Représentation d'objets en 3 dimensions

Franck Trey & Florian Lefevre

19 mai 2008

Table des matières

Première partie

Description

Chapitre 1

Principe de réprésentation

Le but de ce projet est de représenté des objets en 3 dimensions. Pour cela nous disposons d'un ensemble de polygones disposé dans l'espace. Nous devons donc projeter ces polygones sur l'écran en fonction de l'angle de vue.

Ici ne sont représenté que des éléments polygonales. Toute forme plus complexe tel que les courbes, les sphères, etc... sont dissociés en polygones.

Les polygones s'afficheront en couleur si les informations sur la couleur sont données.

Chapitre 2

Base de travail

Le programme peut créer des scènes d'objets ou de simples objets 3d à partir de 2 types de fichiers.

2.1 Fichier OFF

Un fichier OFF (Object File Format) est un fichier contenant un simple object 3d. Chaque information peut séparé par un plusieurs espaces. Les commentaires dans le fichier sont marqués dans le fichier par le symbole # Il se compose en 3 principales parties.

2.1.1 L'entête

L'entête contient le code d'identification du type du fichier. "OFF" dans le cas présent. S'ensuit trois nombres qui indiquent respectivement le nombres de sommets, le nombre de polygones ainsi que le nombre de bord dont se compose l'objet 3d. Le nombre de bord doit obligatoirement être précisé même si ça valeur importe peu.

Par exemple :

```
OFF
8 6 12
```

Cet exemple correspond à un fichier OFF contenant 8 vertices (sommets), 6 polygones et 12 bords.

2.1.2 Liste des vertices

La liste des vertices suit immédiatement après l'entête. Chaque sommet est représenté par trois coordonnées représentant le point dans l'espace. Chaque coordonnées correspond respectivement à l'abcisse (x), l'ordonnée (y) et la profondeur (z) du point dans l'espace. Les coordonnées sont des nombres flottants.

Par exemple :

```
-1.632993   0.000000   1.154701
```

Cet exemple correspond au point d'abcisse -1.632993, d'ordonnée 0 et de profondeur 1.154701.

2.1.3 Liste des polygones

De la même manière que la liste des vertices suit l'entête, la liste des polygones suit la liste des vertices. Chaque polygone est représentés par plusieurs informations :

Nombre de vertices : Chaque ligne commence par le nombre de vertices dont se compose le polygone.

Indices des sommets : Chaque vertices est résenté ici par son index. Si il y a N vertices, les valeurs possibles de ces index se trouve sur la plage 0 ... $N - 1$.

Information sur la couleur : Les informations sur la couleur sont optionnels, elles peuvent ne pas être indiqué. Toutefois si elles le sont, elles suivent le modèle RGBA (Rouge, Vert, Bleu, Alpha). Il existe dans ce cas deux modèles possibles :

- Chaque composante prend une valeur entre 0 et 255 et est donc représenté par un entier.
- Chaque composante prend une valeur entre 0 et 1 et est donc représenté par un flottant.

Par exemple :

```
4  17   5   9   4   1.0   0.0   0.0   1.0
```

Cet exemple représente un polygone a 4 faces composé des points 17, 5, 9 e t4 de la liste des vertices. Ce polygone est visible à 100% et est exclusivement composé de rouge.

2.2 Fichier SFF

Les fichiers SFF (Scene File Format) ne sont pas standard et ont été créer ici pour représenter des scènes d'objets 3d, c'est à dire des scènes qui peuvent contenir plusieurs objets 3d. De même que pour les fichiers OFF, les commentaires sont marqués par un #. Un fichier SFF se compose en deux parties :

2.2.1 L'entête

L'entête contient essentiellement deux informations. Il s'agit tout d'abord du mot-clé "SFF" qu idique que l'on a affaire à un fichier du type SFF. S'ensuit ensuite le nombre d'objet 3d contenu dans la scène.

Par exemple :

```
SFF
7
```

Cet exemple représente l'entête d'un fichier contenant une scène 3d composé de 7 objets 3d.

2.2.2 Liste des objets 3d

Cette liste contient les chemins vers les fichiers OFF associés aux objets 3d. Ces chemins sont relatifs et sont situés par rapport au fichier SFF. S'ensuit également les coordonnées du centre des objets 3d dans l'espace de la scène d'objets 3d.

Par exemple :

```
../off_file/cube.off 0.000000  -13.63299  -21.15470
```

Cet exemple illustre la représentation d'un objet 3d contenu dans le fichier OFF cube.off. Cet objet se situant aux coordonnées 0.000000 -13.63299 -21.15470.

2.3 Les repères

Dans le programme ainsi que dans le reste de ce document, nous aurons besoin d'utiliser plusieurs repères différents afin de pouvoir projeter les poygones contenus dans le fichier à l'écran.

Repère R_0 : Il s'agit du repère de base du fichier SFF. C'est à dire le repère de la scène d'objet 3d.

Repère R_1 : Le repère R_1 est le repère de base des fichiers OFF. C'est le repère propre de chaque objet 3d.

Repère R_2 : Le repère R_2 est propre à chaque objet 3d et son origine correspond au barycentre de l'objet 3d.

Repère R_3 : Il s'agit ici du repère de la vue 3d, c'est ce repère qui indique l'angle de vue avec lequel il faut projeter l'objet 3d. Il s'agit en effet de la transformation du repère sphérique S_1 en repère cartésien.

Repère R_4 : Le repère R_4 n'est autre que le repère projeté de R_3 sur l'écran. Il s'agit donc d'un repère composé de deux axes pour deux dimensions. Il nous sert ici à avoir la notion de distance de l'objet.

Repère R_5 : Il s'agit du repère qui est utilisé par le module QT pour l'affichage. Le point d'origine correspond à l'angle supérieur droit de l'écran.

Repère S_0 : Le repère S_0 n'est autre que le repère de la scène 3d en coordonnées sphériques. Il est donc confondus avec le repère cartésien R_0.

Repère S_1 : Le repère S_1 correspond au repère cartésien R_3. Il a subit une rotation d'angle θ et ϕ par rapport au repère S_0 mais à même origine que ce dernier. Il nous permet donc ici de mettre en valeur la rotation de l'objet 3d dans l'espace.

Chapitre 3

Projections mathématiques

Nous distinguerons 2 grandes catégories. Tout d'abord les projections qui seront faites une seule fois à la création de l'objet 3d. Ensuite nous verrons les projections qui seront faites à chaque mise à jour de l'écran. Pour alléger les formules, nous ne projetons ici qu'un point A qui a pour coordonnées (x_1, y_1, z_1) dans le repère R_1. Ce point A est un point quelconque du polygone composé de n+1 points de A_0 à A_n.

Un polygone étant un ensemble de point, le projection qui s'applique au point A et la même que celle qui s'applique aux polygones.

3.1 Création de l'objet 3d

Lors de la création de l'objet 3d, nous disposons, dans notre fichier OFF, d'un repère (R_1). L'objectif est de créer un nouveau repère R_2 spécifique à l'objet 3d. Puis d'exprimer les coordonnées du point A dans le repère R_0 de la scène 3d. Pour cela nous avons choisit le centre de l'objet comme origine du nouveau repère R_2.

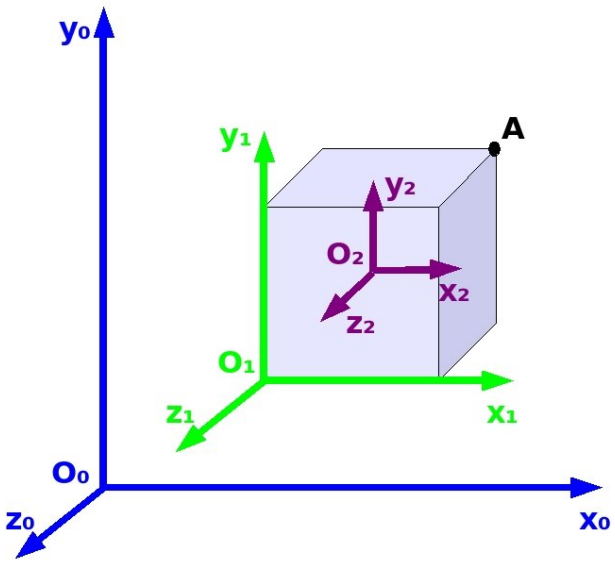

FIG. 3.1 – Point A dans R_0, R_1 et R_2

3.1.1 Préliminaires

A la lecture des fichiers OFF ou SFF, nous disposons des informations suivantes :

- coordonnées du point A dans R_1 : $(x_{A_1}, y_{A_1}, z_{A_1})$;
- coordonnées du centre G de l'objet dans R_0 : $(x_{G_0}, y_{G_0}, z_{G_0})$;

3.1.2 Calcul de l'origine de R_2

Ce repère (R_2) a donc comme origine le barycentre G de l'objet 3d et ses axes ont même direction que ceux du repère R_1.

Soit G un unique point de coordonnées $(R_1, x_{g_1}, y_{g_1}, z_{g_1})$ tel que :

$$\sum_{i=0}^{n} a_i \overrightarrow{A_i G}$$

G est l'isobarycentre de l'ensemble des n points $A_1 ... A_n$. C'est à dire que tout les points $A_1 ... A_n$ ont le même poid. G a donc pour coordonnées dans R_1 :

$$\begin{cases} x_{g_1} = \frac{\sum_{i=1}^{n} x_{A_i}}{n} \\ y_{g_1} = \frac{\sum_{i=1}^{n} y_{A_i}}{n} \\ z_{g_1} = \frac{\sum_{i=1}^{n} z_{A_i}}{n} \end{cases}$$

3.1.3 Coordonnées de A dans le repère R_2

Le point A suit donc une translation suivant le vecteur $\overrightarrow{O_1G}$:

$$\begin{cases} x_2 = x_1 + x_{g_1} \\ y_2 = y_1 + y_{g_1} \\ z_2 = z_1 + z_{g_1} \end{cases}$$

Le point A a donc pour coordonnées dans le repère R_2 : (x_2, y_2, z_2)

3.1.4 Coordonnées de A dans le repère R_0

Avant de projeter le point A sur l'écran, nous devons calculer ces coordonnées dans le repère R_0 qui est le repère de la scène 3d. Nous distinguons deux cas :

- la scène 3d est composé de plusieurs objets 3d ;
- la scène 3d est composé d'un unique objet 3d ;

Unique objet 3d

Si la scène contient un unique objet 3d, nous prenons le centre de l'objet 3d G dans le repère R_2 comme point d'orgine O_0 du repère R_0. Dans ce cas, les coordonnées du point A dans R_0 sont les même que celle dans le repère R_2 puisque les deux repères sont confondus. Et le point G prend donc pour coordonnées dans R_0 : $(0, 0, 0)$.

Ensemble d'objets 3d

Si la scène comporte plusieurs objets 3d, nous disposons alors, lors du chargement du fichier de scène SFF d'une information supplémentaire sur l'objet. Il s'agit des coordonnées de son centre G : $(x_{G_0}, y_{G_0}, z_{G_0})$ dans le repère de scène R_0. Il suffit alors de faire une simple translation suivant le vecteur $\overrightarrow{O_2O_0}$ pour trouver les coordonnées du point A dans R_0. Soit A ayant pour coordonnées (x_2, y_2, z_2) dans R_2 et (x_0, y_0, z_0) dans R_0

$$\begin{cases} x_0 = x_2 + x_{G_0} \\ y_0 = y_2 + y_{G_0} \\ z_0 = z_2 + z_{G_0} \end{cases}$$

3.1.5 Résumé des calculs

$$\begin{aligned} A \quad : \quad & \begin{cases} x_0 = x_2 + x_{G_0} \\ y_0 = y_2 + y_{G_0} \\ z_0 = z_2 + z_{G_0} \end{cases} \\ \Leftrightarrow \quad & \begin{cases} x_0 = x_1 + x_{g_1} + x_{G_0} \\ y_0 = y_1 + y_{g_1} + y_{G_0} \\ z_0 = z_1 + z_{g_1} + z_{G_0} \end{cases} \end{aligned}$$

3.1.6 Conclusion

Nous disposons donc ici des coordonnées $(x_{A_0}, y_{A_0}, z_{A_0})$ du point A dans le repère R_0. Ce sont ces coordonnées qui seront stocké dans le programme afin d'être projeter à l'écran.

3.2 Projection à l'écran

Les calculs pour la projection des points à l'écran se fait en temps réels. Notre objectif est ici de calculer les coordonnées du point A dans le repère R_5 afin qu'il puisse être afficher à l'écran. Pour ce faire, nous allons calculer les coordonnées de ce point dans

le repère S_0 : Ce repère correspond à la transformation du repère cartésien R_0 en repère sphérique.

le repère S_1 : Ce repère a la même origine que le repère S_0 mais dont les axes sont orientés différement. Ce qui nous permet de controler aisément l'angle de vue sur l'objet à partir d'un simple point de référence (l'origine de la vue V).

le repère R_3 : Ce repère correspond à la transformation du repère sphérique S_1 en repère cartésien.

le repère R_4 : Ce repère correspond à la projection du point A sur l'écran. Il nous permet de gérer la notion de distance de l'objet par rapport à l'origine de la vue qui se trouve au centre de l'écran.

le repère R_5 : Ce repère est le système de coordonnées utilisé par QT et nous permet donc d'afficher le point A directement à l'écran.

Remarque : Nous partons ici des coordonnées du point A dans un repère cartésien car il s'agit du moyen le plus simple pour représenter tout les points à n'importe quel moment des calculs. Nous passons ensuite temporairement dans un repère sphérique pour revenir dans un repère cartésien. Ce changement de base de repère nous permet de calculer simplement la rotation en fonction des coordonnées du point V dans le repère sphérique S_0.

3.2.1 Préliminaires

Nous avons comme base de départ :
- coordonnées de A dans R_0 : $(x_{A_0}, y_{A_0}, z_{A_0})$;
- coordonnées sphériques de V dans S_0 : $(r_{V_0}, \theta_{V_0}, \phi_{V_0})$;

Nous cherchons donc à calculer les coordonnées du point A dans le repère R_5 en fonction de ses deux paramêtres.

3.2.2 Calcul de A dans le repère sphérique S_0

Nous avons ici un simple changement de repère, du repère cartésien R_0 au repère sphérique S_0. Nous cherchons donc à calculer les coordonnées $(r_{A_0}, \theta_{A_0}, \phi_{A_0})$ du point A dans le repère S_0.

$$\begin{cases} r_{A_0} = \sqrt{x_{A_0}^2 + y_{A_0}^2 + z_{A_0}^2} \\ \theta_{A_0} = \arccos\left(\frac{y_{A_0}}{r_{A_0}}\right) \\ \phi_{A_0} = \arcsin\left(\frac{x_{A_0}}{\sqrt{x_{A_0}^2 + z_{A_0}^2}}\right) \end{cases}$$

3.2.3 Calcul de A dans le repère sphérique S_1

Le repère S_1 nous permet de mettre en évidence la rotation du point autour du centre de la scène. Il s'agit ici d'exprimer le point A de coordonnées $(r_{A_0}, \theta_{A_0}, \phi_{A_0})$ dans le repère S_0 dans le repère S_1. Le point A prendra donc comme nouvelle coordonnées : $(r_{A_1}, \theta_{A_1}, \phi_{A_1})$. Pour calculer les nouvelles coordonnées de A dans le repère S_1, nous allons utiliser le point V afin qu'il nous donne les angles θ et ϕ, moteurs de la rotation.

$$\begin{cases} r_{A_1} = r_{A_0} \\ \theta_{A_1} = \theta_{A_0} - \theta_{V_0} \\ \phi_{A_1} = \phi_{A_0} - \phi_{V_0} \end{cases}$$

3.2.4 Calcul de A dans le repère R_3

Nous allons donc ici calculer les coordonées du point A dans le repère R_3 qui possède même origine que le repère de la scène R_0 mais dont les axes sont orientés différement. En effet, l'axe (O_{z_3}) correspond au vecteur $\overrightarrow{O_0V}$, le point V étant l'origine de la vue de coordonnées $(R_0, x_{v_0}, y_{v_0}, z_{v_0})$. Le repère R_3 étant la transformation cartésienne du repère sphérique S_1.

$$\begin{cases} x_{A_3} = r_{A_1} \sin(\theta_{A_1}) \sin(\phi_{A_1}) \\ y_{A_3} = r_{A_1} \cos(\theta_{A_1}) \\ z_{A_3} = r_{A_1} \sin(\theta_{A_1}) \cos(\phi_{A_1}) \end{cases}$$

Détail du calcul de l'abscisse de A dans R_3

Nous avons :

$$\begin{aligned} x_{A_3} &= r_{A_1} \sin(\theta_{A_1}) \sin(\phi_{A_1}) \\ &= r_{A_0} \sin(\theta_{A_0} - \theta_{V_0}) \sin(\phi_{A_0} - \phi_{V_0}) \end{aligned}$$

Sachant que,

$$\begin{aligned} \sin(\theta_{A_0} - \theta_{V_0}) &= \sin(\theta_{A_0}) \cos(\theta_{V_0}) - \cos(\theta_{A_0}) \sin(\theta_{V_0}) \\ &= \frac{\sqrt{r_{A_0}^2 - y_{A_0}^2}}{r_{A_0}} \cos(\theta_{V_0}) - \frac{y_{A_0}}{r_{A_0}} \sin(\theta_{V_0}) \\ &= \frac{\sqrt{x_{A_0}^2 + z_{A_0}^2} \cos(\theta_{V_0}) - y_{A_0} \sin(\theta_{V_0})}{r_{A_0}} \end{aligned}$$

De même que,

$$\begin{aligned}\sin(\phi_{A_0}-\phi_{V_0}) &= \sin(\phi_{A_0})\cos(\phi_{V_0})-\cos(\phi_{A_0})\sin(\phi_{V_0})\\ &= \frac{x_{A_0}}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\cos(\phi_{V_0})-\frac{z_{A_0}}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\sin(\phi_{V_0})\\ &= \frac{x_{A_0}\cos(\phi_{V_0})-z_{A_0}\sin(\phi_{V_0})}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\end{aligned}$$

Nous avons donc :

$$\begin{aligned}x_{A_3} &= r_{A_0}\sin(\theta_{A_0}-\theta_{V_0})\sin(\phi_{A_0}-\phi_{V_0})\\ &= \frac{\left(\sqrt{x_{A_0}^2+z_{A_0}^2}\cos(\theta_{V_0})-y_{A_0}\sin(\theta_{V_0})\right)\left(x_{A_0}\cos(\phi_{V_0})-z_{A_0}\sin(\phi_{V_0})\right)}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\\ &= \left(\cos(\theta_{V_0})-\frac{y_{A_0}}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\sin(\theta_{V_0})\right)\left(x_{A_0}\cos(\phi_{V_0})-z_{A_0}\sin(\phi_{V_0})\right)\end{aligned}$$

Détail du calcul de la profondeur de A dans R_3

Nous avons :

$$\begin{aligned}z_{A_3} &= r_{A_1}\sin(\theta_{A_1})\cos(\phi_{A_1})\\ &= r_{A_0}\sin(\theta_{A_0}-\theta_{V_0})\cos(\phi_{A_0}-\phi_{V_0})\end{aligned}$$

Sachant que :

$$\sin(\theta_{A_0}-\theta_{V_0}) = \frac{\sqrt{x_{A_0}^2+z_{A_0}^2}\cos(\theta_{V_0})-y_{A_0}\sin(\theta_{V_0})}{r_{A_0}}$$

De même que :

$$\begin{aligned}\cos(\phi_{A_0}-\phi_{V_0}) &= \cos(\phi_{A_0})\cos(\phi_{V_0})+\sin(\phi_{A_0})\sin(\phi_{V_0})\\ &= \frac{z_{A_0}}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\cos(\phi_{V_0})+\frac{x_{A_0}}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\sin(\phi_{V_0})\\ &= \frac{x_{A_0}\sin(\phi_{V_0})+z_{A_0}\cos(\phi_{V_0})}{\sqrt{x_{A_0}^2+z_{A_0}^2}}\end{aligned}$$

Nous avons donc :

$$z_{A_3} = r_{A_0}\sin(\theta_{A_0}-\theta_{V_0})\cos(\phi_{A_0}-\phi_{V_0})$$

$$
\begin{aligned}
&= \frac{\left(\sqrt{x_{A_0}^2 + z_{A_0}^2}\cos(\theta_{V_0}) - y_{A_0}\sin(\theta_{V_0})\right)(x_{A_0}\sin(\phi_{V_0}) + z_{A_0}\cos(\phi_{V_0}))}{\sqrt{x_{A_0}^2 + z_{A_0}^2}} \\
&= \left(\cos(\theta_{V_0}) - \frac{y_{A_0}}{\sqrt{x_{A_0}^2 + z_{A_0}^2}}\sin(\theta_{V_0})\right)(x_{A_0}\sin(\phi_{V_0}) + z_{A_0}\cos(\phi_{V_0}))
\end{aligned}
$$

Détail du calcul de l'ordonnée de A dans R_3

Nous avons :

$$
\begin{aligned}
y_{A_3} &= r_{A_1}\cos(\theta_{A_1}) \\
&= r_{A_0}\cos(\theta_{A_0} - \theta_{V_0}) \\
&= r_{A_0}\left(\cos(\theta_{A_0})\cos(\theta_{V_0}) + \sin(\theta_{A_0})\sin(\theta_{V_0})\right) \\
&= r_{A_0}\left(\frac{y_{A_0}}{r_{A_0}}\cos(\theta_{V_0}) + \frac{\sqrt{x_{A_0}^2 + z_{A_0}^2}}{r_{A_0}}\right) \\
&= y_{A_0}\cos(\theta_{V_0}) + \sqrt{x_{A_0}^2 + z_{A_0}^2}
\end{aligned}
$$

3.2.5 Calcul de A dans R_4

Le repère R_4 n'est autre que la projection du repère R_3 sur l'écran. Le point étant considéré comme le centre de la zone d'affichage.

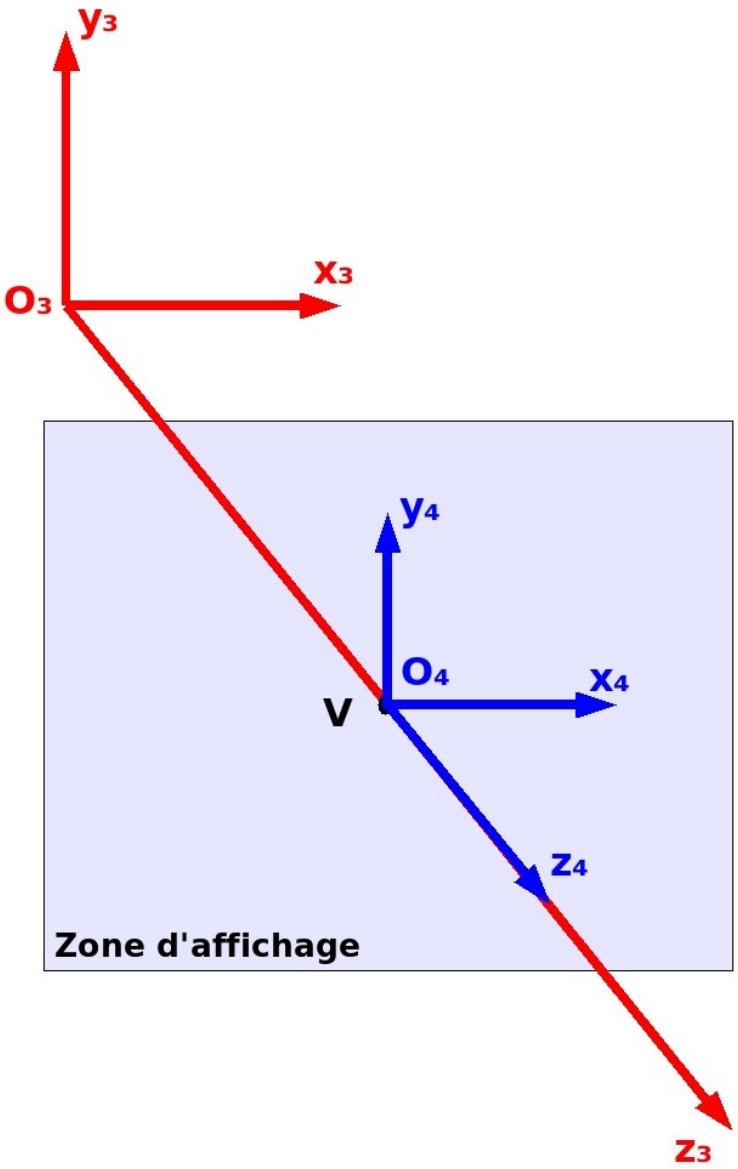

FIG. 3.2 – Repère R_3 et R_4

Nous faisons apparaître, grâce à ce nouveau repère, la notion de distance du point. Nous cherchons donc ici l'image du point A sur l'écran. La profondeur A_z n'intervenant dans l'affichage, nous nous concentrons ici uniquement sur les coordonnées A_x et A_y.

D'après la relation de thales, nous avons :

$$\frac{r_{V_0}}{A_{z_3}} = \frac{A_{x_4}}{A_{x_3}} = \frac{A_{y_4}}{A_{y_3}}$$

Puisque $\|O_3V\| = r_{V_0}$.

Les nouvelles coordonnées du point A projeté sur l'écran (dans le repère R_4)

sont :

$$\begin{cases} x_{A_4} = x_{A_3} \frac{r_{V_0}}{z_{A_3}} \\ y_{A_4} = y_{A_3} \frac{r_{V_0}}{z_{A_3}} \end{cases}$$

3.2.6 Calcul de A dans R_5

Les coordonnées A_{x_5} et A_{y_5} du point A dans le repère R_5 nous permet d'obtenir les coordonnées du point A prêtes à être afficher par le module QT. Nous tenons compte ici de la hauteur H et de la largeur L de la zone d'affichage. Nous savons donc que le point V à pour coordonnées $(\frac{L}{2}, \frac{H}{2})$ dans le repère R_5 étant donné qu'il se situe au centre de l'écran.

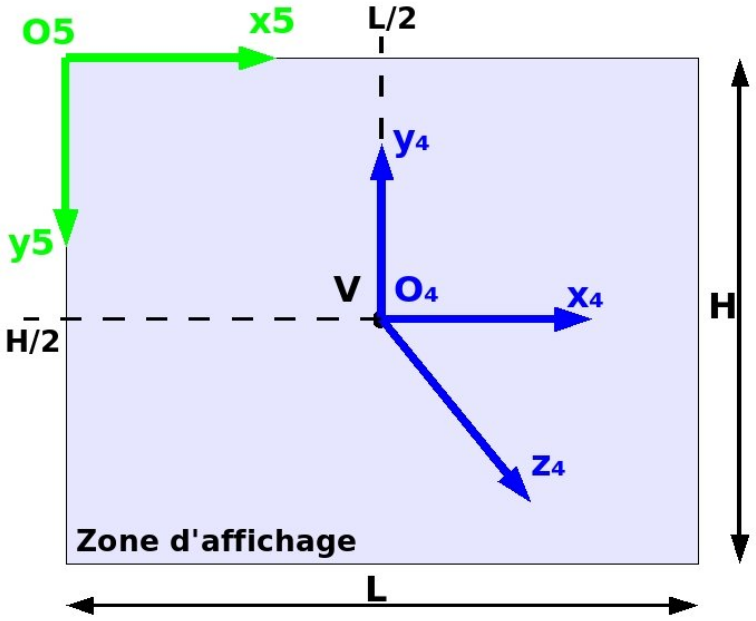

FIG. 3.3 – Repère R_4 et R_5

Les coordonnées $(A_{x_5}, A_{y_5}, A_{z_5})$ du point A dans le repère R_5 sont donc :

$$\begin{cases} x_{A_5} = x_{A_4} - \frac{L}{2} \\ y_{A_5} = -y_{A_4} + \frac{H}{2} \end{cases}$$

3.2.7 Résumé : Projection sur l'écran

Nous partons avec les informations suivantes :

A : $(R_0, x_{A_0}, y_{A_0}, z_{A_0})$

V : $(S_0, r_{V_0}, \theta_{V_0}, \phi_{V_0})$

L : Largeur de la zone d'affichage

H : Hauteur de la zone d'affichage

Abscisse de A dans R_3

On a :

$$x_{A_3} = \left(\cos(\theta_{V_0}) - \frac{y_{A_0}}{p(A)}\sin(\theta_{V_0})\right)(x_{A_0}\cos(\phi_{V_0}) - z_{A_0}\sin(\phi_{V_0}))$$

Avec,

$$\sqrt{x_{A_0}^2 + z_{A_0}^2} = p(A)$$

Abscisse de A dans R_5

On a :

$$\begin{cases} x_{A_5} = x_{A_4} - \frac{L}{2} \\ x_{A_4} = x_{A_3}\frac{r_{V_0}}{z_{A_3}} \end{cases}$$

Ce qui nous donne :

$$\begin{aligned} x_{A_5} &= x_{A_4} - \frac{L}{2} \\ &= x_{A_3}\frac{r_{V_0}}{z_{A_3}} - \frac{L}{2} \end{aligned}$$

Ordonnée de A dans R_3

On a :

$$y_{A_3} = y_{A_0}\cos(\theta_{V_0}) + p(A)$$

Avec,

$$\sqrt{x_{A_0}^2 + z_{A_0}^2} = p(A)$$

Ordonnée de A dans R_5

On a :

$$\begin{cases} y_{A_5} = -y_{A_4} + \frac{H}{2} \\ y_{A_4} = y_{A_3}\frac{r_{V_0}}{z_{A_3}} \end{cases}$$

Ce qui nous donne :

$$\begin{aligned} y_{A_5} &= -y_{A_4} + \frac{H}{2} \\ &= -y_{A_3}\frac{r_{V_0}}{z_{A_3}} + \frac{H}{2} \end{aligned}$$

Profondeur de A

Nous ne gardons que les coordonnées de la profondeur de A dans le repère R_3 car elles nous servent dans l'algorithme du peintre.

On a :

$$\begin{aligned} z_{A_3} &= \left(\cos(\theta_{V_0}) - \frac{y_{A_0}}{\sqrt{x_{A_0}^2 + z_{A_0}^2}}\sin(\theta_{V_0})\right)(x_{A_0}\sin(\phi_{V_0}) + z_{A_0}\cos(\phi_{V_0})) \\ &= \left(\cos(\theta_{V_0}) - \frac{y_{A_0}}{p(A)}\sin(\theta_{V_0})\right)(x_{A_0}\sin(\phi_{V_0}) + z_{A_0}\cos(\phi_{V_0})) \end{aligned}$$

Avec,

$$p(A) = \sqrt{x_{A_0}^2 + z_{A_0}^2}$$

3.3 Origine de la vue

3.3.1 Le point V

Le point V désigne le point d'origine de la vue. Le vecteur $\overrightarrow{O_0V}$ nous donne donc deux informations sur le point de vue :

- information sur l'angle de vue donné par le vecteur $\overrightarrow{O_0V}$ de coordonnées $(R_0, x_{v_0}, y_{v_0}, z_{v_0})$;
- information sur la distance de l'objet (zoom) donné par le module du vecteur $\overrightarrow{O_0V}$: $\sqrt{x_{v_0}^2 + y_{v_0}^2 + z_{v_0}^2}$

Nous représenterons donc le vecteur $\overrightarrow{O_0V}$ par ses trois valeurs afin de simplifier les calculs et l'interprétation :

- θ
- ϕ
- d

3.3.2 Distance de la vue

Dans un soucis de visibilité, nous imposons un minimum et un maximum pour la distance entre les points O_0 et V. Soit d la distance entre les deux points les plus éloignés de l'objet.

Nous avons donc :

$$2 * d \leq \|O_0V\| \leq 5 * d$$

3.3.3 Caméra

La caméra se situe très exactement à l'origine de la vue. Cette origine correspond au point V et la direction de la vue est définie par la droite (O_0V). Au niveau représentation, l'orgine de la vue correspond non pas à l'oeil de l'utilisateur mais au centre de la zone d'affichage de la scène 3d à l'écran.

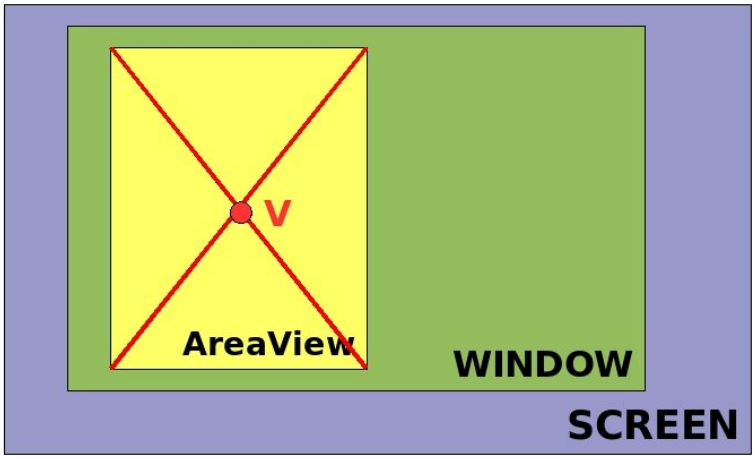

FIG. 3.4 – Représentation de l'origine de la vue

Chapitre 4

Techniques et principes

4.1 Algorithme du peintre

L'algorithme du peintre est un algorithme qui nous permet de définir dans quel ordre les polygones seront affichés. Ce qui permet d'écrire par dessus les surfaces et donne ainsi un semblant de cohérence et de perspective. Ainsi on affiche tout d'abord les surfaces dont les barycentres sont les plus éloignés puis de plus en plus proche.

4.2 Rotation

La rotation de la caméra s'effectue toujours autour du centre de la scène 3d O_0. Cela consiste donc à modifier l'angle de vue en faisant tourner le point V par rapport au centre de la scène R_0. Notre objectif est donc de modifier les angles θ et ϕ puis de recalculer l'abscisse et l'ordonnée du point A dans le repère R_5 ainsi que sa profondeur dans le repère R_3 grâce à ces nouveaus angles.

4.3 Zoom

Un zoom ou un dézoom consiste à rapprocher où à éloigner la caméra de l'objet 3d. En clair, il suffit juste de jouer sur le module du vecteur $\overrightarrow{O_2V}$, c'est à dire avec le coefficient r_{V_0} du point V.

Deuxième partie

Conception logicielle

Chapitre 5

Interface

5.1 Fenêtre principal

5.1.1 Interface

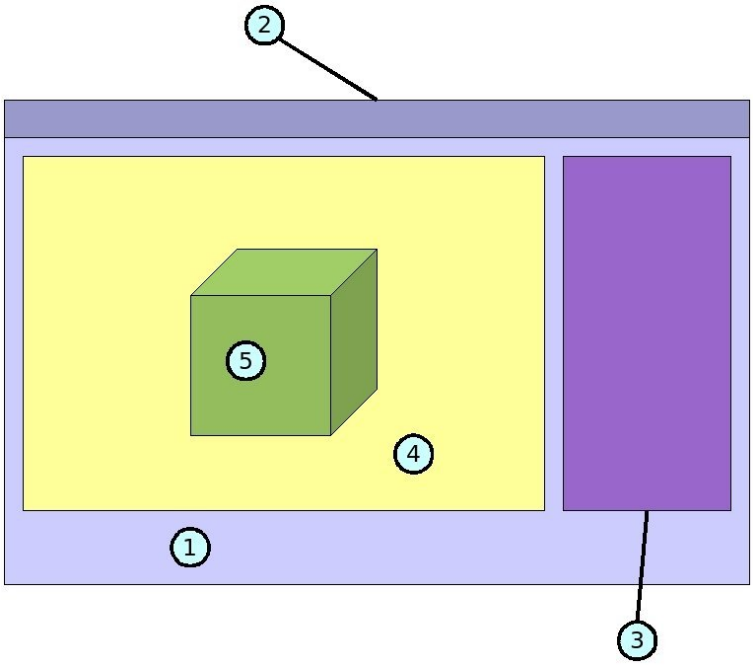

FIG. 5.1 – Fenêtre principal

La fenêtre principale du programme se compose ainsi :

1 : Fenêtre QT

2 : Barre de titre contenant le titre de l'application

3 : Panneau latéral de commande

4 : Zone d'affichage de la scène 3d

5 : Objets 3d à afficher

5.1.2 Squelette

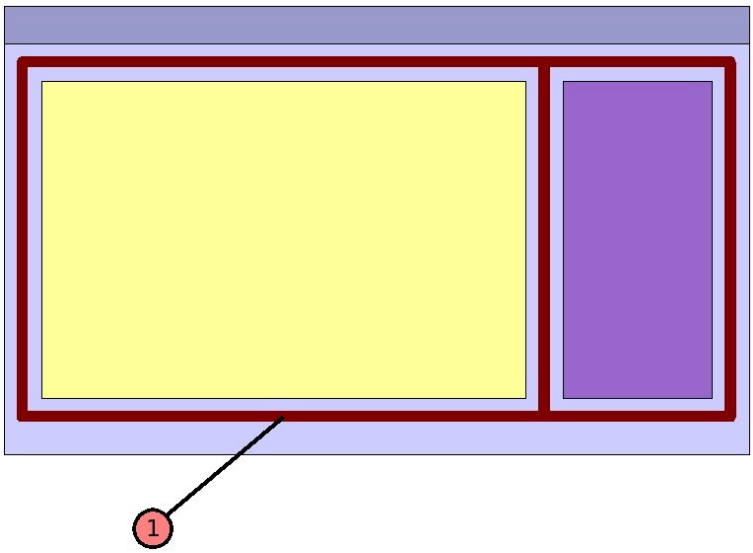

FIG. 5.2 – Squelette de la fenêtre principal

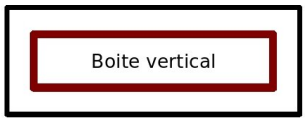

FIG. 5.3 – Légende

5.2 Panneau latéral

5.2.1 Interface

Le panneau latéral permet de faire l'interface entre l'homme et la machine. Il permet donc à l'utilisateur de :

- charger un fichier OFF ou SFF ;
- changer les paramêtre du point de vue V (les angles de vue θ et ϕ et le module r) ;
- de réinitialiser ces précédentes valeurs ;

Le panneau latéral se compose donc de :

1 : Panneau latéral

2 : Bouton de chargement de fichier

3 : Label contenant les informations que l'utilisateur peut modifier

4 : Champs de texte où l'utilisateur peut rentrer les nouvelles valeurs

5 : Bouton validant les valeurs rentrés par l'utilisateur

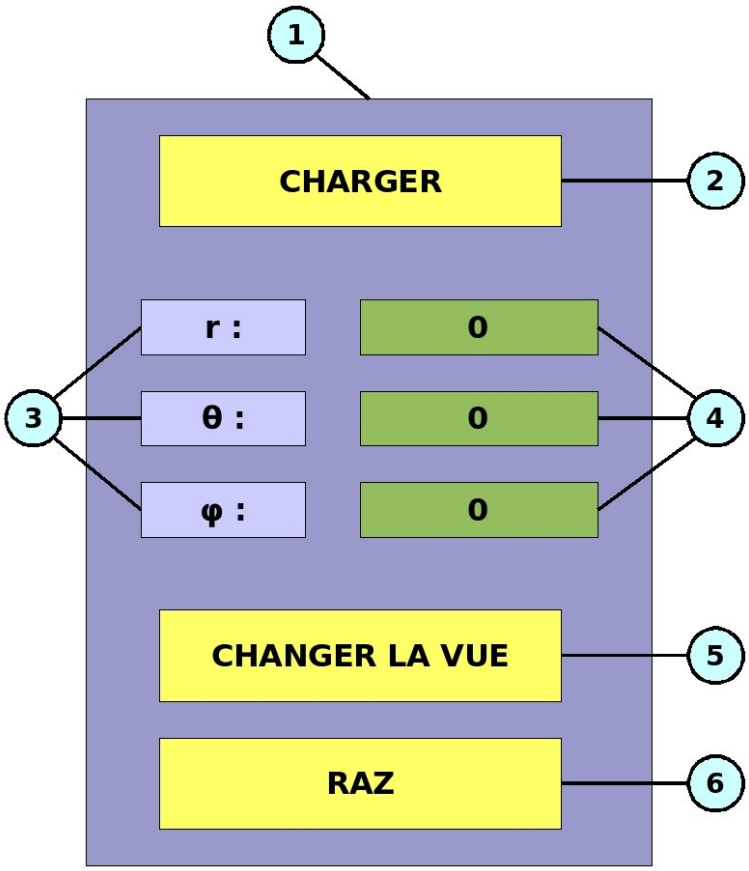

FIG. 5.4 – Panneau latéral

6 : Bouton permettant de réinitialiser ces valeurs

5.2.2 Squelette

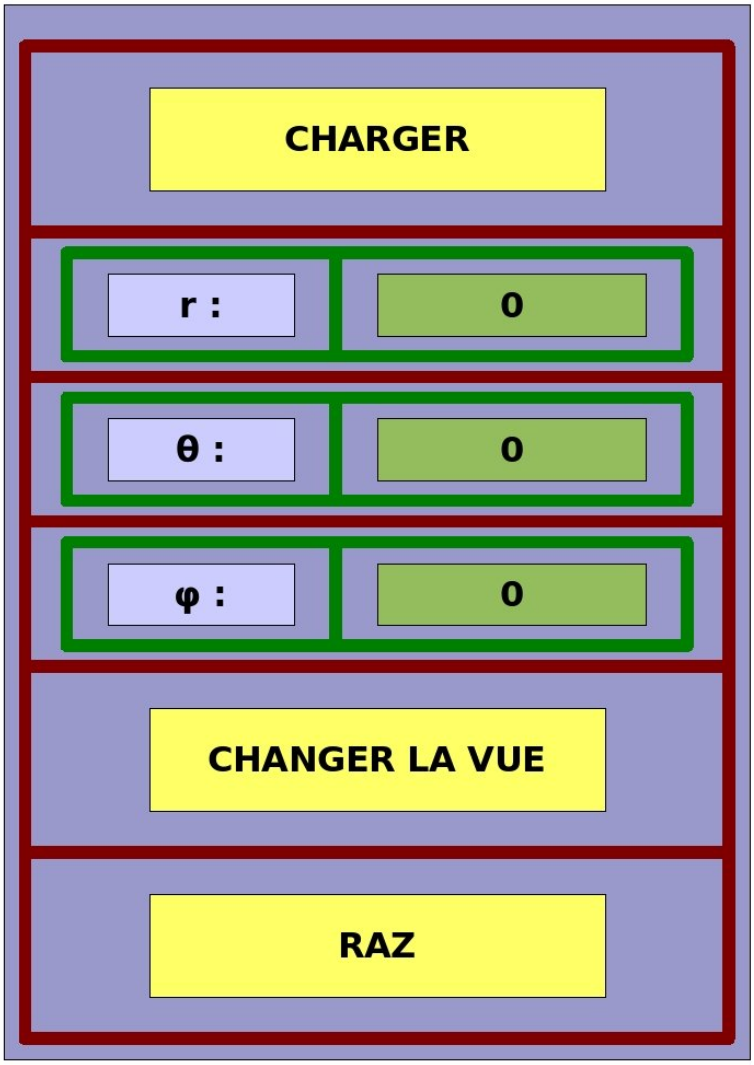

FIG. 5.5 – Squelette du panneau latéral

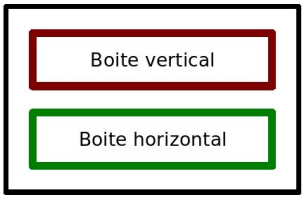

FIG. 5.6 – Légende

Chapitre 6

Fonctionnement

6.1 Initialisation

6.1.1 Dialogue

Dans un premier temps, le programme entreprend un dialogue avec l'utilisateur afin de pouvoir initialiser correctement le programme. Il s'agit d'un simple menu composé de 2 éléments :

- charger un fichier OFF ou SFF ;
- quitter le programme ;

Dans la cas où l'utilisateur souhaite charger un objet 3d à partir d'un fichier, il lui sera demandé de rentrer le chemin vers le fichier.

6.1.2 Chargement du fichier

Nous distinguons ici deux cas distinct :

- l'utilisateur souhaite charger un fichier OFF ;
- l'utilisateur souhaite charger un fichier SFF ;

Fichier OFF

Dans le cas d'un fichier ne contenant qu'un seul objet 3d, nous créons une instance de la classe *Object_ 3D* avec en paramètre le chemin vers le fichier à exploiter.

Fichier SFF

Dans le cas d'un fichier contenant une scène objets 3d, nous testons l'existence et la validité dudit fichier. Nous créons ensuite autant d'instance de la classe *Object_ 3D* que d'objets 3d composant la scène avec en paramètre de chacun, le chemin vers le fichier à exploiter ainsi que les coordonnées du centre dudit objet dans le repère de la scène R_0.

6.1.3 Création des objets 3d

Lors de la création d'une instance de classe *Object_ 3D*, nous devons tout d'abord exploiter le fichier OFF contenant l'objet 3d afin de créer les polygones qui le compose. Ensuite le calculs du barycentre de l'objet 3d nous permet de modifier les coordonnées 3d de chacun des polygones.

Vérification du fichier

Avant même l'exploitation du fichier, nous vérifions :
- son existence et son accès en lecture ;
- son extension en .off ;
- son entête (la première ligne doit contenir le label OFF) ;

Premier lecture du fichier OFF

Nous récupérons ici les nombres *int_ nb_ polygon* et *int_ nb_ point* de polygones et de points 3d.

Définition des vertices

Ici nous créons le tableau des *int_ nb_ point* point 3d. Pour cela, nous lisons chacune des lignes du fichier nous renseignant sur les vertices à récupérer. Zn parallèle, nous créons autant d'instances de la classe *Point_ 3D* avec en paramètre les coordonnées du point dans l'espace.

Définition des polygones

Ici nous créons la liste des *int_ nb_ polygon* polygones qui composent notre objet 3d. Pour cela, nous lisons chacune des lignes du fichier nous renseignant sur les polygones à récupérer. Nous stockons ensuite la liste des vertices utilisés du tableau *tab_ point3d* dans un tableau temporaire. Nous créons en parallèle autant d'instances de la classe *Polygon* avec en paramètre :
- la tableau temporaire contenant les instances de la classe *Point_ 3D* dont se compose le polygone.
- L'objet de la classe QColor contenant les informations sur la couleur du polygone.

Centre de l'objet

Trouver le centre de l'objet 3d revient à calculer le barycentre de chacun des barycentres des polygones qui composent l'objet 3d. Une fois que nous avons trouver ledit centre, nous modifions les coordonnées de chacun des instances *Polygon* donc se compose l'objet en fonction :
- du barycentre de l'objet 3d ;
- des coordonnées du centre de l'objet dans le repère R_0 ;

6.1.4 Création des points 3d

A sa création, une instance de la classe *Point_ 3D* se contente de stocker ses informations sur sa position dans l'espace.

6.1.5 Création des polygones

Lors de sa création, une instance de la classe *Polygon* stocke, s'il elles sont présentes, les informations sur la couleur du polygones ainsi que le tableau de coordonnées des points qui composent le polygone.

6.1.6 Initialisation de la fenêtre

Une fois toutes les étapes précédentes effectués avec succès, nous allons initilisé et crée la fenêtre de rendu QT. La fenêtre de rendu étant représenté par une instance de la classe *Window* prenant en paramètre la liste et le nombre d'argument donné lors de l'éxécution du programme. Il s'agit simplement ici de :

- initialiser Qt ;
- créer la fenêtre d'affichage ;
- créer une fone d'affichage (instance de la classe *AreaView* dans laquelle s'afficheront les objets 3d ;
- créer le panneau latéral contenant les champs de texte et les boutons :

6.2 Affichage

La mise à jour de l'affichage se fait par l'intermédiaire des fonctions *rotation* et *zoom* de la classe *Scene_ 3D*. Ces fonctions prennent en paramêtres les angles θ, ϕ ainsi que le coefficient de zoom r. Il s'agit ici de stocker ces coordonnées du point de vue V dans une structure *struct_ point_ sphere*.

6.2.1 Projection

Projection de la scène 3d

Pour chacun des objets 3d, nous appelons la fonction *Objet_ 3D : :project* qui prend en paramêtre cette structure afin de mettre à jour chacun des objets 3D. L'ordre d'affichage des objets 3D est défini par l'algorithme du peintre et nous est donné grâce à la fonction de tri de la stl.

Projection d'un objet 3d

Avec l'algorithme du peintre, nous définissons la priorité d'affichage des polygones. Puis nous appelons, pour chacun des polygones composant l'objet en question, la fonction de projection *Polygon : :project* de la classe *Polygon*. Nous ne faisons donc que transmettre la structure du point de vue V en coordonnées sphérique.

Projection d'un polygone

Grâce aux coordonnées sphériques du point V, nous projectons chacun des points composant le polygone. Pour ce faire, nous appelons la fonction *Point_ 3D : :project* de chacun des points 3d.

Projection d'un point 3d

Grâce aux coordonnées sphériques du point 3D, nous calculons les coordonnées du point dans le repère R_3 puis nous remplaçons les nouvelles coordonnées par les anciennes.

6.2.2 Affichage

Dans un premier temps, nous effaçons la zone d'affichage. Nous appelons ensuite, les fonctions :

- *Scene_ 3D : :show*
- *Object_ 3D : :show*
- *Polygon_ 3D : :show*

Troisième partie

Architecture logicielle

Chapitre 7

Représentation du système

Nous disposons de cinq classes :

Scene_3D : Classe de la scène 3d.

Object_3D : Classe des objets 3d.

Point_3D : Classe des points avec des coordonnées 3d.

Polygon : Classe des polygones avec des coordonnées dans l'espace 3d.

Window : Classe de la fenetre dans laquelle s'afficheront les objets 3d.

AreaView : Classe de la zone d'affichage.

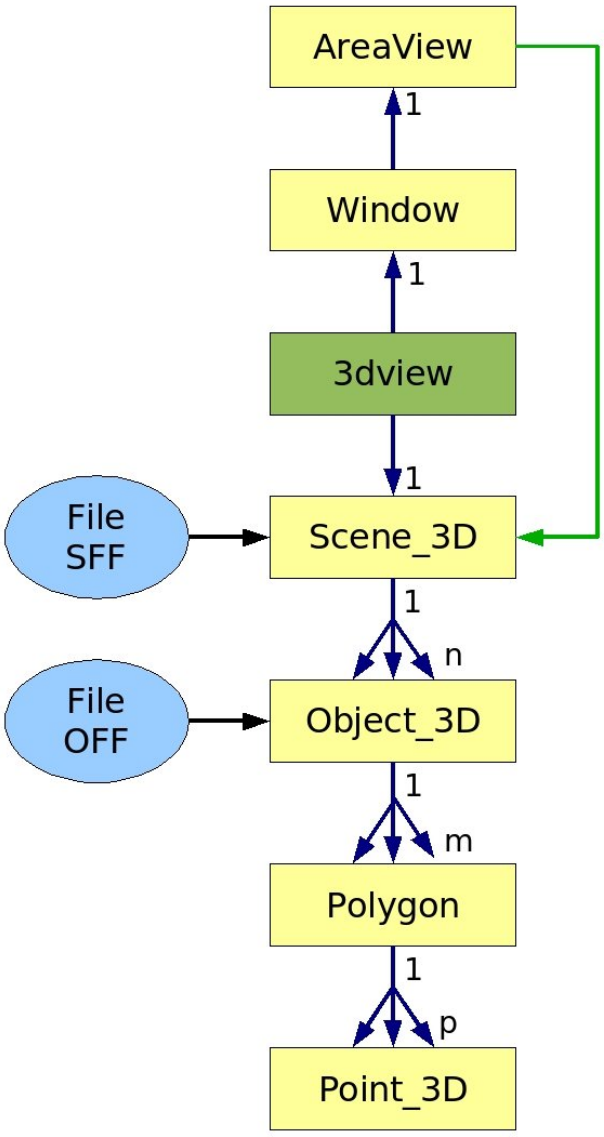

FIG. 7.1 – Architecture logicielle

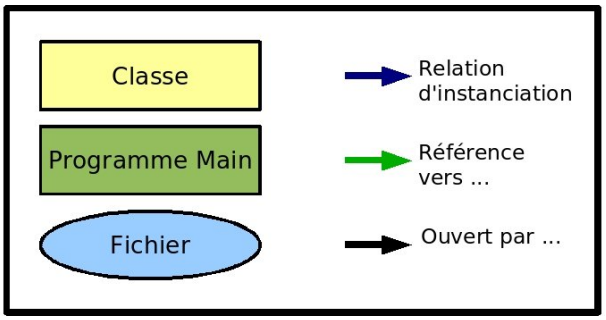

FIG. 7.2 – Légende architecture logicielle

Le programme principale contient une instance de la classe *Scene_ 3D* ainsi qu'une instance de la classe *Window*. A sa création, l'objet de la classe *Scene_ 3D* ouvre le fichier SFF contenant les n objets 3d et créé donc n instances de la classe *Object_ 3D*. L'objet *Scene_ 3D* est crée même si la scène ne contient qu'un seul objet 3d. Dans ce cas, il ne créera qu'une seule instance de la classe *Object_ 3D*. Chaque objet 3d est associé à un fichier OFF contenant les m polygones composant cet objet. Une instance de la classe *Object_ 3D* crée donc m instances de la classe *Polygon*. Chaque polygone étant représenté par p points disposant de coordonnées dans l'espace, chaque objet de la classe *Polygone* utilise donc p instances de la classe *Point_ 3D*. La classe *Window* gère la création de la fenêtre QT ainsi que d'une instance de la classe *AreaView*. Une instance de la classe *AreaView* correspondant à la zone d'affichage dans laquelle les objets 3d s'afficheront et contient une référence vers l'objet *Scene_ 3D*.

Chapitre 8

Programme principal

Le programme principal se compose des deux fichiers sources 3dview.cpp et 3dview.hpp. Il contient essentiellement les fonctions nécéssaires au démarrage du programme. Cela va de la création des objets 3d et de la fenêtre à la gestion des menus d'interface avec l'utilisateur.

Chapitre 9

Structure struct_point_sphere

La structure *struct_point_sphere* nous sert à décrire des points en coordonnées sphériques. Le point M a par exemple les coordonnées (r_M, θ_M, ϕ_M) dans le repère sphérique.

Elle contient essentiellement ces informations :

double dbl_r : Distance entre le point M et l'origine O du repère.

double dbl_theta : Angle formé entre l'axe Oy et la droite OM.

double dbl_phi : Angle formé entre la l'axe Oz et la droite OM projeté sur le plan zOx.

Chapitre 10

Classe Scene_3D

Une instance de la classe *Scene_ 3D* représente une scène de plusieurs objets 3d contenu dans un fichier SFF ou d'un seul objet 3d contenu dans un fichier OFF.

10.1 Variables de classe

Les variables de la classe *Scene_ 3D* sont :

struct_point_sphere origin_view : Coordonnées sphérique du point de vue V.

vector <Objet_3D*> vect_objet3d : Tableau de pointeurs vers des objets de la classe *Object_ 3D*.

10.2 Fonctions membres

Les fonctions membres sont regroupés en 3 catégories :

10.2.1 Fonctions de classe

Constructeur par paramètres

```
Scene_3D : :Scene_3D(string file) ;
```

Le constructeur par paramètres permet de construire la scène 3d à partir du fichier OFF ou SFF *file*.

Constructeur par copie

```
Scene_3D : :Scene_3D(const Scene_3D & ) ;
```

Le constructeur par copie permet de construire une scène 3d à partir d'une autre

Destructeur

```
Scene_3D : :~Scene_3D(void) ;
```

10.2.2 Accesseurs

Paramêtre r

```
double Scene_3D : :get_pov_r(void) ;
void Scene_3D : :set_pov_r(double r) ;
```

Angle θ

```
double Scene_3D : :get_pov_theta(void) ;
void Scene_3D : :set_pov_theta(double theta) ;
```

Angle ϕ

```
double Scene_3D : :get_pov_phi(void) ;
void Scene_3D : :set_pov_phi(double phi) ;
```

10.2.3 Fonctions d'interaction

Lecture d'un fichier

```
bool Scene_3D : :load_from_file(string file) ;
```

La fonction de chargement permet de charge une scène 3d à partir d'un fichier OFF ou SFF.

Test d'un fichier

```
bool Scene_3D : :test_file(string file) ;
```

Cette fonction de test de fichier permet de tester l'existence d'un fichier. Elle permet également de valider son format SFF.

Changement de vue

```
bool Scene_3D : :rotate_view(double dbl_theta, double dbl_phi) ;
```

La fonction de rotation permet de modifier l'origne de la vue V selon les angles *theta*, *phi*.

```
bool Scene_3D : :zoom(double dbl_coeff) ;
```

La fonction de zoom permet de modifier la distance de l'origine de la vue V au point d'origine O_0 du repère R_0.

Affichage

```
bool Scene_3D : :show(AreaView* area) const ;
```

La fonction d'affichage se contente d'afficher à son tour chacun des objets 3d contenus dans la scène 3d.

Chapitre 11

Classe Object_3D

Une instance de la classe *Object_ 3D* représente un objet 3d contenu dans un fichier OFF.

11.1 Variables de classe

Les variables de la classe *Object_ 3D* sont :

vector <Polygon*> vect_polygon : Tableau de pointeurs vers des objets de la classe *Polygon*.

vector <Point_3D*> vect_point3d : Tableau à 1 dimension de pointeurs vers des objets de la classe *Point_ 3D*.

Point_3D* pt_barycentre : Barycentre de l'objet 3D.

11.2 Fonctions membres

Les fonctions membres sont regroupés en 3 catégories :

11.2.1 Fonctions de classe

Constructeur par paramètres

```
Object_3D : :Object_3D(string file_off) ;
Object_3D : :Object_3D(string file_off, Point_3D origin) ;
```

Le constructeur par paramètres permet de construire un objet 3d en fonction du fichier OFF *file_ off* et, s'il existe, de l'origine O_2 du repère R_2 dans les coordonnées du repère R_0.

Constructeur par copie

```
Object_3D : :Object_3D(const Object_3D & ) ;
```

La constructeur par copie permet de construire un objet 3d à partir d'un autre objet 3d.

Destructeur

```
Object_3D : :~Object_3D(void) ;
```

11.2.2 Opérateurs

Opérateur de comparaison

```
bool Object_3D : :operator<(const Object_3D & other_object) const ;
```

La redéfinition de l'opérateur de comparaison inférieur ($<$) sert dans l'algorithme du peintre afin de trier et de définir l'ordre d'affichage des objets 3d.

11.2.3 Accesseurs

Accesseurs tableau polygone

```
Polygon* Object_3D : :get_polygon(int int_n) ;
```

Accesseurs nombre de polygone

```
int Object_3D : :get_nb_polygon(void) ;
```

11.2.4 Fonctions d'interaction

Lecture fichiers

```
bool Object_3D : :load_from_file(string file) ;
```

La fonction de chargement permet de charger un objet 3d à partir d'un fichier OFF.

Test fichiers

```
bool Object_3D : :test_file(string file) ;
```

Cette fonction permet de tester si le fichier existe bien et s'il s'agit bien d'un fichier OFF. Notamment, grâce à l'extension.

Affichage

```
bool Object_3D : :show(AreaView* area) ;
```

La fonction d'affichage du polygone se contente d'afficher chacun des polygones le composant. L'algorithme de sélection des surfaces (algorithme du peintre) doit être appliqué ici.

Projection à l'écran

```
bool Object_3D : :project(struct_point_sphere* origin_view) ;
```

Cette fonction permet de projeter l'objet 3d à l'écran en fonction de l'origine de la vue *origin_ view*.

Calcul du barycentre

```
Point_3D* Object_3D : :barycentre(void);
```

La fonction de calcul du barycentre se contente de renvoyer le barycentre de l'objet 3d.

Chapitre 12

Classe Point_3D

Une instance de la classe *Point_ 3D* représente un point dans l'espace.

12.1 Variables de classe

Les variables de la classe *Point_ 3D* sont :

double dbl_x : Abcisse du point.

double dbl_y : Ordonnée du point.

double dbl_z : Profondeur du point

12.2 Fonctions membres

Les fonctions membres sont regroupés en 4 catégories :

12.2.1 Fonctions de classe

Constructeur par défaut

```
Point_3D : :Point_3D(void) ;
```

Construit le point avec les coordonnées $(0, 0, 0)$ par défaut.

Constructeur par paramètres

```
Point_3D : :Point_3D(double x, double y, double z) ;
```

Construit le point ayant pour coordonnées (x, y, z).

Constructeur par copie

```
Point_3D : :Point_3D(const Point_3D & other_point) ;
```

Construit un point 3d à partir d'un autre point.

Destructeur

```
Point_3D : :~Point_3D(void) ;
```

12.2.2 Accesseurs

Abcisse

```
double Point_3D : :get_x(void) ;
void Point_3D : :set_x(double x) ;
```

Les accesseurs à l'abcisse permette respectivement de récupérer et de modifier l'abcisse du point.

Ordonnée

```
double Point_3D : :get_y(void) ;
void Point_3D : :set_y(double y) ;
```

Les accesseurs à l'ordonnée permette respectivement de récupérer et de modifier l'ordonnée du point.

Profondeur

```
double Point_3D : :get_z(void) ;
void Point_3D : :set_z(double z) ;
```

Les accesseurs à la profondeur permette respectivement de récupérer et de modifier la profondeur du point.

12.2.3 Fonctions d'intéraction

Projection à l'écran

```
bool Point_3D : :project(struct_point_sphere* origine_view) const ;
```

Cette fonction permet de projeter un point 3d dans la zone d'affichage *area* en fonction de l'origine de la vue *origin_ view*. Elle permet notamment de calculer les nouvelles coordonnées du point dans le repère R_3.

Affichage

```
bool Point_3D : :show(AreaView* area, struct_point_sphere* origine_view) const ;
```

Calcul d'une distance

```
double Point_3D : :distance(const Point_3D & point_compare) const ;
```

Cette fonction permet de calculer la distance entre deux points.

Chapitre 13

Classe Polygon

Une instance de la classe *Polygon* représente un polygone situé dans l'espace. C'est à dire un polygone avec des coordonnées 3 dimensions.

13.1 Variables de classe

Les variables de la classe *Polygon* sont :

QColor polygon_color : Couleur du polygon.

list <Point_3D*> list_vertice : Liste des vertices (point 3d) composant le polygone.

Point_3D* pt_barycentre : Point 3d représentant le barycentre du polygone.

13.2 Fonctions membres

Les fonctions membres sont regroupés en 3 catégories :

13.2.1 Fonctions de classe

Constructeur par défaut

```
Polygon : :Polygon(void) ;
```

Le constructeur par défaut permet de construire un polygone de base. Le polygone par défaut est un triangle.

Constructeur par paramètres

```
Polygon : :Polygon(list <Point_3D*> list_vertice) ;
Polygon : :Polygon(list <Point_3D*> list_vertice, QColor color) ;
```

Le constructeur par paramètres permet de construire un polygon à partir des ensembles de paramètres suivant :

- liste des vertices le composant ;
- liste des vertices le composant et couleur le définissant ;

Constructeur par copie

```
Polygon : :Polygon(const Polygon & other_polygon) ;
```

Le constructeur par copie permet de créer un polygone à partir d'un autre polygone.

Destructeur

```
Polygon : :~Polygon(void) ;
```

13.2.2 Opérateur

Comparaison

```
bool Polygon : :operator<(const Polygon & other_polygon) const ;
```

La redéfinition de l'opérateur de comparaison inférieur ($<$) sert dans l'algorithme du peintre afin de trier et de définir l'ordre d'affichage des polygones.

13.2.3 Accesseurs

Accesseurs tableau vertices

```
Point_3D* Polygon : :get_vertice(int int_n) ;
```

Accesseur permettant de récupérer la vertice numéro *int_ n*.

Accesseurs nombre de vertices

```
int Polygon : :get_nb_vertice(void) ;
```

Accesseur permettant de récupérer le nombre de vertice contenu dans la liste des vertices.

13.2.4 Fonctions d'interaction

Affichage

```
bool Polygon : :show(AreaView* area) ;
```

Fonction permettant d'afficher le polygone dans la zone d'affichage *area* en fonction de l'origine de la vue *origin_ view*.

Projection à l'écran

```
bool Polygon : :project(struct_point_sphere* origin_view) ;
```

Fonction permettant de recalculer les nouvelles coordonnées de chacun des points composant le polygone à partir de l'origine de la vue.

Calcul du barycentre

```
Point_3D* Polygon : :barycentre(void);
```

La fonction de calcul du barycentre se contente de renvoyer le barycentre du polygone.

Chapitre 14

Classe Window

Une instance de la classe *Window* représente la fenêtre dans laquelle les objets 3d seront affichés. Cette classe hérite de la classe QWidget du module QT.

14.1 Variables de classe

Les variables de la classe *Window* sont :

AreaView* area_view : La zone d'affichage sur laquelle s'afficheront les objets 3d.

14.2 Fonctions membres

Les fonctions membres sont regroupés en 3 catégories :

14.2.1 Fonctions de classe

Constructeur par paramètres

```
Window : :Window(Scene_3D* scene) ;
```

Le constructeur a deux tâches à accomplir :
- initialisation de QT ;
- création de la fenêtre ;

Le constructeur par paramètre permet de construire la fenêtre en fonction de la scène d'objet 3D *scene*.

Destructeur

```
Window : :~Window(void) ;
```

Le destructeur sert ici pour libérer l'espace mémoire de la zone d'affichage

14.2.2 Accesseurs

Accesseurs zone d'affichage

```
AreaView* Window : :get_areaview(void) ;
```

Cet accesseur permet de récupérer un pointeur vers la zone d'affichage.

14.2.3 Fonctions d'interaction

Initialisation de Qt

```
bool Window : :init_qt(void) ;
```

Cette fonction permet d'initialiser le module QT.

Création de la fenêtre

```
bool Window : :create_window(void) ;
```

Cette fonction permet de créer et d'initialiser la fenêtre QT.

Chapitre 15

Classe AreaView

La classe *AreaView* sert à définir la zone où s'afficheront les objets en 3d dans la fenêtre. Elle hérite de la classe QWidget du module QT.

15.1 Variables de classe

Les variables de la classe *AreaView* sont :

Scene_3D* scene : Scène d'objet 3d qui sera affiché dans la zone d'affichage.
int int_width : Largeur de la zone d'affichage.
int int_height : hauteur de la zone d'affichage.

15.2 Fonctions membres

Les fonctions membres sont regroupés en 3 catégories :

15.2.1 Fonctions de classe

Constructeur par paramètres

```
AreaView : :AreaView(QWidget *parent, Scene_3D* scene3d) ;
AreaView : :AreaView(QWidget *parent, Scene_3D* scene3d, int width, int height) ;
```

Le contructeur permet de contruire la zone d'affichage dépendant de la fenêtre *parent* et de la scene 3d *scene3d*. Il peut aussi construire la zone d'affichage en fonction des dimensions *width* et *height* données en paramètre s'il existe.

Destructeur

```
AreaView : :~AreaView(void) ;
```

15.2.2 Accesseurs

Scène 3D

```
Scene_3D* AreaView : :get_scene_3d() const ;
```

Cet accesseur permet d'avoir un lien vers la scène d'objets 3d.

Dimensions

```
int AreaView : :get_width(void) ;
int AreaView : :get_height(void) ;
```

Ces accesseurs permettent respectivement de récupérer la hauteur et la largeur de la zone d'affichage.

Taille de la zone d'affichage

```
QSize AreaView : :get_size() const ;
```

Cet accesseur permet de récupérer les dimensions de la zone d'affichage au format QSize.

Taille minimum de la zone d'affichage

```
QSize AreaView : :get_minimum_size() const ;
```

Cet accesseur permet de récupérer les dimensions minimal de la zone d'affichage au format QSize.

15.2.3 Fonctions d'intéraction

Affichage des objets

```
void AreaView : :paintEvent(QPaintEvent * event) ;
```

Cette fonction d'affichage est la fonction appelé pour dessiner le contenu de la fenêtre.