

Synthèse d'image : compte rendu, le dragon abeille

Valentin VERSTRACTE & Evan PETIT

L3 — November 11, 2021

Table des matières

1	Une conception orienté objet	1
1.1	Le cylindre paramétrique	1
1.2	Les ailes, deux courbes de bézier	2
1.2.1	La représentation 2D	2
1.2.2	La représentation 3D	2
1.2.3	Le problème de symétrie	3
1.3	La box	3
1.4	Les primitives	3
2	Les textures	4
3	Les lumières	4
4	Les animations	4
4.1	Une animation automatique	4
4.2	Une animation manuelle	4
5	Les touches disponibles	5
6	Les critiques/Le lore	5

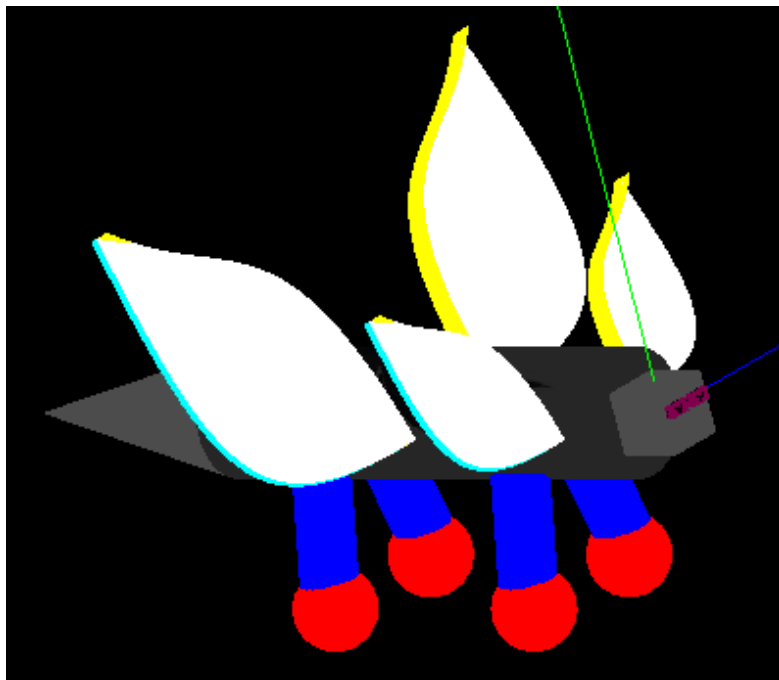


Figure 1: A CHANGER

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()`.

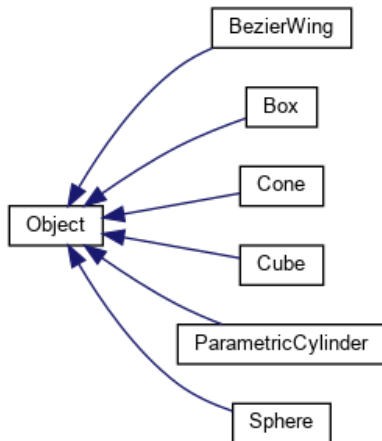


Figure 2: Diagramme de classes

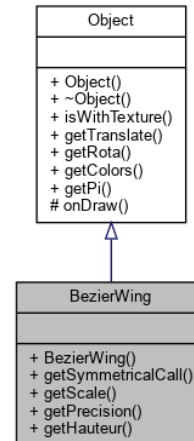


Figure 3: Exemple diagramme classe bézier

1.1 Le cylindre paramétrique

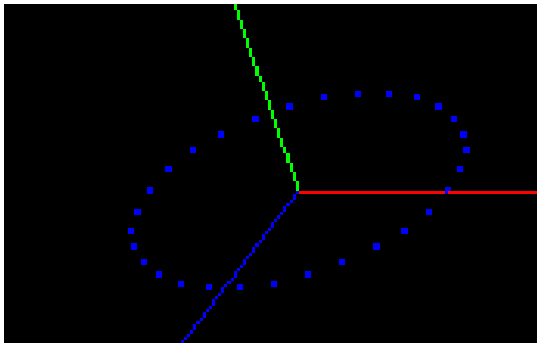


Figure 4: 30 points évalués

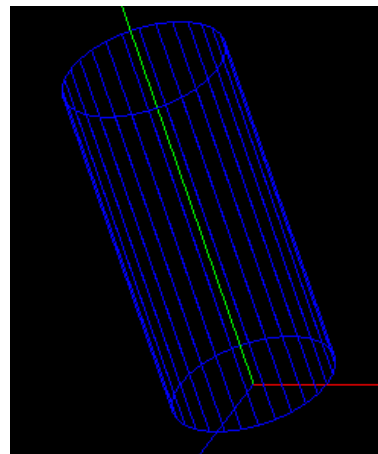


Figure 5: La construction des polygones

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 4

- On crée des polygones à partir de ces points :
 - La partie inférieur gauche est égale à $x(t), 0, z(t)$
 - La partie inférieur droite est égale à $x(t+1), 0, z(t+1)$
 - La partie supérieur gauche est égale à $x(t), \text{hauteur}, z(t)$
 - 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 5

- On ferme la partie inférieur et supérieur en créant 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.

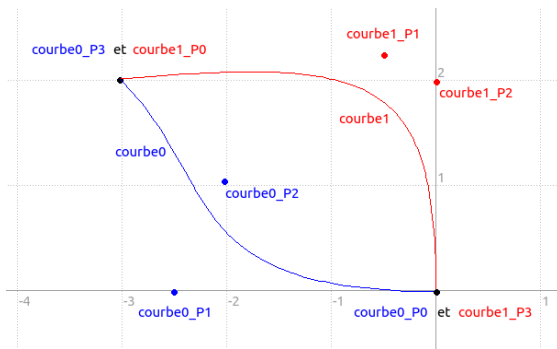


Figure 6: Courbes de bézier sur kig

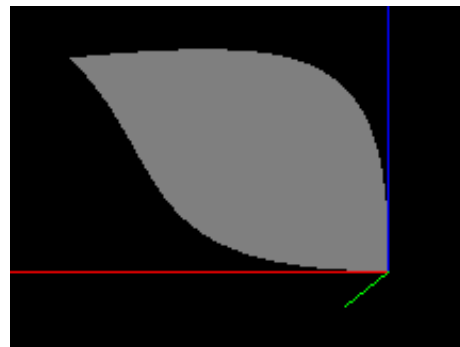


Figure 7: Courbes de bézier 2D OpenGL

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 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

En gris, la même aile et "premier polygone" que la figure 7. 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



Figure 8: Vue 3D 0

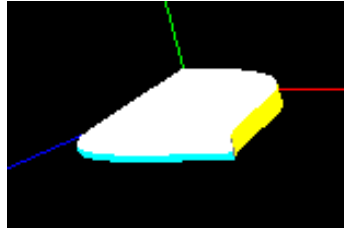


Figure 9: Vue 3D 1

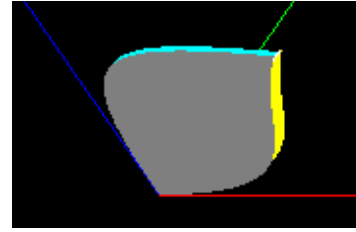


Figure 10: Vue 3D 2

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). Le troisième est constituer des points de la courbe bleue de la figure 6 en bleue 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 6.

1.2.3 Le problème de symétrie

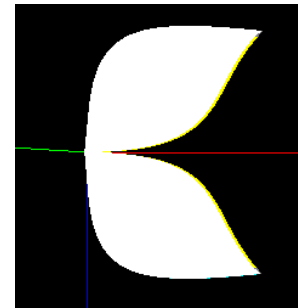
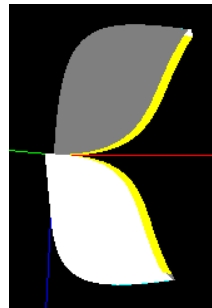
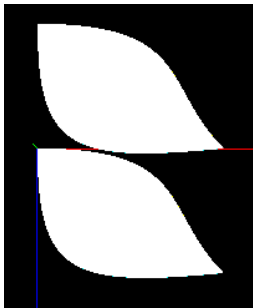


Figure 11: Symétrie incorrecte Figure 12: Tentative de rotation Figure 13: Problème résolu

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 tenté 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 `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

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 classe. 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 (coordonnées tel que 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 $(0,0,0)$.

1.4 Les primitives

Il n'y a pas grand à chose à dire sur les primitives utiliser. Elles sont juste entouré de cette conception objet ce qui, aurait très bien pu ce faire sans. Il a tout de même été choisit de les moulés dans cette conception pour garder le code propre et lisible.

Les primitives utilisés sont :

- Le cone

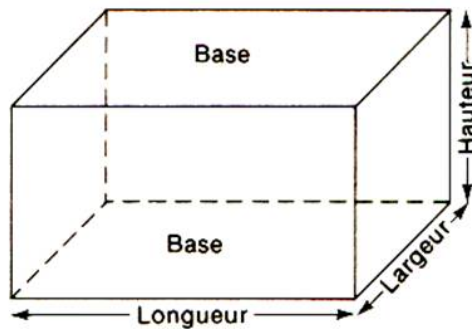


Figure 14: Représentation de la box

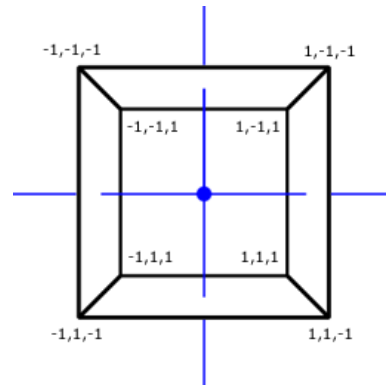


Figure 15: Coordonnées du carré servant "de base"

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

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.

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". Chaque classe gère localement cette traduction.

3 Les lumières

4 Les animations

4.1 Une animation automatique



L'animation automatique se porte les ailes du dragon. On incrémente un angle de + ou - 25 ° 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.












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

4.2 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

5 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
-  : active ou désactive l'animation automatique
-  : quitter l'application
-     : déplace la caméra en haut, en bas, à droite, à gauche

6 Les critiques/Le lore

Comme tout bon scientifique nous avons soumis notre projet à la critique (critique visuel / avis de proche). La première et plus récurrente a été : "mais c'est une abeille". Après des heures de travail acharné elle a été la plus blessante mais réaliste. Son surnom et sa légende sont d'ailleurs venu de là : "le dragon abeille". La deuxième critique était au niveau de la taille de sa tête trop petite. Ce choix est justifié par sa carrure importante, nous voulions être sur qu'il ne prenne pas la grosse tête ...