

Synthèse d'image : Compte Rendu de Projet

Valentin VERSTRACTE & Evan PETIT

L3 — November 14, 2021

Table des matières

1	Introduction	2
1.1	Fonctionnalités implémentées	2
2	Une conception orientée objet	3
2.1	Le cylindre paramétrique	3
2.2	Les ailes et courbes de béziers	4
2.2.1	La représentation 2D	4
2.2.2	La représentation 3D	5
2.2.3	Le problème de symétrie	5
2.3	La box	6
2.4	Les primitives	6
3	Les textures	6
4	Les lumières	7
4.1	Lumière ponctuelle	7
4.2	Projecteur	7
5	Les animations	8
5.1	Une animation automatique	8
5.2	Une animation manuelle	8
6	Les touches disponibles	8

1 Introduction

Pour commencer, voici un exemple du dragon fini (sans lumières ou textures) - La caméra ou le clavier permettent de déplacer la caméra autour.

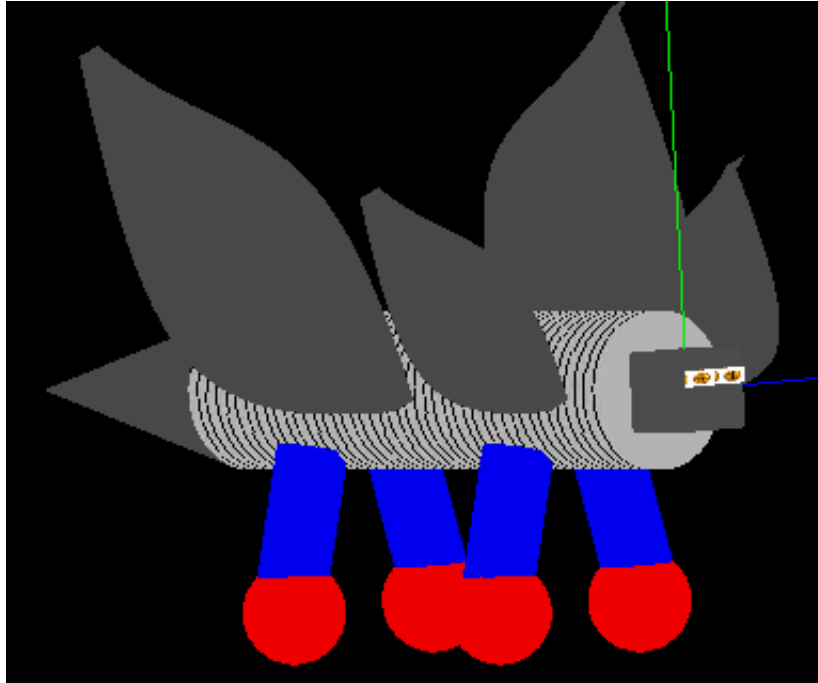


Figure 1: Dragon ou abeille?

1.1 Fonctionnalités implémentées

Toutes les fonctionnalités demandées dans le sujet ont été implémentées. En particulier :

- Il s'agit d'une représentation 3D d'un dragon (à priori)
- Une primitive (ici le cylindre) a été réalisée depuis sa représentation paramétrique.
- Deux textures sont utilisées. Une plaquée sur les yeux du dragon, une enroulée autour de son corps.
- Deux types de lumières sont gérées. Une lumière ponctuelle rouge, et un projecteur vert.
- Il est possible de zoomer à l'aide de 'z', de dézoomer avec 'Z', de diriger la caméra à l'aide des touches directionnelles. *Le reste des touches est décrit dans la section 6. touches disponibles.*
- Une animation manuelle qui permet de baisser ('n') ou lever ('h') la queue du dragon.
- Une animation automatique qui fait battre les ailes du dragon tout au long du déroulement de l'application.
- **En bonus** : Les ailes du dragon sont modélisées à l'aide de courbes de Bézier.

2 Une conception orientée objet

Les objets composant le dragon sont de trois types. Primitives connues par GL (Sphere...), primitives définies par leur représentation paramétrique (ici les cylindres) ainsi que les ailes du dragon qui sont des solides construits depuis des courbes de Bézier.

Afin de faciliter l'écriture et la lecture du code - et profiter d'une fonctionnalité agréable de C++ - ces éléments sont représentés par des classes. Chacune d'elle est dérivée de la classe Object. Cette **classe Object** permet de contenir la position d'un objet, et d'effectuer la translation, la rotation, définir les couleurs et si "l'objet" est lié à une texture ou non. Pour appliquer tout cela, il suffit d'appeler **this->onDraw()** qui s'occupe d'afficher l'objet en particulier dans la fonction d'actualisation. Ensuite, dans ces classes dérivée, le constructeur initialise les différents attributs. Il appelle la fonction draw qui, comme l'indique son nom, dessine l'objet en fonction des paramètres (coordonnées, angle, couleurs...). Cette structure aussi efficace que pratique exige cependant des conditions : La méthode draw doit commencer par un `glPushMatrix()` et `this->onDraw()` (hérité de objet) et doit finir par un `glPopMatrix()`. Ainsi on s'assure que l'objet est indépendant du reste, ce que l'on souhaite.

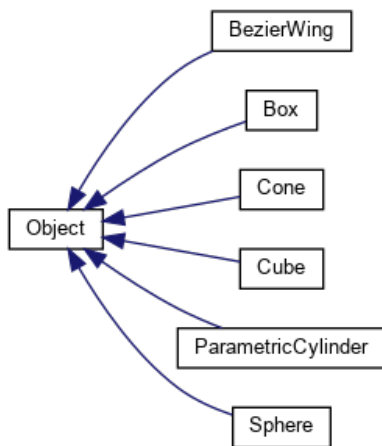


Figure 2: Diagramme de classes

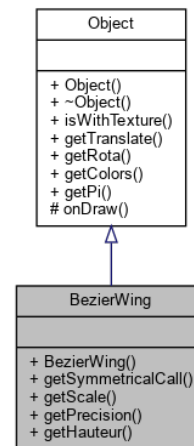


Figure 3: Exemple diagramme classe bézier

2.1 Le cylindre paramétrique

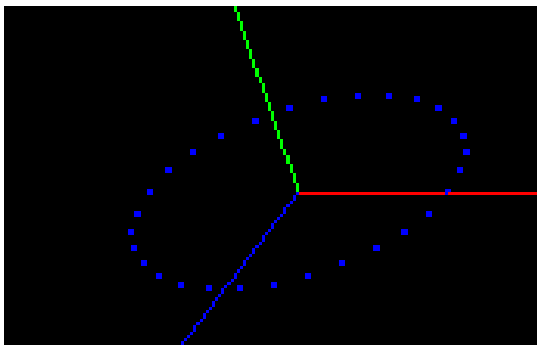


Figure 4: 30 points évalués

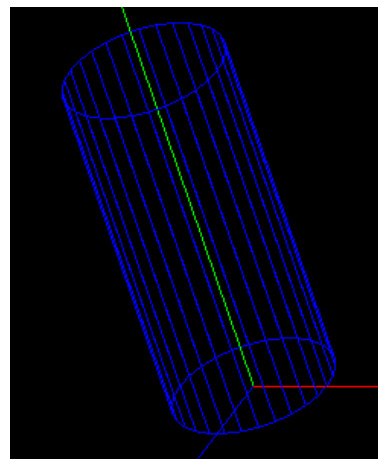


Figure 5: La construction des polygones

Le cylindre paramétrique est celui vu en cours. Il dépend ici de 3 paramètres importants : le rayon, la hauteur et la précision. Le rayon et la hauteur sont plutôt révélateurs de ce qu'ils sont. Cependant comme on utilise une équation paramétrique il est nécessaire d'évaluer les points à un instant t . La précision représente justement le nombre d'évaluation que l'on va effectuer. Elle représente aussi le nombre de faces que le cylindre va obtenir. Plus la précision est élevée, plus la surface du cylindre est lisse et ronde - au prix d'un temps de calcul un peu plus élevé.

Ainsi pour construire ce cylindre plusieurs étapes sont importantes :

- On évalue les points autour d'un cercle de rayon donné en paramètre. On obtient pour chaque point une coordonnée (x,z) que l'on stocke dans un tableau à l'indice t . Exemple de 30 points figure 4
- On crée des polygones à partir de ces points :
 - La partie inférieure gauche est égale à $x(t), 0, z(t)$
 - La partie inférieure droite est égale à $x(t+1), 0, z(t+1)$
 - La partie supérieure gauche est égale à $x(t), \text{hauteur}, z(t)$
 - La partie supérieure droite est égale à $x(t+1), \text{hauteur}, z(t+1)$

Exemple de polygones avec les 30 points précédents (vision fil de fer) figure 5

- On ferme la partie inférieure et supérieure en créant 2 polygones en utilisant tous les points contenus dans t . On le fait d'abord pour un premier polygone à la hauteur 0, puis pour le deuxième à la hauteur donnée en paramètre

2.2 Les ailes et courbes de béziers

2.2.1 La représentation 2D

Les ailes sont construites à partir de deux courbes de Bézier cubiques - *vu en synthèse d'image au semestre 3*. Une courbe de Bézier cubique est une courbe polynomiale paramétrique définie uniquement à partir des coordonnées de 4 points de contrôle. On connaît l'équation pour évaluer un point d'une courbe de Bézier :

$$P(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3$$

Où P_0, P_1, P_2, P_3 représente les points de contrôle. Ainsi tout comme notre cylindre paramétrique, cette classe aura besoin d'un paramètre précision pour déterminer le nombre de points à évaluer, la "**précision**". Pour obtenir les coordonnées adéquates, on modélise déjà sur un logiciel de géométrie dynamique (ici Kig) la forme de l'aile souhaitée.

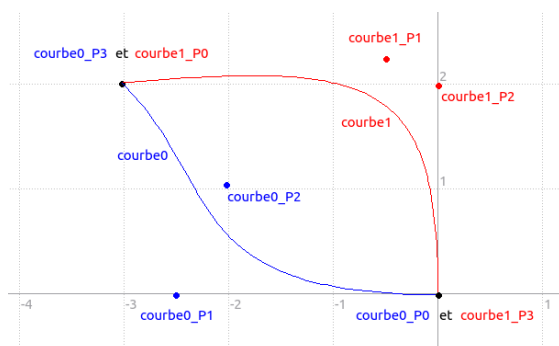


Figure 6: Courbes de Bézier sur Kig

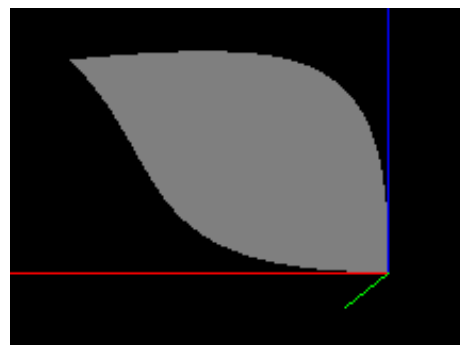


Figure 7: Courbes de Bézier 2D OpenGL

La figure 6 montre la représentation 2D et le choix des différents points de contrôle pour les deux courbes. Pour représenter cette aile avec OpenGL, on va construire un polygone en 2D où chaque point du polygone appartient à une de nos deux courbes de bézier. On commence par les points de la courbe en bleu puis on finit avec les points de celle en rouge. On commence de (0,0), on monte jusqu'en (-3,2) et on redescend jusqu'en (0,0). On obtient une aile en 2D cf figure 7.

Pour simplifier les dimensions, deux opérations supplémentaires sont effectuées au tout début. Les points de contrôle ont un maximum de 3 en x et z. On divise d'abord tous les points de contrôle par 3 pour que la taille maximum de l'aile soit de 1 en x et z. Ensuite, on ne souhaite pas forcément que les ailes soient aussi petites. On rajoute un paramètre dimension dans notre classe et on multiplie les coordonnées de tous nos points de contrôle par ce paramètre. Ainsi la taille maximum en x et z sera la valeur de "dimension".

2.2.2 La représentation 3D



Figure 8: Vue 3D 0

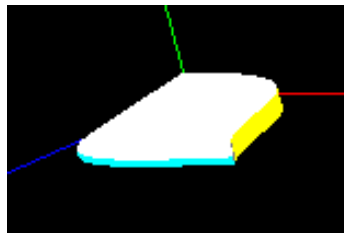


Figure 9: Vue 3D 1

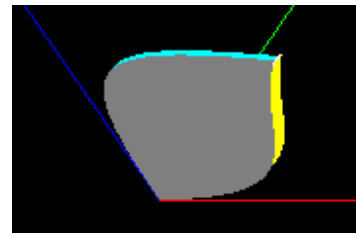


Figure 10: Vue 3D 2

En gris, la même aile et "première surface" que la figure 7. En blanc "deuxième surface". En bleu "troisième surface". En jaune "quatrième surface"

La représentation 3D est constituée de 4 surfaces. Les deux premières sont deux ailes 2D de hauteur 0 et une deuxième de hauteur de 0.4 dans le plus grand des cas. La deuxième est légèrement "penchée". Sa hauteur vers la courbe bleue (cf figure 6) commence vers 1/3 de la hauteur max tandis que vers la courbe rouge, elle est de 0.4 (hauteur max). La troisième est constituée des points de la courbe bleue de la figure 6 en bleu qui commence à la hauteur de la première aile 2D jusqu'à la hauteur de la deuxième aile 2D. La quatrième est équivalente hormis qu'on l'applique pour la courbe rouge de la figure 6.

2.2.3 Le problème de symétrie

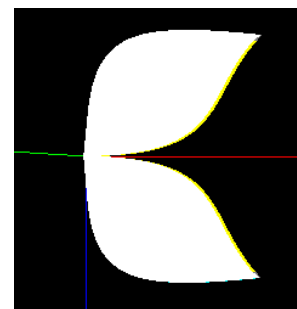
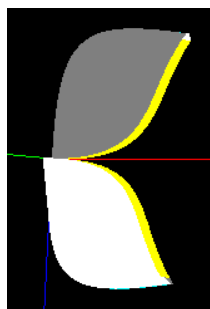
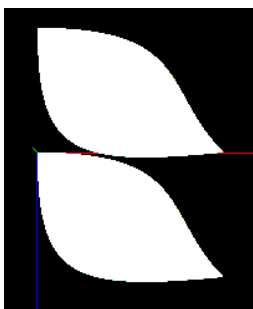


Figure 11: Symétrie incorrecte Figure 12: Tentative de rotation Figure 13: Problème résolu

Un problème inattendu pour essayer de créer des ailes comme dans la figure 13 (résultat final) a été tout d'abord celui de la figure 11. Une rotation a été naïvement tentée pour résoudre le problème cf figure 9 mais le résultat n'est toujours pas bon (notamment à cause de la hauteur de 1/3 (cf figure 9 le bleu) "en dessous au lieu de haut dessus"). Pour obtenir la figure 13 et résoudre le problème il a été décidé de jouer avec les coordonnées des points de contrôle. Il suffit de prendre l'opposé en z pour obtenir une deuxième aile symétrique. Ainsi un booléen **symmetrical** a été rajouté à notre classe pour préciser si on souhaite prendre des points de coordonnées avec l'opposé en z.

2.3 La box

La box est un parallélépipède rectangle composé d'une longueur, largeur et hauteur comme décrit figure 14. On recevra ainsi ces 3 composantes en paramètre de notre constructeur d'objet. L'objet est centré en (0,0,0). Sa construction est simple. Pour une face on envoie dans une fonction les quatre coordonnées d'une face d'un cube (voir coordonnées de la figure 15). On multiplie ensuite respectivement les coordonnées x , y et z par la moitié de : la longueur, largeur et hauteur. On crée à partir de ces points un polygone qui nous donne une face de notre box. On répète l'opération pour toutes les faces et on obtient notre box centrée en (0,0,0).

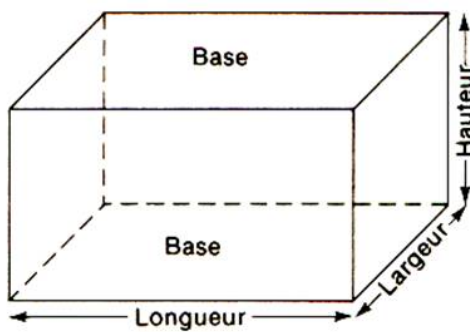


Figure 14: Représentation de la box

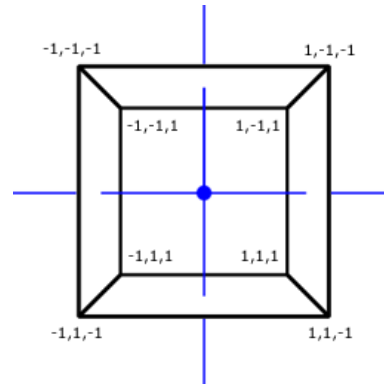


Figure 15: Coordonnées du carré servant "de base"

2.4 Les primitives

Il n'y a pas grand à chose à dire sur les primitives utilisées. Pour que le code soit cohérent, il a été décidé d'encapsuler ces primitives dans des classes C++, afin que la syntaxe de la construction d'un objet (bras, tête, oeil, etc...) soit la même peu importe cet objet, que ce soit une primitive de GL, un cylindre paramétrique, une aile représentées par des courbes de bézier, une box.

Les primitives natives à GL utilisées sont :

- Le cône
- Le cube (dont on aurait pu se passer en donnant une longueur / largeur / hauteur de 1 à la box)
- La sphère

3 Les textures

Une texture est représentée par une classe dans notre conception. Son constructeur prend en paramètre un string qui s'occupe de charger en mémoire la texture depuis son chemin dans la machine. Ensuite, pour décrire qu'il faut utiliser cette texture il suffit d'appeler `enableTexture()` sur l'objet instancié. Cette fonction appelle simplement `glTexImage2D()` de glut avec les paramètres nécessaires.

Comme chaque classe hérite de objet, elle peut utiliser la fonction `isWithTexture()` pour savoir si elle a besoin de traduire des coordonnées en "coordonnées de texture" (texels). Chaque classe gère localement cette traduction.

4 Les lumières

Les lumières sont une partie assez succincte de l'application. Elles ne nécessitent pas l'encapsulation dans une classe, juste d'une fonction dans la classe principale qui est appelée à chaque rafraîchissement.

Dans le code, le nom de cette fonction est **manageLights** et prend en paramètre un entier. Ici, en particulier, il s'agira soit de 0, 1 ou 2. L'appel de la fonction permet d'afficher en fonction du paramètre et d'une condition if sur l'entier : 0 pour aucune lumière (à part celles par défaut), 1 pour une lumière ponctuelle rouge située derrière et en dessous du dragon, 2 pour un projecteur vert lumineux situé au même endroit.

Dans les deux cas, on utilise le modèle du cours :
Définir un vecteur position (homogène) qui contient les coordonnées de la source de lumière. Ensuite, trois autres vecteurs qui permettent de définir les composantes diffuses, ambiantes et spéculaires des lumières. De cette manière, on peut changer les propriétés des lumières à l'aide de la fonction **gLightfv**

4.1 Lumière ponctuelle

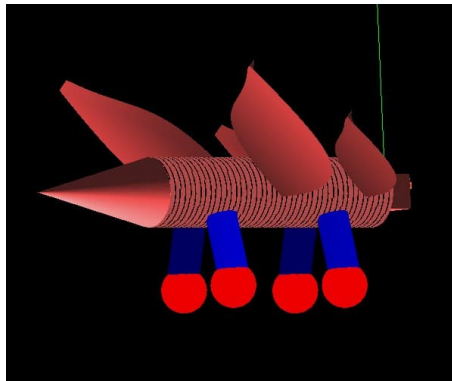


Figure 16: Lumière ponctuelle (venant de derrière/dessous le dragon)

La lumière ponctuelle correspond à **GL LIGHT0**. Elle est définie par des composantes diffuse et ambiante légèrement rouges (0.5 0.2 0.2), ainsi que spéculaire blanche. Puisqu'il s'agit du type de lumière le plus classique, il n'y a pas grand chose d'autre à faire : Les rayons lumineux partent uniformément dans toutes les directions autour de la source.

4.2 Projecteur

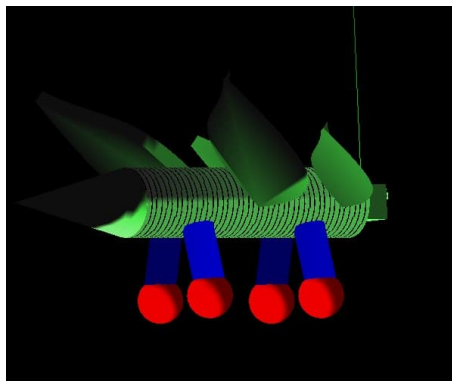


Figure 17: Projecteur (venant du même endroit que la figure précédente)

Le projecteur correspond à **GL LIGHT1**. Il est défini par des composantes similaires à la lumière précédente, mais ici il s'agit de vert un peu plus lumineux (0.2 0.8 0.2). Un projecteur est défini par trois propriétés supplémentaires : **CUTOFF** l'angle d'ouverture, **EXPONENT** le coefficient qui définit la perte d'intensité lumineuse aux bords du cône du projecteur, ainsi que **DIRECTION** le vecteur direction de la lumière.

Dans ce projet, l'angle est petit (15°) : On peut voir sur la figure qu'une grande partie de la queue du dragon et de ses ailes ne sont pas comprises dedans. La direction est (-2,1,0) - C'est à dire directement dirigé vers la tête du dragon qui est au centre de la scène (puisque la position du projecteur est 20,-10,0)

5 Les animations

5.1 Une animation automatique













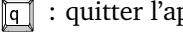
L'animation automatique se porte sur les ailes du dragon. On incrémente un angle de + ou - 25 ° ce qui donne au dragon l'impression de voler en battant des ailes. La logique algorithmique est plutôt simple. On incrémente un tout petit l'angle dans la fonction anim (appelée par glut chaque fois qu'il ne fait rien). Si l'angle est supérieur à 25 on décrémente. Si l'angle est inférieur à -25 on incrémente.

L'animation étant très demandeuse en ressources, il est possible de la désactiver / activer avec la touche A.

5.2 Une animation manuelle

Cette animation permet de baisser ou lever la queue du dragon en fonction des appuis sur les touches h et n. Elle fonctionne comme pour l'animation précédente : On incrémente ou décrémente d'un petit angle l'angle de l'objet **queue**.

6 Les touches disponibles

-  : affichage du carré plein
-  : affichage du mode de fil de fer
-  : affichage en mode sommets seuls
-  : permet de zoomer
-  : permet de dézoomer
-  : élève la queue du dragon
-  : abaisse la queue du dragon
-  : active ou désactive l'animation automatique
-  : désactive toutes les lumières
-  : active **GL LIGHT0** qui correspond à la lumière ponctuelle
-  : active **GL LIGHT1** qui correspond au projecteur
-  : quitter l'application
-  : déplace la caméra en haut, en bas, à droite, à gauche