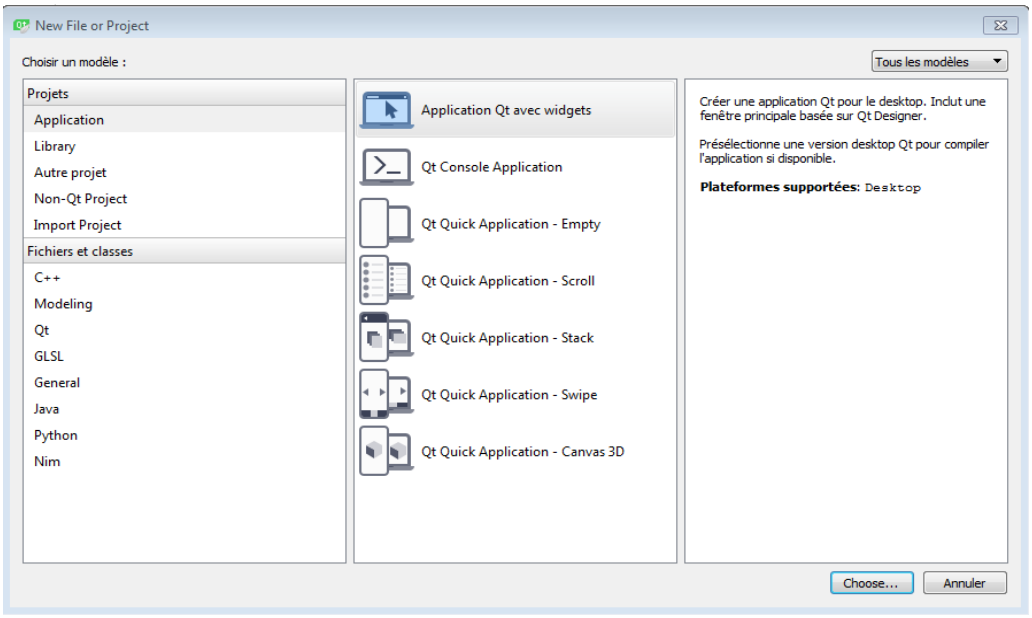
Une fois l'installation est terminée cliquer sur terminer.

3.3.2. Créer votre projet :

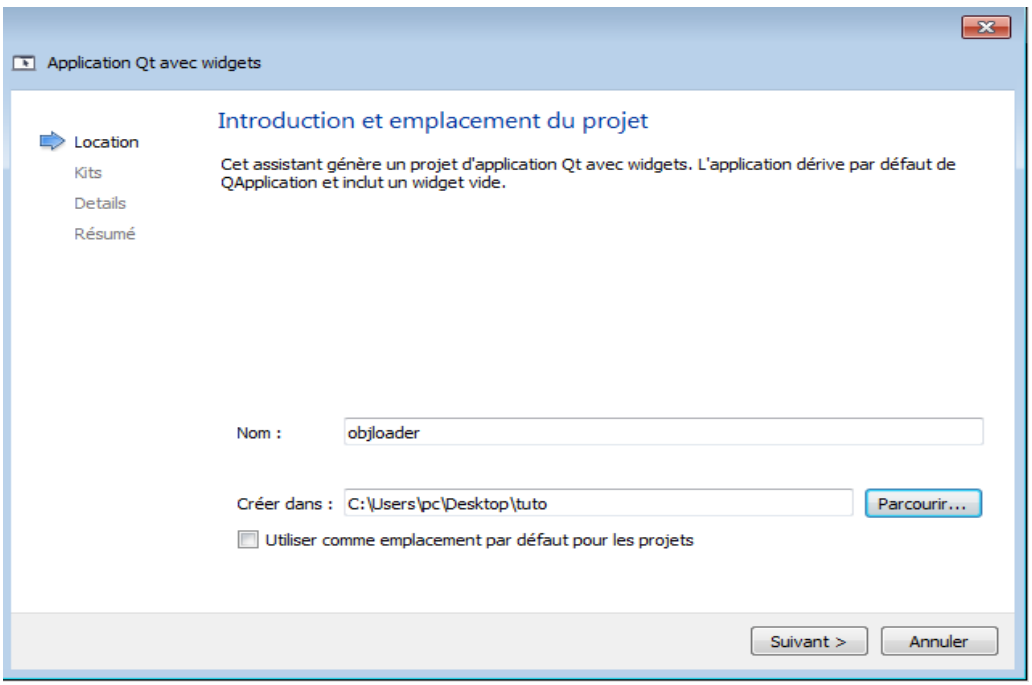
Pour créer votre projet lancer Qt creator et sélectionner :

Fichier puis cliquer sur **Nouveau fichier ou projet**

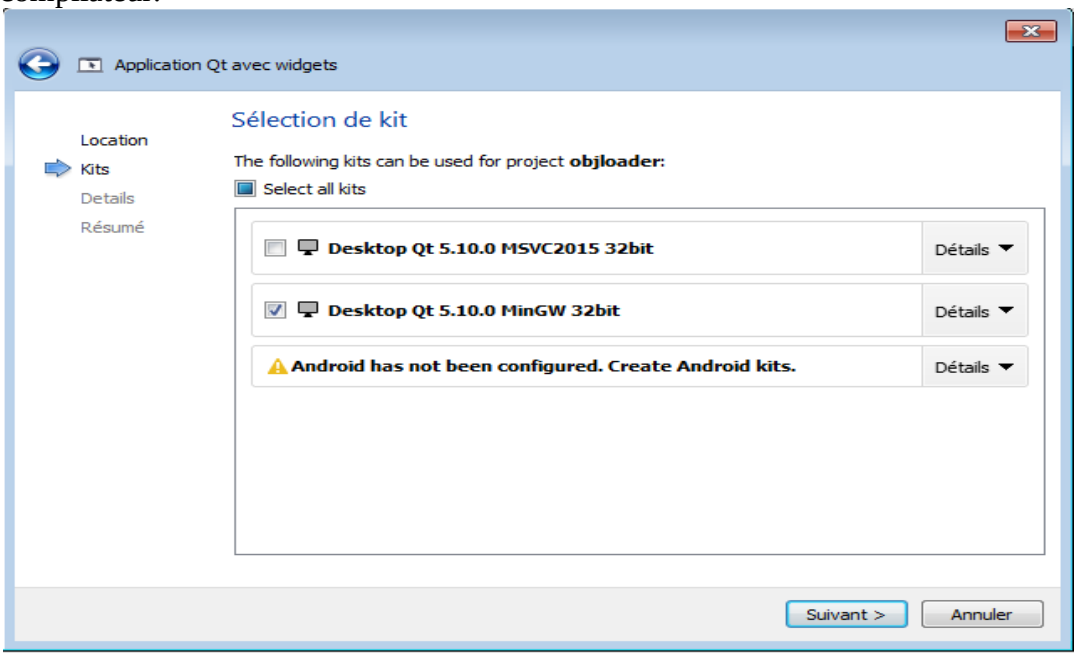
On va développer une nouvelle application Qt utilisant des widgets donc cliquer sur **Application** ensuite sur **application Qt avec widget** comme il est indiqué sur l'image ci-dessous



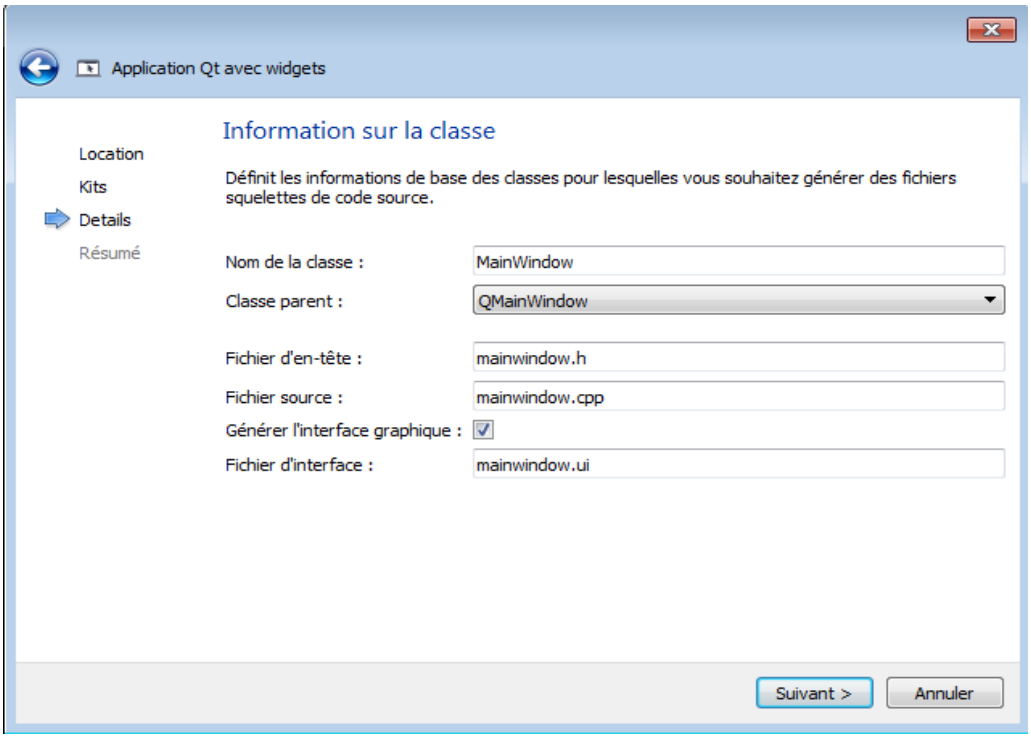
Choisir un répertoire pour sauvegarder tous les fichiers de ce projet



Choisir le **Kit** de développement , dans ce projet on va utiliser MinGW 32 bit comme compilateur.

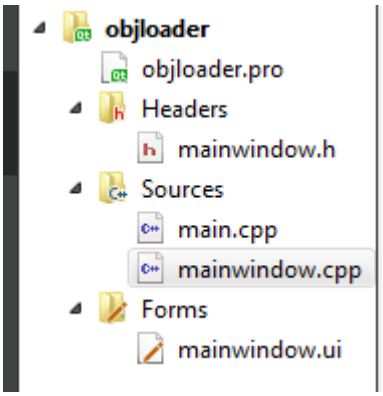


Ensuite choisir la classe de départ de votre programme. Une "QMainWindow" est le choix par défaut, elle contient tous les outils nécessaires pour la fenêtre principale du programme (voir l'image ci-dessous)



Une dernière fenêtre vous permettra, si vous voulez, d'utiliser un gestionnaire de versions. Comme dernière étape il ne reste plus qu'à valider, et votre projet est prêt.

Le projet créé contient Cinq fichiers comme il est montré sur l'image suivante :



- **Le fichier objloader.pro :** Ce fichier décrit votre projet , à savoir qu'elle utilise les composants Qt de base (*core*), les composants graphiques **(*gui*).**
  le fichier indique aussi les fichiers sources
  (*SOURCE*) **main.cpp** et **mainwindow.cpp**, le fichier d'entête
  (HEADERS) **mainwindows.h**, et le fichier de description graphique de l'interface (*FORMS*) **mainwindow.ui** (voir figure 3.1)

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = objloader
TEMPLATE = app

DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += \
        main.cpp \
        mainwindow.cpp

HEADERS += \
        mainwindow.h

FORMS += \
        mainwindow.ui
```

Figure 3.1:le contenu du fichier *.pro

- **mainwindow.h** : le header déclarant la classe **MainWindow**
- **mainwindow.cpp** : c'est le fichier source implémentant la classe **MainWindow** au début ce fichier ne contient que l'implémentation du constructeur et du destructeur. Le premier initialise l'interface « `ui->setupUi(this)` »;, le second libère la mémoire utilisée pour l'interface « `delete ui;` ».

main.cpp : c'est lui qui s'exécute en premier et qui lance la partie graphique, Le lancement consiste à instancier un objet **a** de la classe **QApplication**, un objet **w** de la classe **mainwindow** et enfin d'appeler la méthode **show()** de ce dernier qui permet l'apparition de la fenêtre(voir la figure 3.2).

La méthode **a.exec()** lance **un thread** principal d'évènement ce qui permet le programme de rester actif pour répondre aux actions qu'on va faire, tant que la fenêtre principale est active .

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Figure 3.2:le contenu du fichier main.cpp

- le fichier **mainwindow.ui :** c'est un fichier qui définit l'interface graphique

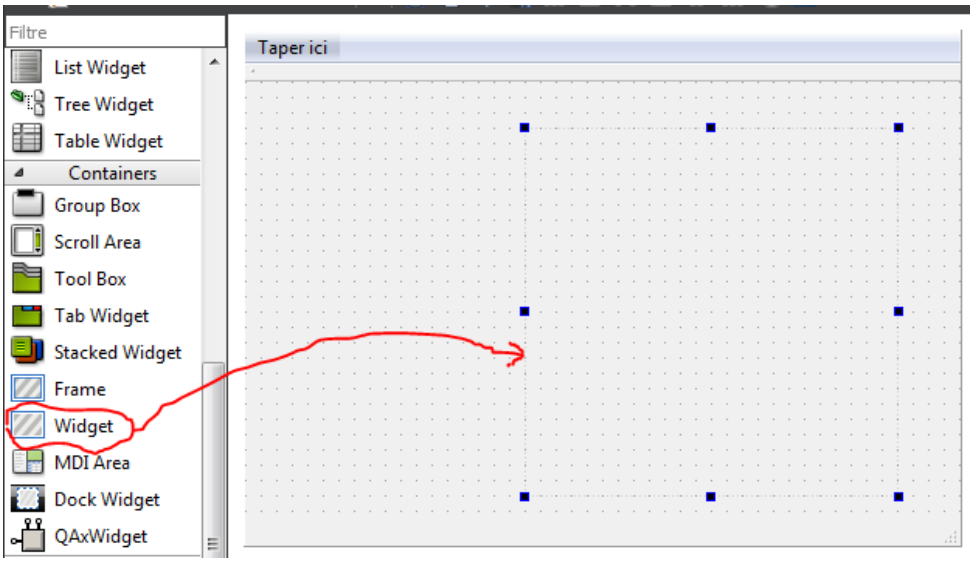
3.3.3. Intégration d'opengl dans l'interface Qt :

Qt propose un module OpenGL qui permet d'afficher des rendus OpenGL dans une fenêtre Qt, donc pour permettre au compilateur d'ajouter le module OpenGL aller vers le fichier **objloader.pro** et ajouter cette ligne **QT += opengl**

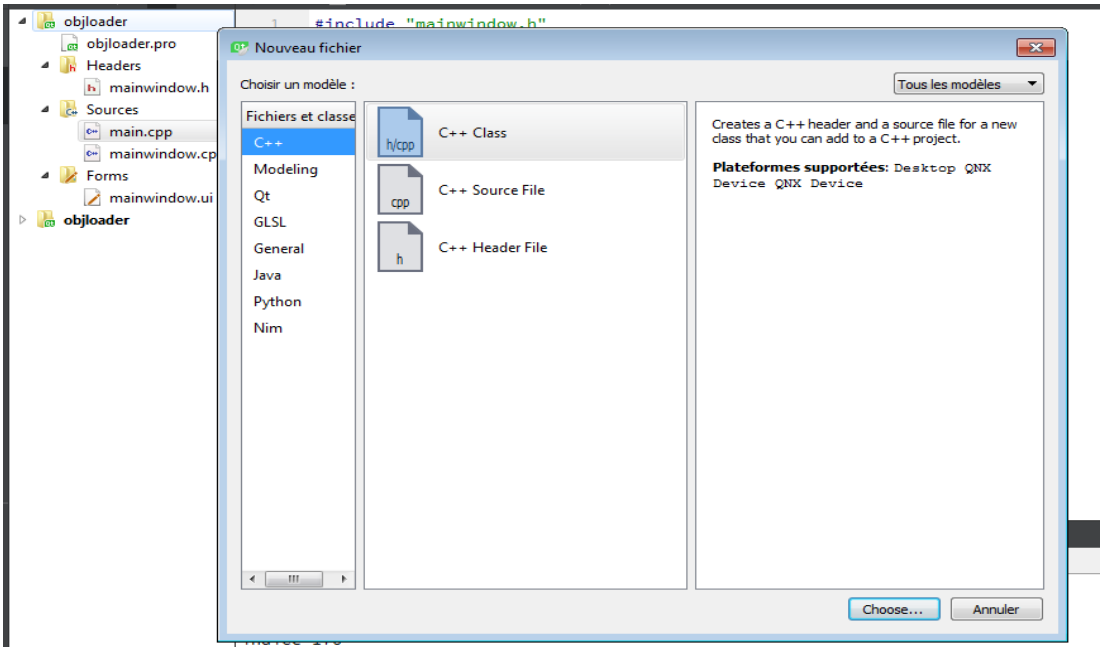
3.3.4 .création de la zone d'affichage :

Qt nous permet de créer la fenêtre qui contient la zone d'affichage OpenGL
Pour l'affichage d'objet 3D charger par le loader qu'on a développé , on ajoute une Widget en suivant les étapes suivantes :

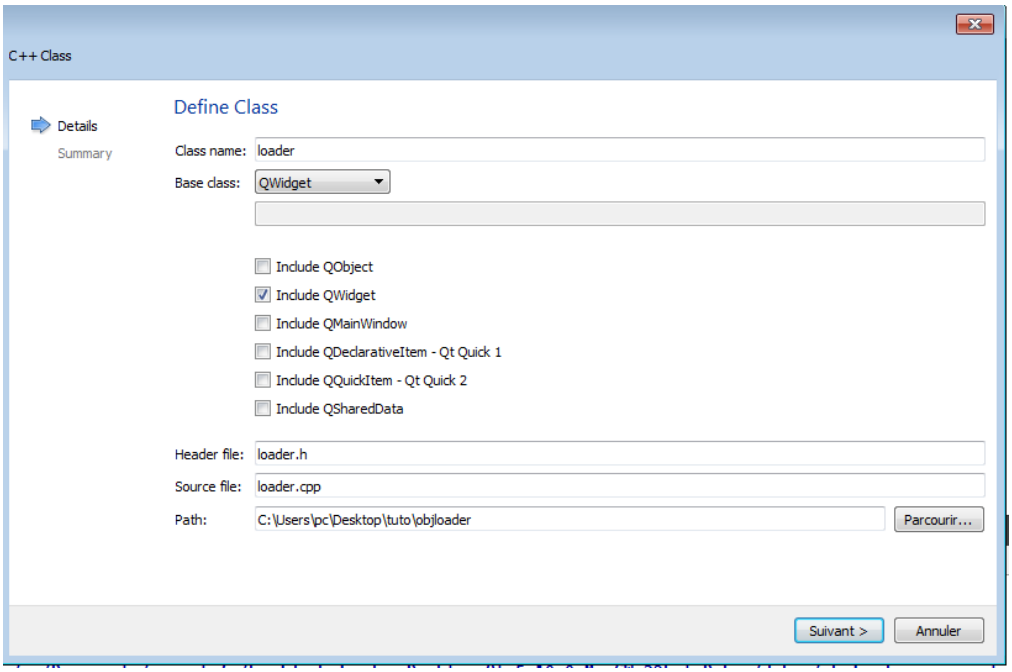
1. Commencez par glisser un Widget de notre liste de widgets vers notre fenêtre principale (voire l'image ci-dessous).



2. Maintenant ajouter une nouvelle classe et nommer **loader** car dans cette classe qu'on va développer notre chargeur de fichier obj , pour réaliser ça cliquer avec le bouton droit sur votre projet aller vers **Ajouter nouveau** puis cliquer sur **c++ class** (Voire l'image ci-dessous)



Autre fenetre s'affiche (voire l'image ci-dessous) ,choisir comme nom de la classe **loader** et comme classe de base **Qwidget**.



normalement la classe de base c'est **QGLWidget** qui herite de **QObject** et qui permet afficher des graphismes avec OpenGL mais ça n'est pas permet dans Qt 5.10.0 par contre pour les autre versions ,

Pour resoudre ce probleme j'ai choisi comme classe de base **Qwidget (QGLWidget** Hérite de **Qwidget)** puis j'ai fait des petites modéfications au niveux des fichier **loader.cpp** et **loader.h** (voire l'image ci-dessous)

```
#ifndef LOADER_H
#define LOADER_H

#include <QWidget>

class loader : public QWidget
{
    Q_OBJECT
public:
    explicit loader(QWidget *parent = nullptr);

};

#endif // LOADER_H
```

le fichier loader.h avant le changement

```
#ifndef LOADER_H
#define LOADER_H

#include <QGLWidget>

class loader : public QGLWidget
{
    Q_OBJECT
public:
    explicit loader(QWidget *parent = nullptr);


};
#endif // LOADER_H
```

le fichier loader.h après le changement

```
#include "loader.h"
#include <GL/glu.h>

loader::loader(QWidget *parent) : QWidget(parent)
{

}
```

Le fichier loader .cpp avant le changement

```
#include "loader.h"
#include <GL/glu.h>

loader::loader(QWidget *parent) : QGLWidget(parent)
{

}
```

Le fichier loader .cpp après le changement

QGLWidget fournit trois fonctions importantes que l'on peut réimplémenter dans la sous-classe **objloader** pour effectuer les tâches basiques d'OpenGL :

- **PaintGL() :** Affiche la scène OpenGL. Est appelé à chaque fois que le widget a besoin d'être mis à jour.
- **resizeGL()** : Définit la vue OpenGL, la matrice de projection, etc. Est appelé à chaque fois que le widget est redimensionné.
- **initializeGL() :**Définit le contexte de rendu OpenGL, crée les listes d'affichage, etc. Est appelé une seule fois avant que **resizeGL()** ou **paintGL()** ne soit appelé.

Voici le code minimal pour obtenir la fenêtre d'affichage :

C'est le fichier loader.h :

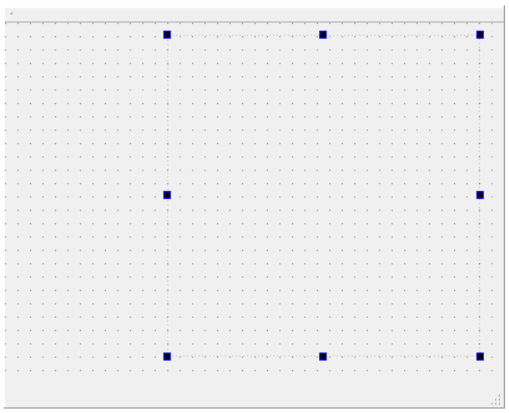
```
#ifndef LOADER_H
#define LOADER_H

#include <QGLWidget>
#include<GL/gl.h>
#include<GL/glu.h>

class loader : public QGLWidget
{
    Q_OBJECT
public:
    explicit loader(QWidget *parent = nullptr);

    void initializeGL();
    void paintGL();
    void resizeGL(int w,int h);
};
#endif // LOADER_H
```

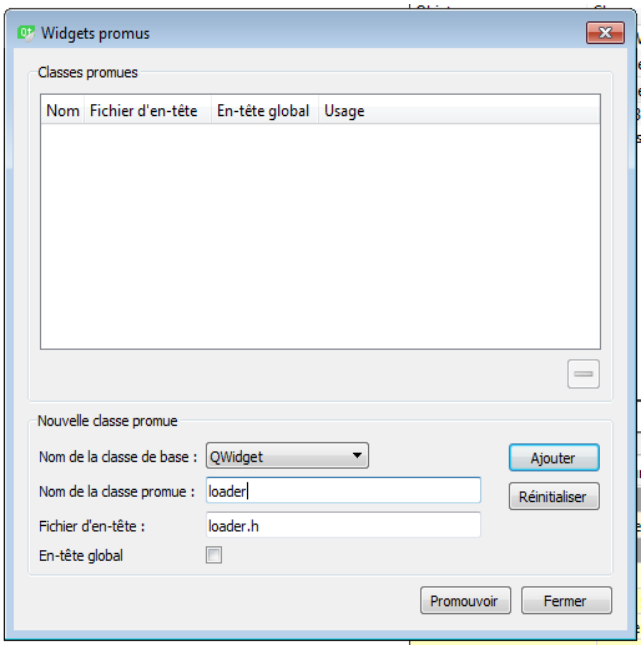
Voici le fichier loader.cpp avec l'explication :

```
#include "loader.h"

#include <GL/glu.h>

loader::loader(QWidget *parent) : QGLWidget(parent)
{

}

void loader::initializeGL()
{
   glClearColor(0,0,0,0);// Fond de l'écran en noir.

   glEnable(GL_DEPTH_TEST);// Activation des tests de profondeur du z-buffer

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
}


void loader::paintGL()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);// Nettoie l'écran
(buffer) avec les valeurs précisées par glClearColor (de même pour le depth
buffer)



  glMatrixMode(GL_MODELVIEW); // On charge la matrice du model view


    glLoadIdentity();

}
```

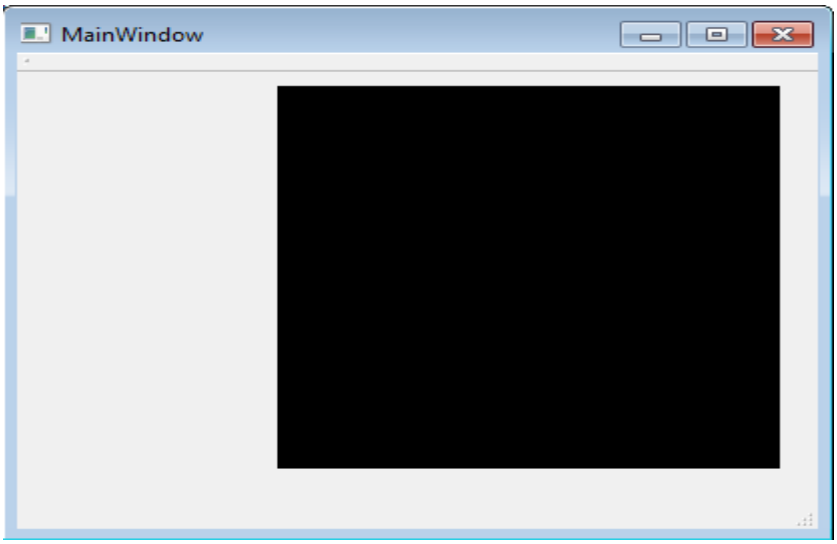
```
void loader::resizeGL(int w,int h)
{
glViewport(0,0,w,h);// Création de la taille du Viewport pour indiquer les
 frontières du cadrage actif

glMatrixMode(GL_PROJECTION);// On charge la matrice de projection

glLoadIdentity();//On l'initialise à la matrice identité.

gluPerspective(50,640.0/480.0,1.0,500.0);// On définit le mode de projection
dans la matrice

}
```

Apres l'implémentation de la classe **loader** il ne reste que d'aller à l'interface et cliquer avec le bouton droit sur le widget puis cliquer sur **promouvoir en..** et saisir le nom de la classe promue, cliquer sur **ajouter** ensuite sur **promouvoir** comme il' est montré sur l'image suivante.





Après l'exécution du code, une zone d'affichage s'affiche



3.4 .Le développement de l'application :

L'application qu'on a développé contient trois fonctions principales :

- La fonction **loadobject** () **:** consiste à lire le contenu d'un fichier .obj et le charger dans des vecteurs.
- La fonction **drew() :** cette fonction se charge à dessiner le modèle 3D en parcourant les donnés stockées dans les vecteurs et en utilisant le contexte opengl.
- La fonction **drewwired() :**cette fonction permet de visualiser le maillage de l'objet en effectuant un rendu en fils-de-fer.

3.4.1. Le chargement du fichier .ob

Etape 1 : déclaration des variables

Comme on a vu dans le premier chapitre , un fichier obj contient différents type de données c'est pour cette raison on a utilisé un ensemble de structures pour les représenter :

- Une structure **coordinate** pour représenter les vertex et les vecteurs normals

```
struct coordinate{
   float x,y,z;
       coordinate(float a,float b,float c); };
```

- Une structure **textcoord** pour représenter les textures

```
struct texcoord
 {
            float u,v;
            texcoord(float a,float b);
 };
```

- Une structure **face** pour représenter les faces soit triangulaire ou quadrangulaire

```
struct face{

          bool four;
          int fposition[4];
          int ftexcoord[4];
          int fnorm[4];



          face(int f1,int f2,int f3,int t1,int t2,int t3,int
               n1,int n2,int n3) ;

          face(int f1,int f2,int f3,int f4,int t1,int t2,int
               t3,intt4,int n1,int n2,int n3,int n4); };
```

- Une structure **material** pour représenter les données du fichier .mtl

```
struct material{
          std::string nom;
          float alpha,ns,ni;
          float dif[3],amb[3],spec[3];
          int illum;

        material(const char* na,float alphaa,float nss,float
        nii,float* diff,float* ambb,float* specc,int illumm); };
```

Après la déclaration des structures On doit définir des variables temporaires de type vecteur de pointeur dans lesquelles on stocke le contenu du fichier .obj :

```
std::vector<std::string*> coord;
std::vector<coordinate* >vertex;
std::vector<coordinate*> normals;
std::vector<texcoord*> texturecoordina
std::vector<face*>faces;
std::vector<material*> materials;
```

Etape 2 : l'ouverture du fichier .obj en lecture :

Cette partie de code permet d'envoyer un message d'erreur s'il ya un problème lors de l'ouverture du fichier, sinon on parcoure le fichier ligne après ligne du début jusqu'à la fin Et on mettre chaque ligne dans le vecteur **coord.**

```
std::ifstream in (filename);
    if(!in.is_open())

    {std::cout<<"impossible d'ouvrir le fichier"<< std::endl;

       return -1;
    }
    char buf[256];
    while(!in.eof())
    {in.getline(buf,256);
      coord.push_back(new std::string(buf));
    }
```

Etape 3 :la lecture des données a partir du vecteur coord :

Après l'étape de l'ouverture du fichier .obj en lecture, toutes les données maintenant sont stockées dans le vecteur **coord.**

Le but de cette étape est de lire les données à partir du vecteur **coord** et de les classer selon leurs types dans le vecteur convenable.

On parcoure le vecteur **coord** en utilisant une boucle **for**, plusieurs cas peut être différenciés pour faire la classification des données :

- Si la ligne commence par 'v' : if ((*coord[i])[0]=='v'&& (*coord[i])[1]==' ')

  La ligne commençant par 'v' représente une position donc on la met dans le vecteur **vertex** Avec l'instruction: vertex.push_back(new coordinate(tmpx,tmpy,tmpz));

- Si la ligne commence par 'vn' : if ((*coord[i])[0]=='v'&& (*coord[i])[1]==' n')

  La ligne commençant par 'v' représente le vecteur normal donc on la met dans le vecteur **normals** Avec l'instruction :

  normals.push_back(new coordinate(tmpx,tmpy,tmpz)); }

- les lignes commençant par 'vt' représente les coordonnées de texture on le met dans le vecteur **texturecoordina** avec la même méthode que 'vt' .
- les lignes commençant par 'f'
- les lignes commençant par "mtllib" :

if ((*coord[i])[0]=='m'&&(*coord[i])[1]=='t'&&(*coord[i])[2]='l'&&(*coord[i])[3]=='l')

si la ligne commence par mtllib (représente le nom du fichier mtl) on passe le nom du fichier .mtl à la classe **ifstream** et on fait la lecture du fichier avec la même méthode que le fichier obj et en charge son contenu dans un vecteur temporaire **tempmat** de type string similaire au vecteur **coord**

- si la ligne commence par "usemtl", on parcoure le vecteur **materials** pour chercher l'index correspondant à ce nom du matériel :

```
for(int i=0;i<materials.size();i++)
      {
           if(strcmp(materials[i]->nom.c_str(),tempnamemat)==0)
                              indexmat=i;
                              break;     }
```

- si la ligne commence par 'f '': if ((*coord[i])[0]=='f') :

Le traitement des faces est un peut plus complexe par rapport aux autres données

En premier temps on teste si la face est de type triangulaire ou bien quadrangulaire en calculant le nombre d'espace dans la ligne :

`f v0/vt0/vn0 v1/vt1/vn1 v2/vt2/vn2` : 3 espace ⇒triangulaire

`f v0/vt0/vn0 v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn4:` 4 espace ⇒ quadrangulaire

Voici le code qui permit de différencier entre les types des faces :

- **le cas d'un maillage quadrangulaire**

```
if (count(coord[i]->begin(),coord[i]->end(),' ')==4)
```

Les attributs **vt** et **vn** sont optionnels mais **v** (position) est obligatoire, il est donc possible de représenter un sommet par :

- `v` : position seule,
- `v/vt` : position + coordonnées de texture,
- `v//vn` : position + normale,
- `v/vt/vn` : position + coordonnées de texture + normale.

Donc pour savoir comment sont représentés les sommets on fait des tests imbriqués dans le test précédent :

Exemple : Représentation des sommets par position + coordonnées de texture + normale

```
if(coord[i]->find("/")!=std::string::npos)
   {
     int ft[4];
sscanf(coord[i]->c_str(),"f %d/%d/%d %d/%d/%d %d/%d/%d %d/%d/%d"
,&fp1,&ft[0],&fn1,&fp2,&ft[1],&fn2,&fp3,&ft[2],&fn3,&fp4,&ft[3],&fn4
);
faces.push_back(newface(fp1,fp2,fp3,fp4,ft[0],ft[1],ft[2],ft[3],fn1,fn
2,fn3,fn4)); }
```

Exemple : Représentation des sommets par position + normale

```
   if(coord[i]->find("//")!=std::string::npos)
{sscanf(coord[i]->c_str(),"f %d//%d %d//%d %d//%d
%d//%d",&fp1,&fn1,&fp2,&fn2,&fp3,&fn3,&fp4,&fn4 );
 faces.push_back(new face(fp1,fp2,fp3,fp4,0,0,0,0,fn1,fn2,fn3,fn4)); }
```

Et continuer avec les autres cas de représentation des sommets.

- **le cas d'un maillage triangulaire**

Pour un maillage avec des faces triangulaires on applique les mêmes instructions qu'on a utilisées dans le cas d'un maillage quadrangulaire.la seul différance c'est d'utiliser le constructeur des faces avec trois sommets.

Exmple :

```
else if(coord[i]->find("/")!=std::string::npos)
  {int ft[3];
   sscanf(coord[i]->c_str(),"f %d/%d/%d %d/%d/%d %d/%d/%d" ,&fp1
,&ft[0],&fn1,&fp2,&ft[1],&fn2,&fp3,&ft[2],&fn3 );
 faces.push_back(new face(fp1,fp2,fp3,ft[0],ft[1],ft[2],fn1,fn2,fn3)); }
```

3.4.2. Le rendu du fichier chargé :

Après le chargement du fichier obj vient l'étape de l'affichage de l'objet 3d ce qu'on appelle le rendu.

Pour cette étape on a développé deux fonctions :

a. la fonction drew() :

fonction **drew()** prend en charge à afficher les maillages quadrangulaires et triangulaires.

Cette fonction permet le parcoure du vecteur **faces** et d'utiliser les index stockés pour accéder aux coordonnées de position (vertex) ,de texture (texturecoordina) et les coordonnées du vecteur normal(normals) puis dessiner l'objet point par point (sommet par sommet) en utilisant opengl :

Le dessin d'une primitive se fait entre glBegin() et glEnd();

La fonction glBegin() prend le paramètre `GL_QUADS` pour un maillage quadrangulaire

Et `GL_TRIANGLES` pour un maillage triangulaire :

```
glBegin(GL_QUADS)

           //sommet1

          glNormal3f(normals[faces[i]->fnorm[0]-1]-
>x,normals[faces[i]->fnorm[0]-1]->y,normals[faces[i]->fnorm[0]-1]->z);

          glVertex3f(vertex[faces[i]->fposition[0]-1]-
>x,vertex[faces[i]->fposition[0]-1]->y,vertex[faces[i]->fposition[0]-
1]->z);
                          .
                          .
                          .
                          .

             //sommet3
          glNormal3f(normals[faces[i]->fnorm[3]-1]-
>x,normals[faces[i]->fnorm[3]-1]->y,normals[faces[i]->fnorm[3]-1]->z);
          glVertex3f(vertex[faces[i]->fposition[3]-1]-
>x,vertex[faces[i]->fposition[3]-1]->y,vertex[faces[i]->fposition[3]-
1]->z);

          glEnd();
```

b.la fonction drewwired() :

cette fonction permet de dessiner le maillage de l'objet (effectuer un rendu en fils-de-fer)

elle contient les même instructions que **drew()** avec une seule différence c'est d'utiliser :

glBegin(GL_LINE_LOOP) .

3.4.3. Le déplacement des vertex avec la souris :

L'algorithme de déplacement des vertex consiste à calculer le point le plus proche à la position du curseur (distance minimal) en parcourant le vecteur qui contient tous les points constituant le maillage et en calculant la distance entre chaque point et le curseur ,donc le point le plus proche au curseur va être déplacer .

C'est la calasse QMouseEvent qui permet la gestion de la souris , elle contient un ensemble Des gestionnaires d'événements (méthodes) qui assurent la programmation événementielle tels que mousePressEvent() , mouseMoveEvent(), les coordonnées de la position du curseur de souris au moment de l'appui peuvent être obtenues au moyen des méthodes x() et y().

3.5. L'exécution du programme :.

- Après le chargement du fichier obj l'affichage se fait à travers la procédure **drew(),** La prise en compte des coordonnées des vecteurs normales lors du rendu permet une très bonne exploitation des effets de la lumière donc les traits de l'objet sont plus claire (voir figure 3.3), par contre la négligence des vecteurs normales Donne une image floue (voir figure 3.4) .

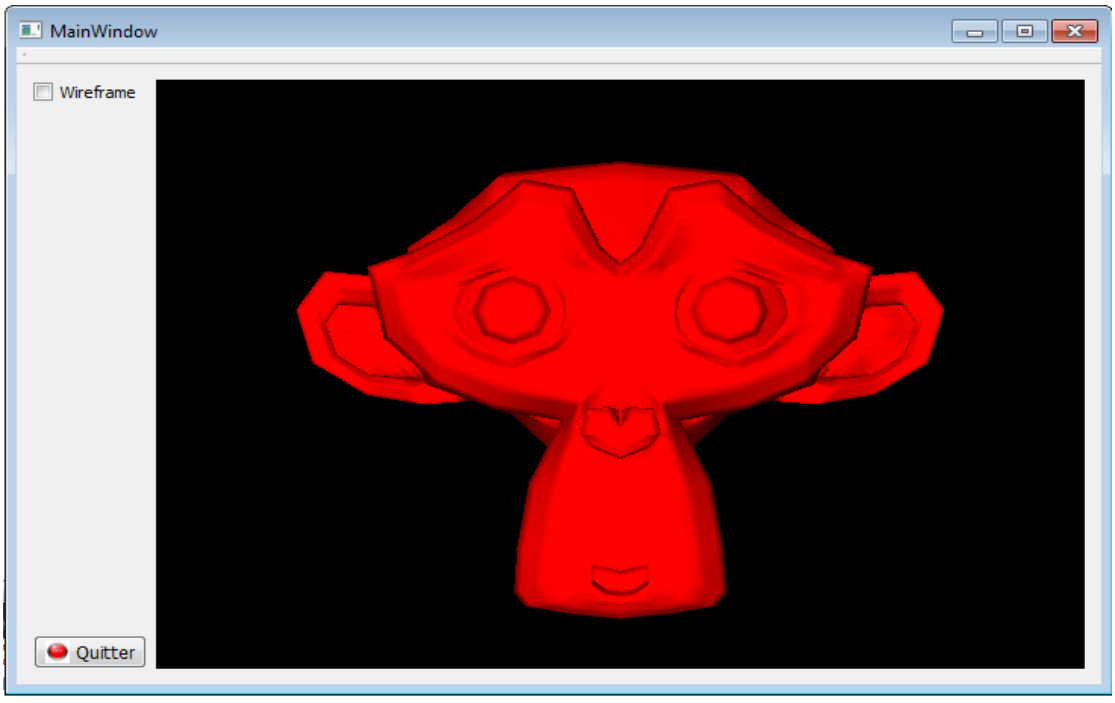


Figure 3.3:le rendu d'un fichier obj en lisant les vecteurs normales

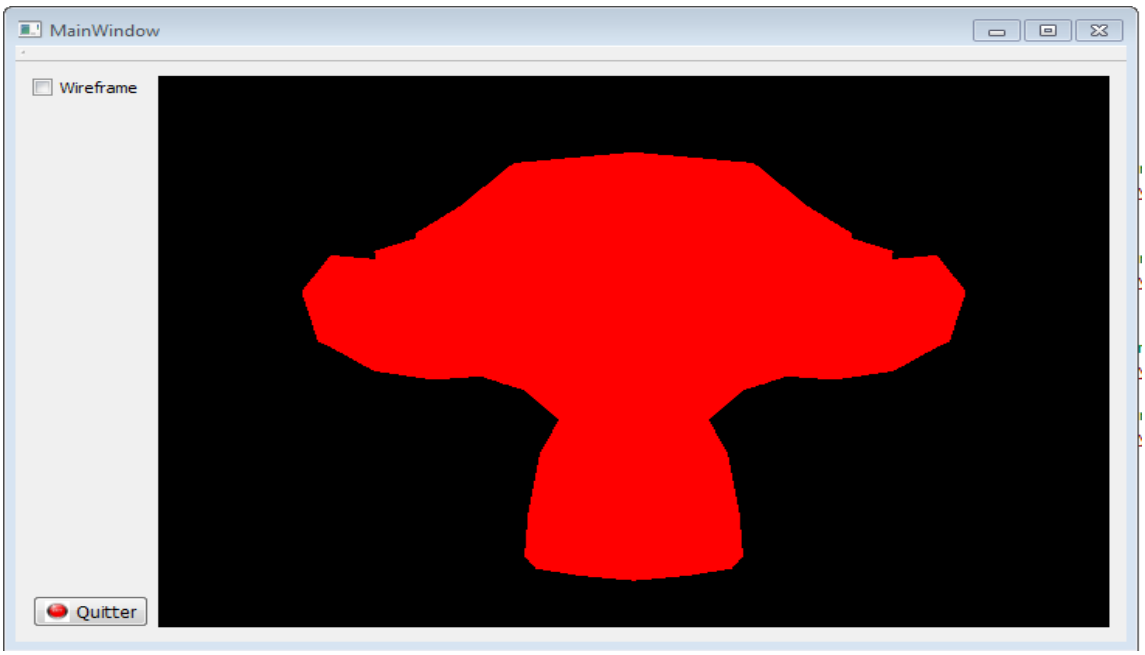


Figure3.4:le rendu d'un fichier obj sans la lecture des vecteurs normales

- Une coche sur le widget check box permet l'execution de la procedure **drewwired**() qui donne un rendu en fils-de-fer (voir les figures 3.5 et 3.6)

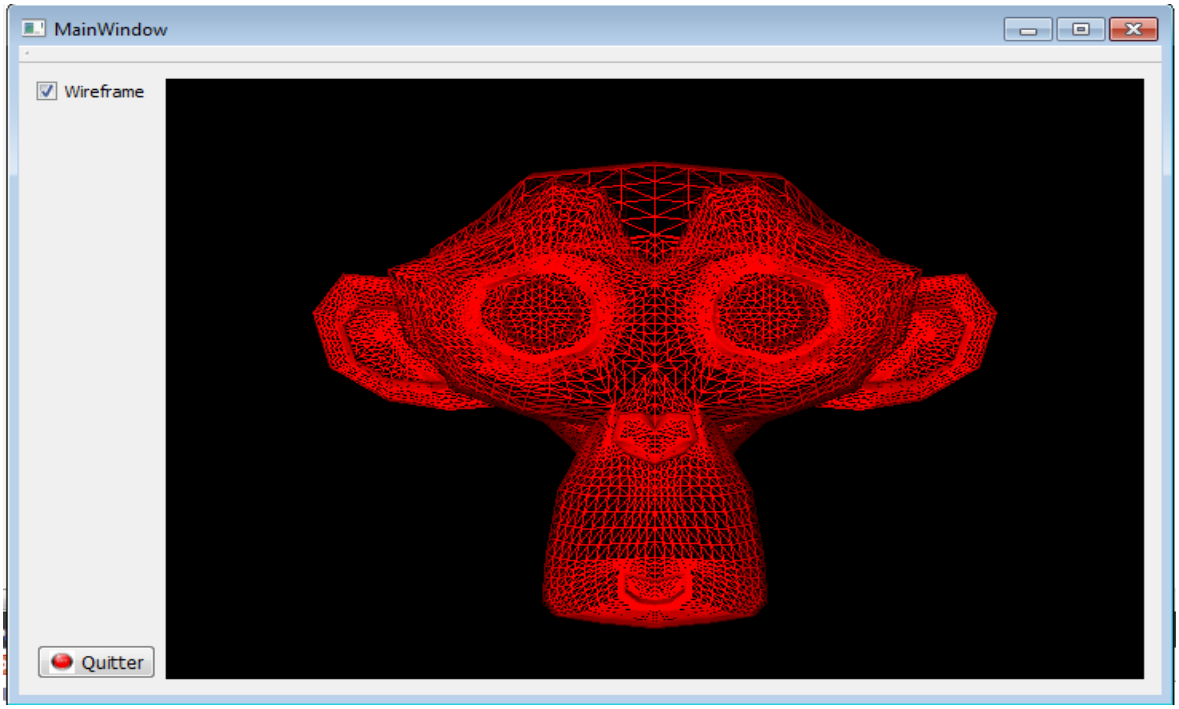


Figure 3.5:le rendu d'un objet 3d en fils de fer

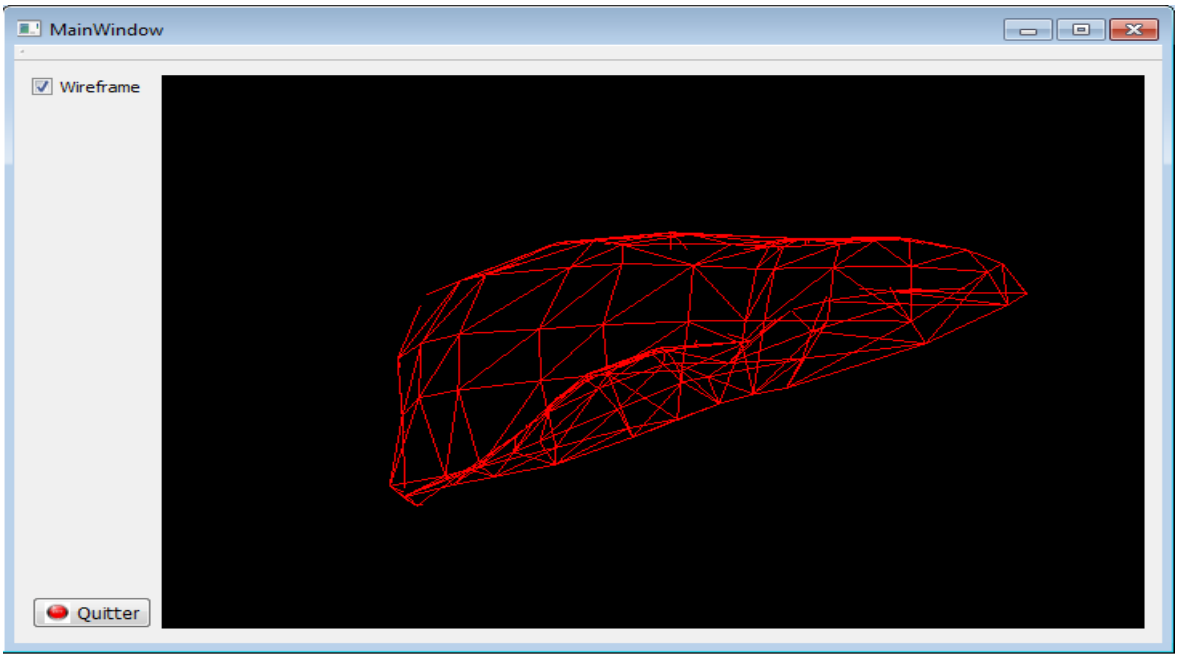


Figure3.6:Le maillage du foie

- Une simple clique sur le maillage et un glissement de la souris avec un bouton gauche enfoncé permettent le déplacement du vertex le plus proche au clique de la souris ce qui donne une déformation du maillage (voir figure 3.7 et 3.8)

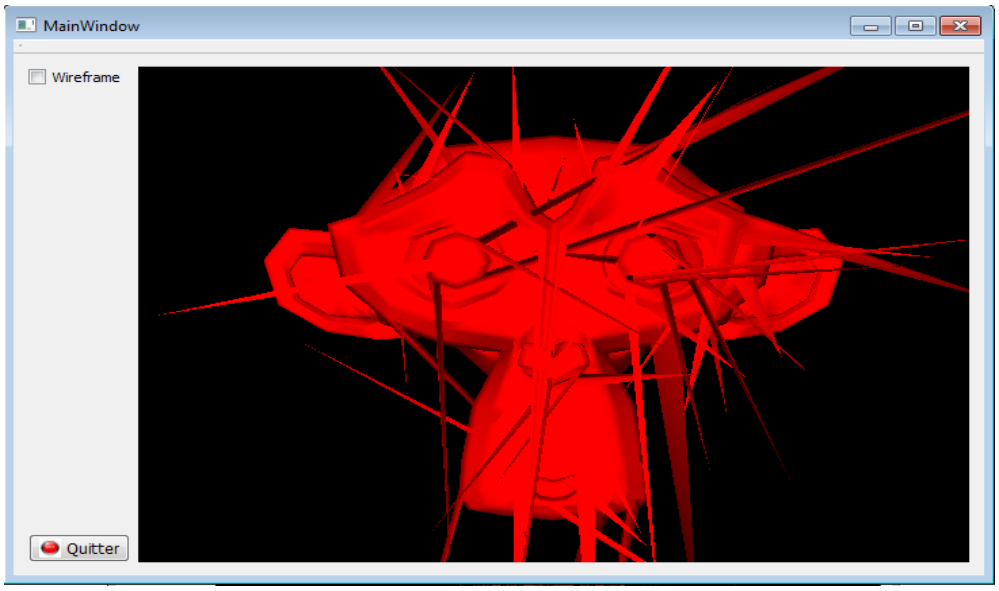


Figure3.7:objet 3d après le déplacement des vertices

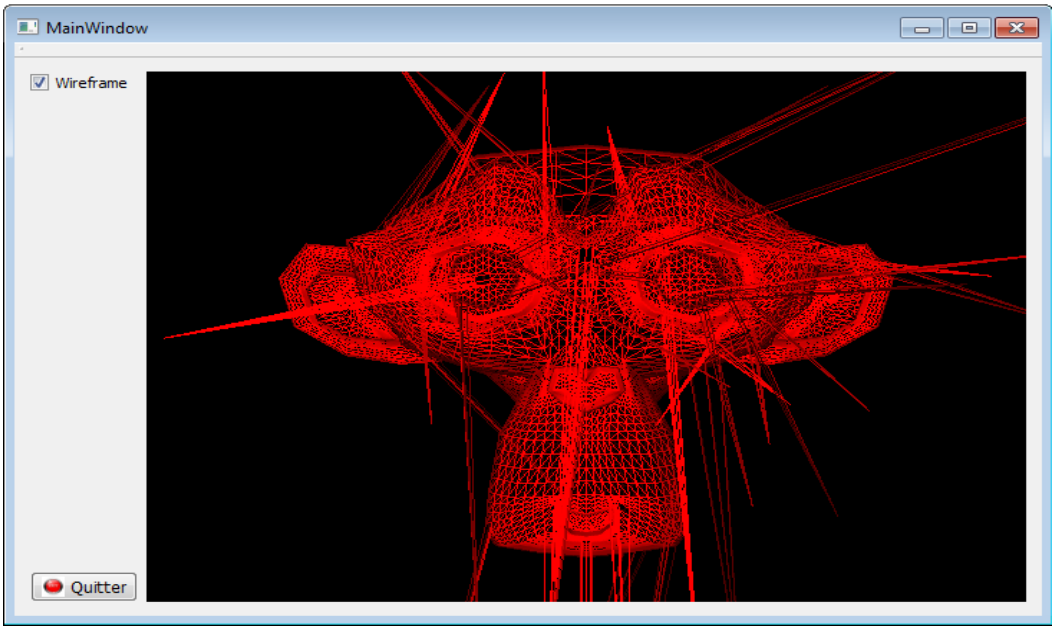


Figure 3.8:le maillage d'un objet 3d après le déplacement des vertices

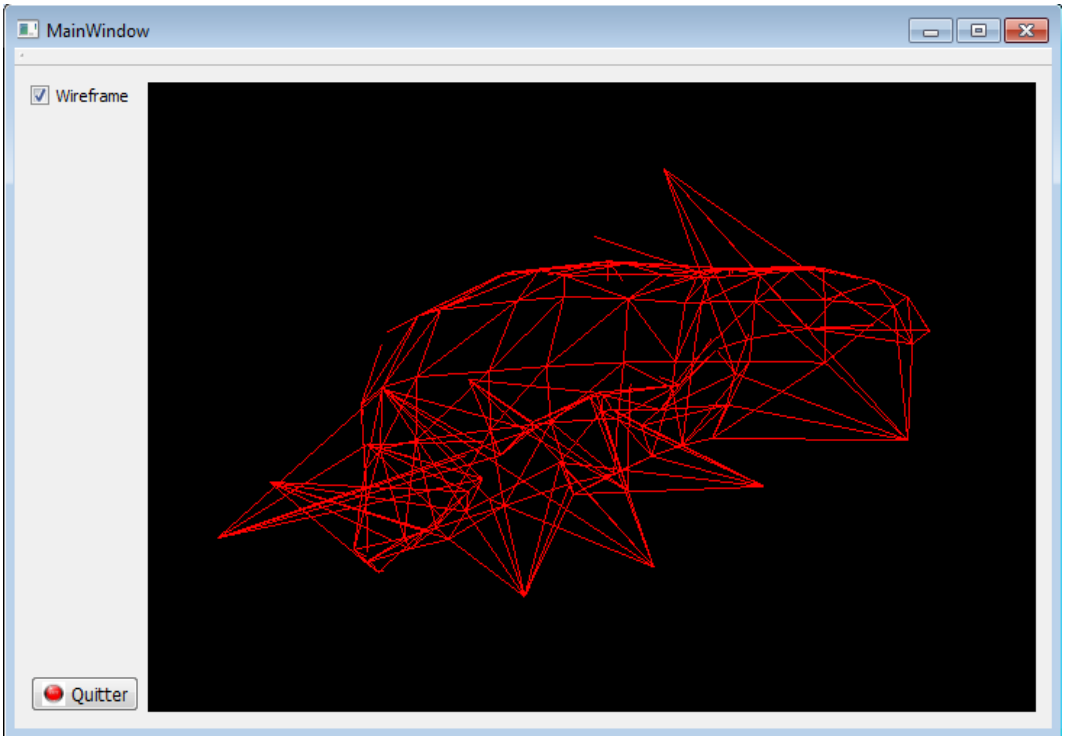


Figure 3.9:Le maillage du foie après le déplacement des vertices

3.6. Conclusion :

Dans ce chapitre nous avons démontré notre choix du Qt, Opengl et du langage c++,de plus nous avons expliquer les différentes étapes de l'installation et de préparation de l'environnement de développement .

Nous avons aussi montré l'implémentation de chaque composant de notre programme afin d'assurer une présentation claire et détaillée de notre outil de visualisation.

Finalement nous avons donné quelques résultats d'exécution du programme en utilisant des captures d'écran.

Conclusion générale :

Au cour de ce travail on a présenté une étude plus au moins détaillée sur les notions de base de la 3d tel que la modélisation, les éléments constituant un maillage, les différentes techniques pour représenter et manipuler un objet 3d .

Concernant la partie de visualisation ou bien le rendu on a essayé d'expliquer d'une manière un peu simple l'algorithme de lancer des rayons et les phénomènes de la physique de la lumière et de l'optique géométrique qui interviennent lors de l'étape de rendu.

Tenant compte que l'outil développé sert à charger des fichiers obj une étude complète est faite sur ce type des fichiers.

Parmi les objectifs visés dans ce projet est la conception d'un outil de visualisation en temps réel exécutable sur les différents systèmes d'exploitation, l'utilisation des deux bibliothèques multiplateforme Qt et Opengl nous a offert les moyens pour atteindre notre objectif

Le choix du langage d'implémentation et la préparation de l'environnement de développement jouent un rôle important dans la construction de l'application. C'est pour cette raison qu'on a consacré une grande partie du troisième chapitre pour parler de ceci.

Le loader développé permet de charger seulement les fichiers obj et de visualiser les objets 3d non composés donc comme perspectives on souhaite de compléter le développement du loader on lui donne la possibilité de :

Charger le fichier .mtl .

Lire les différentes extensions des fichiers 3d .

Charger une sciène 3d (plusieurs objets) .

Régler la proportionnalité entre le glissement de la souris et le déplacement des vertex afin de mieux contrôler la déformation de l'objet 3d en utilisant la technique de déplacement des vertex avec la souris.

Bibliographie :

[1] : L'Infographie 3D en Médecine TPE réalisé par Andrawes Al Bahou, Eyasse Al Battah, et Nicolas Moulin

[2] : https://www.reviversoft.com/fr/file-extensions/obj

[3] : https://www.institut-grasset.qc.ca/actualites/la-decouverte-du-rendu-et-des-modeles-3d-pour-les-etudiants-en-cours-danimation-3d

[4] : Mémoire d'ingéniorat en aéronautique 2011 : Calcul des performances aérodynamiques de la configuration aile-fuselage Ara M100 par maillage hybride , Mourad BOUDJERIS Université Saad Dahleb (Blida)

[5] : Thèse de doctorat :Modèles de maillages déformables 2D et multi résolution surfaciques 3D sur une base d'ondelettes :Sébastien Valette

[6] : université de Bretagne sud école d'ingénieur ENSIBS Mécatronique 2eme année:création de différents éléments sous Gmsh pour l'utilisation du maillage s sous Herezeh

[7] : Rapport de stage Vers la simulation aux grandes échelles sur maillage hybride : Pierre CAYOT

[8] : mémoire présenté en vue de l'obtention Du diplôme de maîtrise des sciences appliquées : comparaison de la qualité des simulations d'écoulements visqueux turbulents sur différents maillages d'hexaèdres : piaget daniel

[9] : THÈSE de doctorat : Modélisation et implémentation de parallélisme implicite pour les simulations scientifiques basées sur des maillages Hélène COULLON 2014

[10] : J. Vince.Mathematics for Computer Graphics. Springer, 2010.

[11] : cour informatique graphique : www-ihm.lri.fr/~mbl/ENS/IG1/COURS/11-TransfoGeom.pdf

[12] : https://www.d-booker.fr/

[13] : https://openclassrooms.com/courses/charger-des-fichiers-obj

[14] : https://www.cs.utah.edu/~boulos/cs3505/obj_spec.pdf

[15] : https://www.fileformat.info/format/material/index.htm

[16] : DESS Génie Informatique : Analyse et Synthèse d'Images : James L. Crowley

[17] : Mémoire Présenté pour l'obtention du diplôme de MAGISTER en Électronique , Reconstitution d'une scène 3D à partir d'indices visuels 2D :BRIOUA Abdelkrim Université Batna

[18] : Illumination François Faure 24 février 2009

[19] : IFT3730: Infographie 3D Illumination locale Pierre Poulin, Derek Nowrouzezahrai Hiver 2013 DIRO, Université de Montréal

[20] : " Rendu réaliste http://morpheo.inrialpes.fr/people/Boyer/Teaching/Master/c7.pdf

[21] :https://openclassrooms.com/courses/programmez-avec-le-langage-c/introduction-a-qt

[22] : http://igm.univ-mlv.fr/~dr/XPOSE2008/Le%20Framework%20Qt/utils.html

[23] : https://www.wikizero.com/fr/Qt

[24] : https://qt.developpez.com/outils/

[25] :Qt Creator : http://qt-project.org/doc/qtcreator-2.5/creator-tool-chains.html

[26]: projet.eu.org/pedago/sin/term/5-Qt.pdf

[27] : Thèse de doctorat : Visualisation interactive de simulations à grand nombre d'atomes en physique des matériaux : SIMON LATAPI : école centrale PARIS

[28] :TP OpenGL 6 Eclairage en OpenGL :Licence Informatique 3 ème année :Année 2017-2018 :Université du LITTORAL COTE D'OPALE

[29] :TP OpenGL 3 Création d'objets Licence Informatique 3 ème année : année 2017-2018 : Université du LITTORAL COTE D'OPALE

[30] :Fabrice Aubertfabrice.aubert@li.frLicence/MasterUSTL - IEEA2008-2009: Chapitre IV : Les textures Programmation 3D

[31] : Mémoire de fin d'études pour l'obtention du diplôme de Licence en Informatique : Système de visualisation 3D AVEC OpenGL pour la navigation dans un environnement fermé (cas d'un labyrinthe),Mr MOUALEK Djalloul Youcef

[32] : Petite documentation d'OpenGL *Université Montpellier M. Hascoët II 2004/2005*

[33] :https://cpp.developpez.com