

# Synthèse d'image : compte rendu

Valentin VERSTRACTE & Evan PETIT

L3 — November 10, 2021

## Introduction

### 1 Une conception orienté objet

Chaque primitive, courbe de bézier et courbe paramétrique sont représentées par des classes. Chacune d'elles est dérivée de la classe objet. Cette classe objet permet de contenir et d'effectuer la translation, la rotation, 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 de tout le reste. Ensuite, dans ces classes dérivées, le constructeur initialise les différents attributs. Il appelle la fonction draw qui, comme l'indique son nom, dessine. 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()`.

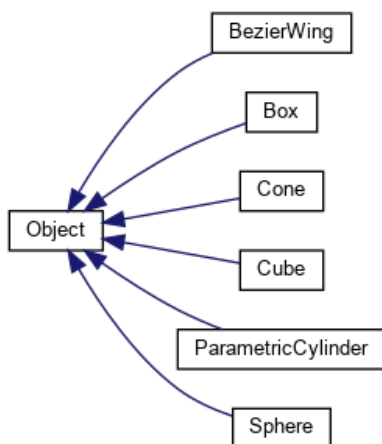


Figure 1: Diagramme de classes

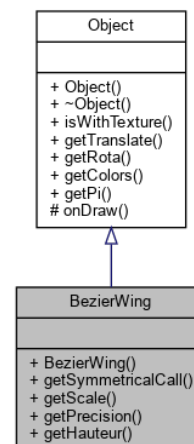


Figure 2: Exemple diagramme classe bézier

#### 1.1 Le cylindre paramétrique

Le cylindre paramétrique dépend 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.

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 3
- On crée des polygones à partir de ces points :
  - La partie inférieure gauche est égale à  $x(t), 0, z(t)$

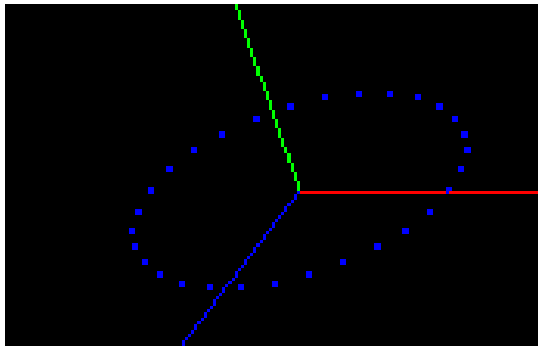


Figure 3: 30 points évalués

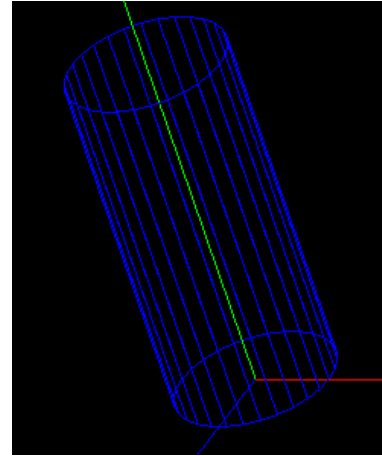


Figure 4: La construction des polygones

- La partie inférieur droite est égale à  $x(t+1), 0, z(t+1)$
- La partie supérieur gauche est égale à  $x(t), hauteur, z(t)$
- La partie supérieur droite est égale à  $x(t+1), 0, z(t+1)$

Exemple de polygones avec les 30 points précédent (vision fil de fer) figure 4

- On ferme la partie inférieur et supérieur en créent 2 polygones en utilisant tout les points contenu 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

## 1.2 Les ailes, deux courbes de bézier

### 1.2.1 La représentation 2D

Les ailes sont construites à partir de deux courbes de bézier cubiques. Une courbe de bézier cubique est une courbe polynomiale paramétrique à partir de 4 points de contrôle. On rappelle 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.

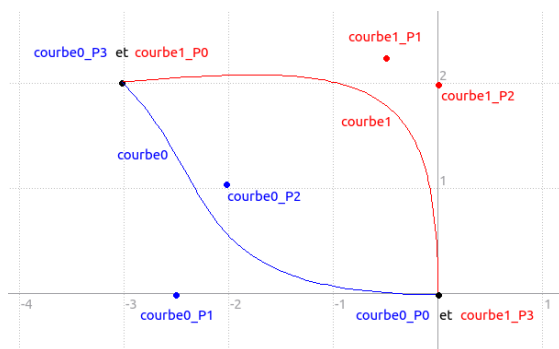


Figure 5: Courbes de bézier sur kig

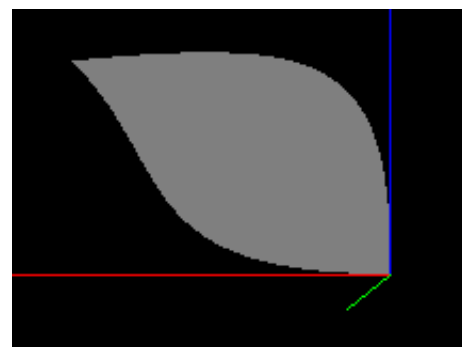


Figure 6: Courbes de bézier 2D openGL

La figure 2 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 6.

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 tous nos points de contrôle par ce paramètre. Ainsi la taille maximum en x et z sera la valeur de "dimension".

### 1.2.2 La représentation 3D



Figure 7: Vue 3D 0

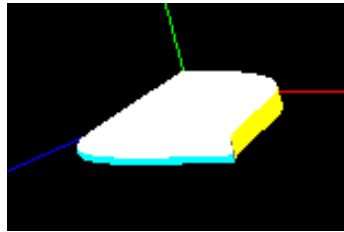


Figure 8: Vue 3D 1

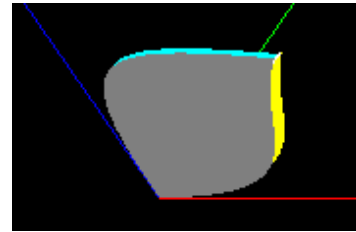


Figure 9: Vue 3D 2

En gris, la même aile et "premier polygone" que la figure 6. En blanc "deuxième polygone". En bleu "troisième polygone". En jaune "quatrième polygone"

La représentation 3D est constituée de 4 polygones. Les deux premiers 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 5) commence vers 1/3 de la hauteur max tandis que vers la courbe rouge, elle est de 0.4 (hauteur max). Le troisième est constitué des points de la courbe bleue de la figure 5 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 figure 5.

### 1.2.3 Le problème de symétrie

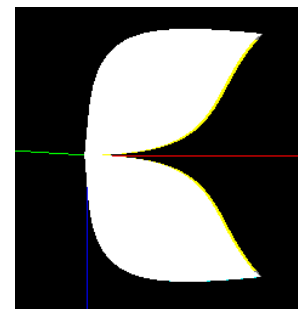
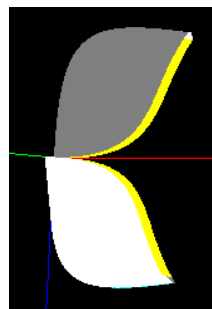
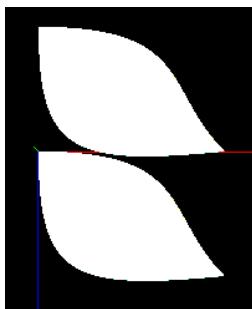


Figure 10: Symétrie incorrecte Figure 11: Tentative de rotation Figure 12: Problème résolu

Un problème inattendu pour essayer de créer des ailes comme dans la figure 12 (résultat final) a été tout d'abord celui de la figure 10. 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 8 le bleu) "en dessous au lieu de haut dessus"). Pour obtenir la figure 12 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 symetricall a été rajouté à notre classe pour préciser si on souhaite prendre des points de coordonnées avec l'opposé en z.

### 1.3 La box

### 1.4 Les primitives

## 2 Les textures

Une texture est représenté par une classe. Son constructeur prend en paramètre un string qui s'occupe de charger en mémoire la texture. 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 nécessaire.

## 3 Les animations













### 3.1 Une animation automatique

L'animation automatique se porte les ailes du dragon. On incrémente un angle de  $+ \text{ ou } - 25^\circ$  se qui donne au dragon l'impression de voler. La logique algorithmique est plutôt simple. On incrémente un tout petit l'angle dans la fonction `anim` ( appeler par glut chaque fois qu'il ne fait rien ). Si l'angle est supérieur à 25 on décremente. Si l'angle est inférieur à -25 on incrémente.

### 3.2 Une animation manuelle

L'animation n'est pas très impressionnante. En peut juste baisser ou lever la queue en fonction des touches `h` et `n`

## 4 Les touches disponibles

-  : affichage du carré plein
-  : affichage du mode de fil de fer
-  : affichage en mode de sommets seuls
-  : permet de zoomer
-  : permet de dézoomer
-  : élève la queue du dragon
-  : abaisse la queue du dragon
-  : quitter l'application
-     : déplace la caméra en haut, en bas, à droite, à gauche