Pinon Alexandre
Perion Maxence

2021-2022

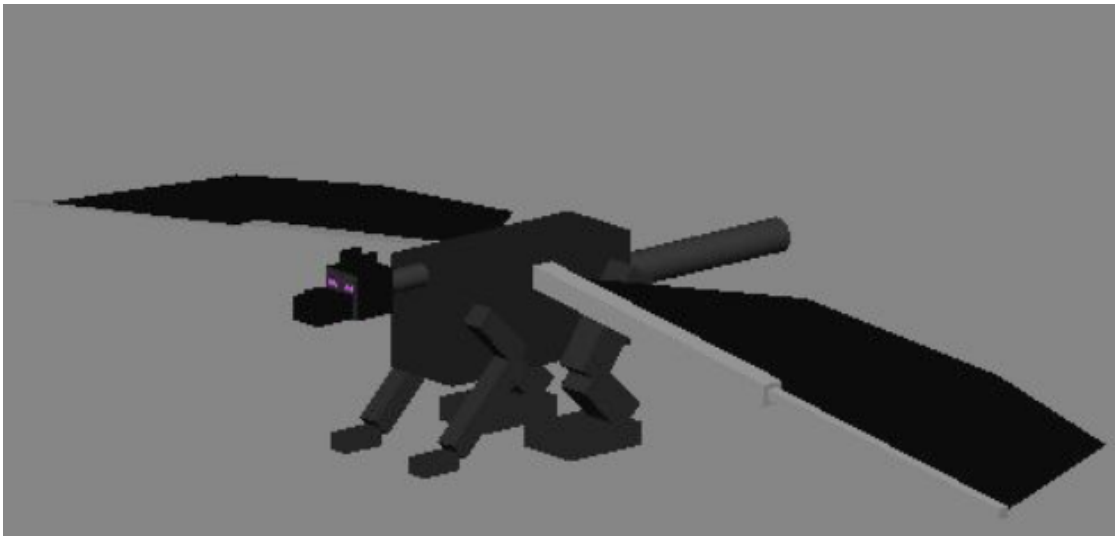
Projet de synthèse d'images

Introduction : Pour ce projet de synthèse d'images, nous devions réaliser un dragon à l'aide d'OpenGL (GLUT) et nous allons présenter dans ce rapport la construction de notre dragon, ses animations et ses lumières.

Nous nous sommes inspirés de « l'ender dragon » du jeu vidéo Minecraft :

L'ender dragon

Pour au final, obtenir ceci :



Résultat final

Nous nous sommes basés sur un repère classique pour modéliser ce dragon : l'axe X comme abscisse (côté), l'axe Y comme hauteur et l'axe Z comme côte (profondeur).

Nous utilisons quelques propriétés d'openGL : GL_DEPTH_TEST pour la profondeur, GL_NORMALIZE pour normaliser automatiquement les normales (meilleur rendu pour les lumières) et GL_LIGHT_MODEL_VIEWER (meilleur rendu pour les lumières).

Pour nous faciliter la tâche, nous avons créé une classe Texture qui s'occupe de tout gérer pour une texture donnée. Elle va la charger et la stocker via son constructeur puis nous n'aurons plus qu'à lui demander de s'activer ou de se désactiver.

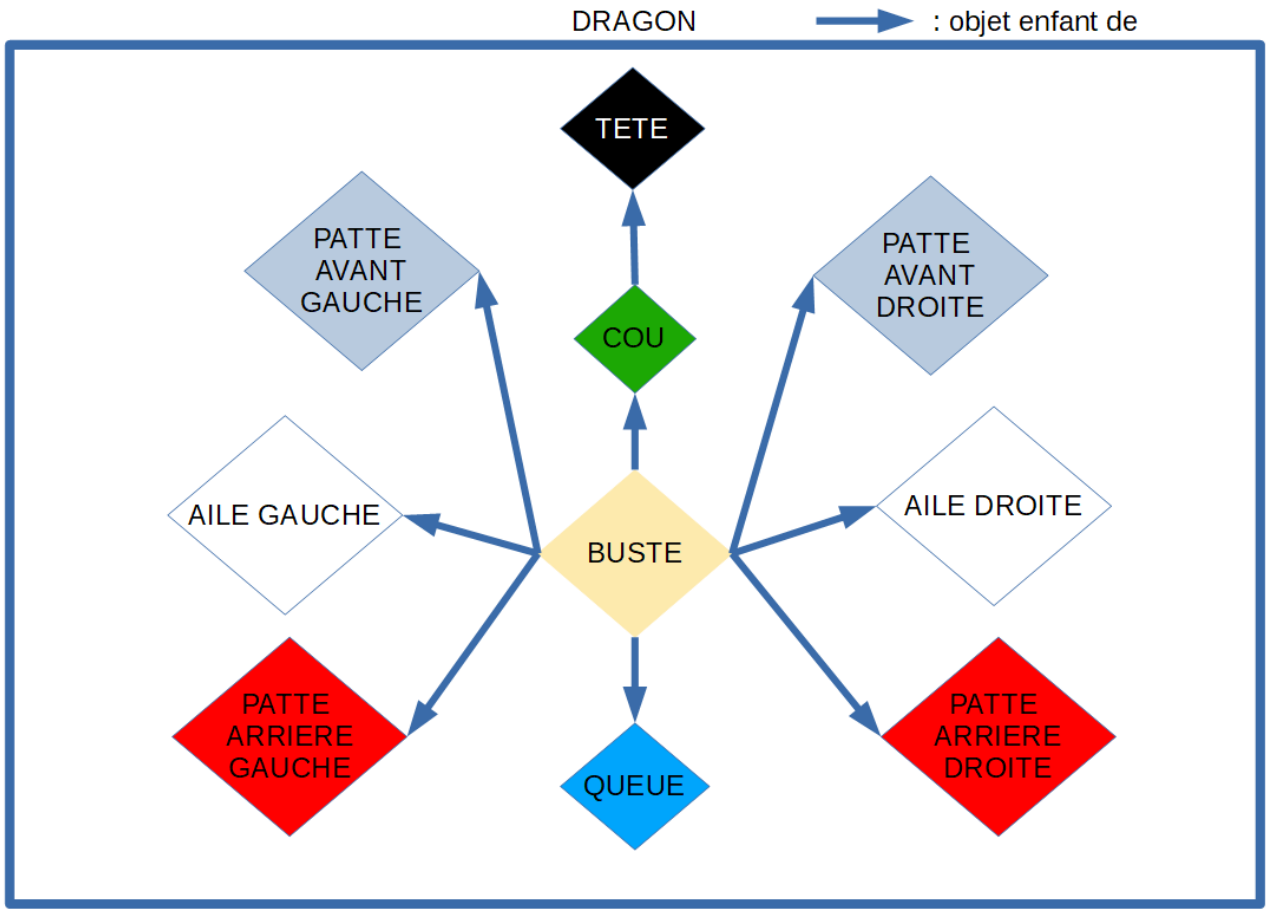
Pour faciliter le déplacement dans la scène, il est possible de tourner autour du dragon avec les flèches directionnelles mais pour encore plus de confort, nous utilisons une méthode plus sophistiquée : au lieu de se déplacer lors de la détection de l'appui d'une touche donnée, ce qui basiquement ne permet le déplacement que dans une direction à la fois, nous utilisons un tableau qui va stocker l'état de certaines touches (détecté par les évènements « touche appuyée » et « touche relachée ») pour ensuite appliquer les transformations demandées selon toutes les touches. Ceci nous permet de nous déplacer selon plusieurs touches simultanément.

Avant de démarrer notre construction, nous avons établi un schéma de la structure globale que nous voulons obtenir pour la construction :

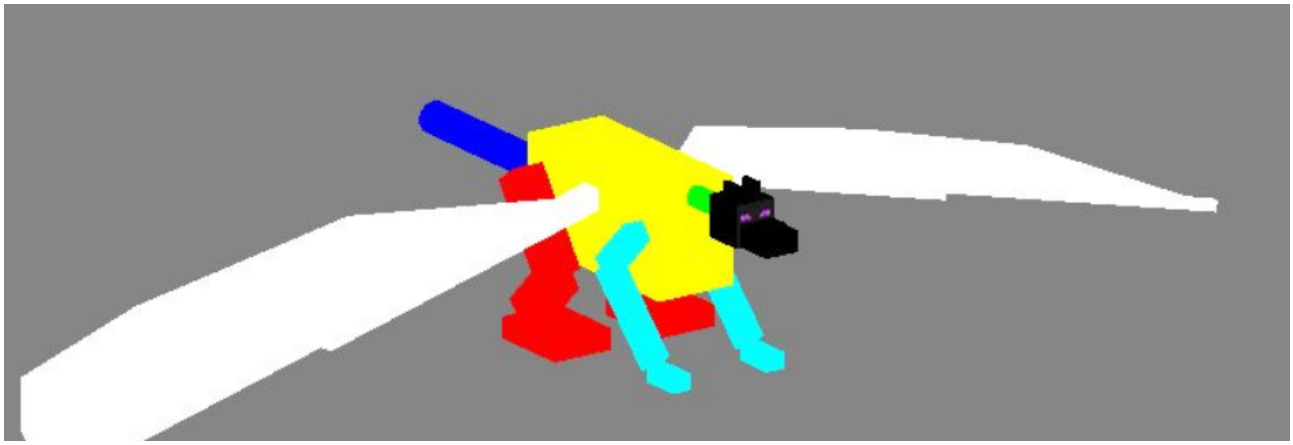
Schéma des différentes formes à utiliser

I. La construction du dragon

Notre créature est composées de plusieurs parties plus ou moins indépendantes les unes des autres : le buste, la queue, le cou, la tête, les pattes et les ailes ; comme on peut le voir sur la hiérarchie des parties.



Hiérarchie des parties du dragon



Affichage coloré des différentes parties du dragon

1. Le buste

Le buste est la partie centrale de la créature, il s'agit de son ventre. Nous avons choisis de le représenter simplement avec un cube, déformé par un scale selon des variables de largeur, hauteur et de longueur (largeurCorp, longueurHauteurCorp, longueurCorp). La variable représentant sa dimension en hauteur (axe Y) a été nommée « longueurHauteurCorp » afin de ne pas la confondre avec une hauteur, qui aurait pu représenter un déplacement vertical par rapport au « sol ».
Une fois le buste réalisé, nous dessinons ses parties enfants.

2. La queue

La queue est représentée par un cylindre alongé sur l'axe z selon une variable longueurQueue et de rayon rayonQueue. Afin de se positionner au bout du corps, nous effectuons une rotation de la matrice de 180 degrés sur l'axe y pour pointer vers l'arrière et une translation de la moitié de la longueur du corps pour se positionner au bout du buste sur l'axe Z et de hauteurQueue sur l'axe Y pour la hauteur. Il faut ensuite effectuer une autre translation de la moitié de la longueur de la queue afin qu'un bout du cylindre touche parfaitement l'arrière du buste (le point de pivot de notre cylindre est situé à son centre et non à sa base).

Le cylindre utilisé n'est pas une primitive de glut (glutSolidCylinder) mais une figure réalisée algorithmiquement par nos soins à l'aide de sa représentation paramétrique. Pour cela, nous avons créé une classe Cylindre qui se charge de dessiner le cylindre en question selon ses paramètres (rayon et longueur) dès qu'on crée une de ses instances.
Pour créer les points du cylindre, nous commençons par discrétiser 2 cercles, un pour chaque « couvercle », en utilisant la formule point = (rayon * cos(angle), rayon * cos(angle), hauteur). L'angle utilisé varie entre 0 et 2pi avec un pas dépendant de la discrétisation en question et la hauteur dépend du couvercle (haut ou bas) et de la longueur voulu pour le cylindre.

Il faut ensuite créer les faces du cylindre en reliant 2 points consécutifs d'un couvercle aux 2 points consécutifs correspondant sur l'autre couvercle, ce qui est réalisé en itérant parmi le nombre de subdivision et en sélectionnant un point, le point suivant modulo le nombre de subdivision (au cas où le point serait le dernier, il faut le raccrocher au premier), le point homologue sur l'autre couvercle (on ajoute le nombre de subdivision puisqu'il y a un point pour chaque subdivision) puis le point homologue sur l'autre couvercle de notre premier point. Dans cet ordre, nous respectons le sens anti-horaire, déterminant la direction de la face (FRONT).

Pour finir notre cylindre, il ne nous reste plus qu'à fermer les couvercles. Cette fois, nous allons réaliser des faces à 3 sommets (et non 4 comme précédement) pour plus de facilité. Il s'agit donc de sélectionner chaque point d'un couvercle, le point central et le point suivant le premier. Cet ordre est dans le sens anti-horaire pour le couvercle du bas mais il faut changer de sens pour respecter la direction de la face pour le couvercle du haut : on prends le point suivant, le point central puis le point sélectionné. Le point central de chaque couvercle est le centre de chaque cercle, de coordonnées (0, 0, hauteur) avec hauteur la hauteur du couvercle en question.

Puisque les différentes parties du dragon interagissent avec la lumière, nous devons également ajouter des normales lors de l'ajout des faces de notre cylindre. Pour les couvercles, nous avons juste à indiquer le sens correspondant au couvercle : si c'est le couvercle du haut, la normale va dans le sens positif de la profondeur (axeZ), si c'est le couvercle du bas alors elle va dans le sens négatif. Pour les autres faces, nous calculons directement la normale avec le produit vectoriel de deux arêtes de la face.

3. Le cou

Tout comme la queue, le cou est un cylindre (algorithmique) de longueur longueurCou et de rayon rayonCou. Il est placé tout comme la queue mais dans l'autre sens : il pointe vers l'avant du dragon, à une hauteur de hauteurCou. Une fois le cylindre placé, on va se déplacer au bout de celui ci, en rajoutant encore une fois une translation sur l'axe Z de la moitié de sa longueur pour se positionner correctement pour la suite et passer à sa partie enfant : la tête.

4. La tête

La tête est dessinée au bout du cou et est composée d'un cube de côté dimTete. Pour qu'elle soit posée exactement au bout du cou, nous devons translater de la moitié de sa dimension sur l'axe Z. Pour afficher des yeux violets terrifiants comme sur notre référence, nous placardons une texture sur la face avant de ce cube. Il s'agit donc d'un petit carré aux dimensions de la tête qui est affiché exactement sur la face. Nous utilisons également différents petits cubes aux dimensions proportionnelles à la taille de la tête pour dessiner des crêtes et ses machoires.

5. Les pattes

Le dragon compte 4 pattes : 2 avant et 2 arrières. Elles sont composées chacune de 3 cubes : la cuisse, le mollet et le pied. Les pattes arrières sont indiques entre elles, à l'exception de leur position de part et d'autre du buste du dragon ; et de même pour celles avant.

La première chose à faire pour dessiner les jambes arrières est de se positionner à leur emplacement grâce à une translation sur l'axe Z dans le sens négatif de la moitié de la longueur du buste de la créature auquel nous ajoutons une variable d'ajustement : avancementJambesArrieres pour pouvoir les déplacer par rapport à l'arrière du buste à notre guise. A cette translation s'ajoute un autre déplacement sur l'axe Y de la variable hauteurJambesArrieres.
On va ensuite pouvoir commencer à dessiner le mollet et pour cela nous devons encore nous déplacer selon si c'est la jambe droite ou gauche sur le côté (axe X) de la moitié de la largeur du corps et de la moitié de la jambe arrière (afin qu'elle soit bine collée au buste). Après avoir effectué une rotation sur l'axe X de angleCuissesArriere, nous pouvons dessiner le mollet avec un cube dimensionné selon largeurJambesArrieres et longueurCuissesArrieres.
On va ensuite se déplacer du reste de la moitié de la longueur de la cuisse et tourner de angleMolletsArriere pour dessiner le mollet : un cube dimensionné selon largeurJambesArrieres et longueurMolletsArriere.
Nous pouvons maintenant finir cette jambe avec le pied, après une rotation de -angleMolletsArriere – angleCuisseArriere sur l'axe X avec un pivot au centre de la jambe pour se remettre droit (parallèle au sol) : un cube dimensionné selon largeurPiedsArrieres, hauteurPiedsArrieres et longueurPiedsArriere.
Les jambes avant sont dessinées en suivant le même procédé mais avec des variables différentes, ce qui nous permet directement de changer le sens des angles des rotations.

6. Les ailes

Les ailes sont composées de deux parties: l'armature et la partie souple. Nous commençons par translater en haut du corps à l'emplacement des ailes (la moitié de la hauteur du corps – un écart définis dans la variable decalageHauteurAiles). Pour chaque aile, nous allons réaliser avec un polygone la partie souple selon des proportions par rapport aux variables définissant l'envergure des ailes : longueurAiles et largeurAiles, sans oublier la normale qui pointe vers le ciel (1 sur l'axe Y).

Attention, les points du polygone ne sont pas dans le même sens pour les deux ailes afin de respecter le sens anti-horaire, définissant l'orientation de la face.

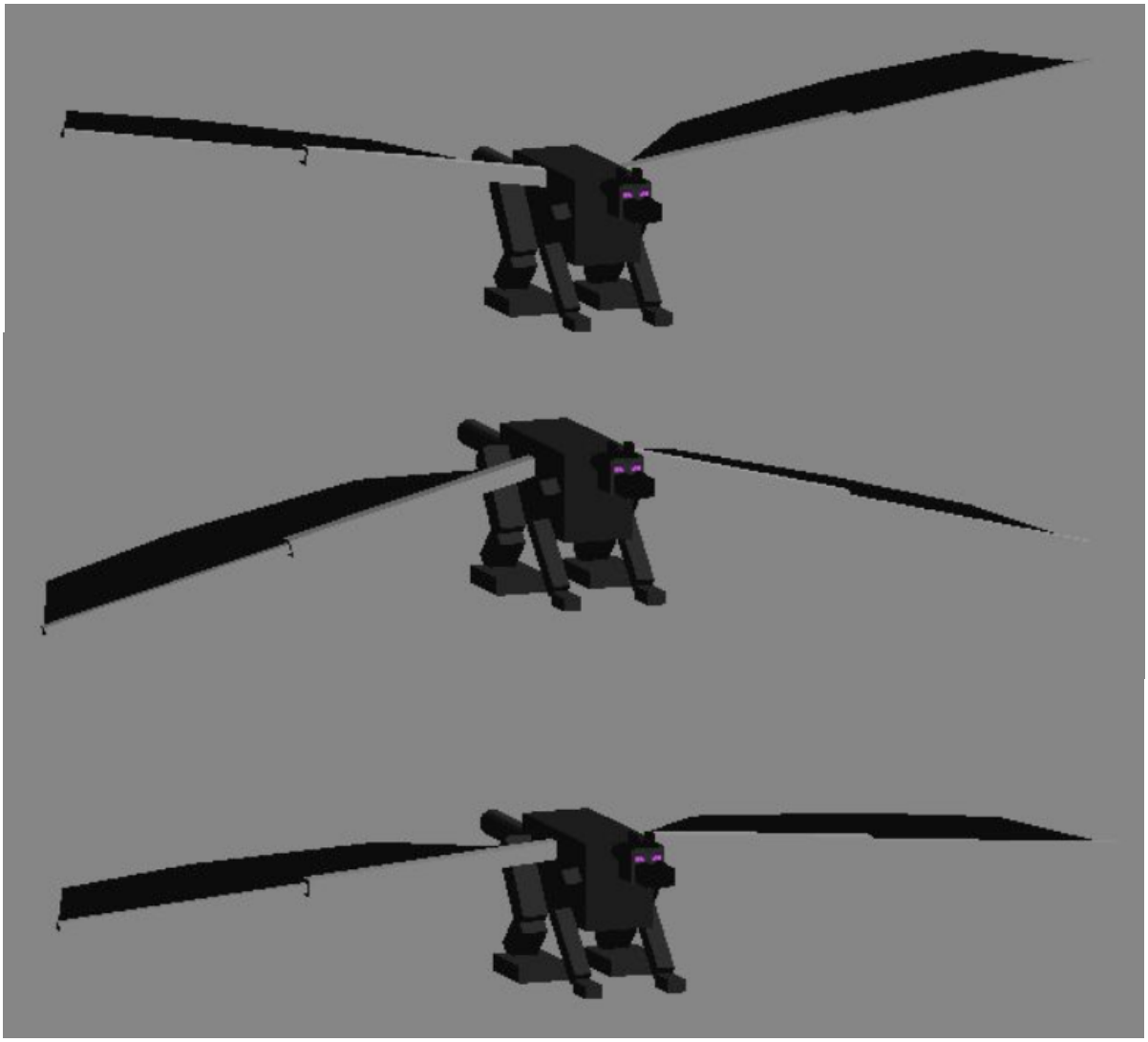
L'armature est quant à elle composée de 2 cubes, l'un plus fin que l'autre. On commence par translater du quart de la longueur des ailes, de dessiner un cube de dimensions la moitié de la largeur des ailes comme longueur. On va ensuite se déplacer encore sur l'axe X d'une unité (puisqu'un scale a été fais pour avoir la moitié de la taille des ailes sur l'axe X) et on va dessiner un autre cube « en enfant » du premier, de même longueur mais de la moitié de sa largeur.

II. Les animations

Le dragon n'est pas figé et est doté de deux animations différentes : le battement des ailes et l'envoi d'une boule de feu.

1. Le battement des ailes

Le battement des ailes est une animation automatique qui se joue en permanence et en boucle. Il s'agit simplement de la rotation des ailes, donnant au dragon une petite impression de mouvement.



Battement des ailes du dragon

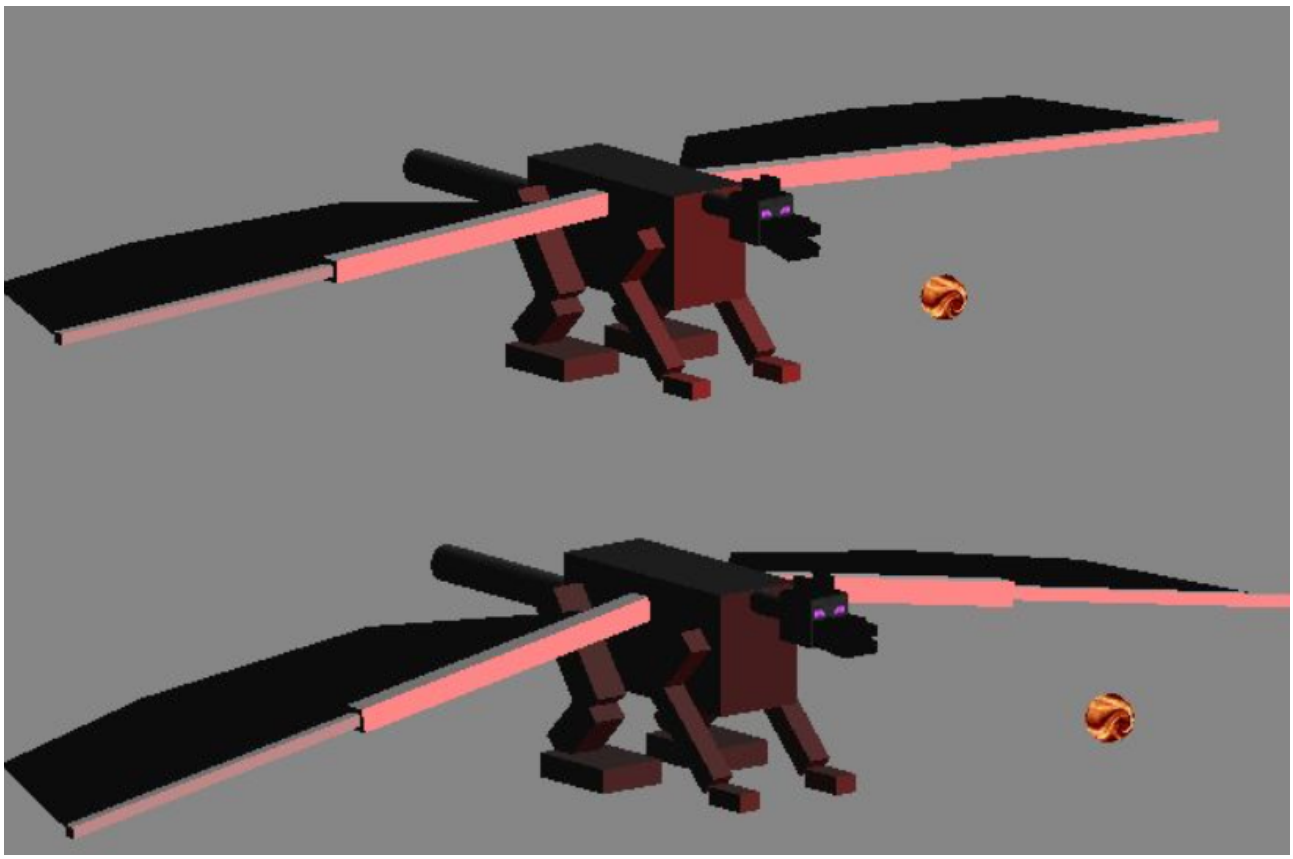
Pour réaliser cette animation, nous utilisons une fonction « anim » et quelques variables : ailesAngle qui contrôle le degré d'inclinaison des ailes, ailesSpeed qui détermine à quelle vitesse l'angle va varier (battement rapide ou lent), minAilesAngle et maxAilesAngle qui ont pour rôle de poser des limites à l'angle des ailes (on ne fais pas un tour entier, seulement dans cet intervalle).

La fonction anim est appelée fréquement (glutIdleFunc) et dans cette fonction nous allons ajouter à l'angle de rotation des ailes la variable de la vitesse de rotation : ailesAngle += ailesSpeed. Nous allons ensuite tester si l'angle est inférieur à l'angle minimum autorisé ou supérieur au maximum et si c'est le cas, on va juste inverser le signe de la vitesse de rotation, en multipliant par -1, afin que le prochain ajout à la rotation soit dans l'autre sens ce qui aura pour effet de faire battre les ailes dans l'autre sens.

Le dessin va ensuite être redessiné grâce à l'appelle à glutPostRedisplay à la fin de la fonction d'animation et lors du dessin des ailes, une rotation est effectuée sur l'axe X selon notre variable de rotation ailesAngle qui est modifié dynamiquement par la fonction d'animation et ainsi de suite.

2. L'envoi d'une boule de feu

Puisqu'un dragon est généralement synonyme de feu, nous avons décider d'ajouter une animation de lancer de boule de feu.



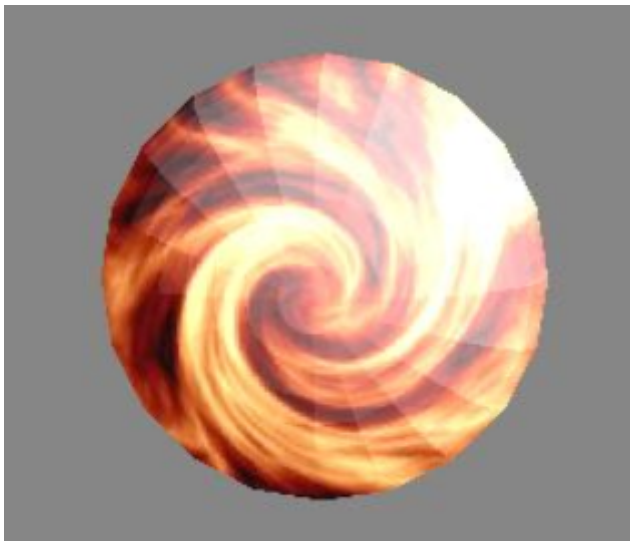
Lancer d’une boule de feu

A l’inverse du battement des ailes, cette animation n’est pas automatique ni en boucle alors nous avons du utiliser comme repère le temps, que nous récupérons grâce à glutGet(GLUT_ELAPSED_TIME) qui renvoie le nombre de millisecondes écoulées depuis l’initialisation de la scène.
Lorsque l’utilisateur appui sur la touche ‘b’ nous déclenchons l’animation via une fonction startAnimFireBall qui va modifier la valeur de quelques variables : animFireBall, booléen indiquant si l’animation se déroule ou non, passe à vrai, spawnFireBall, booléen indiquant si la sphère de la boule de feu doit être dessinée ou non, avancementFireBall, nombre indiquant la distance parcouru par la boule, passe à 0 et startTimeAnimFireBall, nombre stockant le temps à laquelle l’animation à débutée passe au temps actuel.

Lorsque la fonction « anim » est appelée nous allons, en plus d’effectuer l’animation automatique du battement des ailes comme décris précédement, tester si nous sommes aussi dans l’animation de lancer de boule de feu grâce à la valeur du booléen animFireBall. Si c’est le cas, nous allons déterminer à quel moment de l’animation nous nous situons grâce au temps : en soustrayant au temps actuel le temps du début de l’animation, nous obtenons le temps écoulé dans l’animation. Pour pouvoir changer la durée complète de l’animation beaucoup plus facilement, nous avons choisi d’utiliser une variable animFireBallDuration qui détermine la durée totale et avec laquelle nous allons diviser le temps écoulé depuis le début, puis encore diviser par 1000 car il s’agit de millisecondes, pour obtenir un temps écoulé normalisé entre 0 et 1 de l’avancement de l’animation.

L’animation est séparée en 3 étapes pouvant se chevaucher : le dragon ouvre la bouche, crache la boule de feu (elle se déplace) et le dragon referme la bouche.
Durant le premier tiers (temps normalisé inférieur à 0,33), on ouvre la bouche en augmentant l’angle contrôlant la rotation de la machoire inférieure, comme pour la rotation des ailes, mais cette fois non pas avec une vitesse mais en fonction de l’avancée de l’animation. En effet, l’angle d’ouverture est égal à un angle maximal défini par la variable mouthMaxAngleFireBall, multiplié par le temps normalisé de l’animation, le tout multiplié par 3 de sorte à ce que l’angle soit égal au maximum lorsque l’animation est arrivée au tiers de sa durée.

Ensuite, le dragon doit cracher sa boule de feu, ce qu’on indique en passant à vrai le booléen spawnFireBall si le temps normalisé est supérieur à 0,33. La boule doit alors se déplacer, ce qu’on réalise en augmentant la variable avancementFireBall grâce à une autre variable représentant sa vitesse : speedFireBall. Lorsque le dragon va être redessiné, spawnFireBall va indiquer qu’il faut ajouter une sphère lors du dessin de la tête. Cette sphère sera translatée sur l’axe Z de avancementFireBall pour simuler l’envoi de la boule. Attention, cette sphère est généré algorithmiquement par une classe spéciale, tout comme les cylindres de la queue et du cou. Nous avons utilisé le même procédé : génération des points selon la représentation paramétrique, reliure des points consécutifs pour former les faces et calcul des normales grâce au produit scalaire. Nous enroulons également une texture autour de cette sphère, en étalant linéairement la texture le long des deux axes selon l’équation de 2 droites par rapport aux coordonnées des sommets.
En effet, on colle aux points dans la coordonée sur l’axe X sont aux extrémités du cercle centrale (donc égal au rayon) les extrémités de la texture, ceux qui sont au centre le centre de la texture et les points entrent ont des coordonnées de texture proportionnelles, ce qui donne une équation de droite et nous procédons de façon similaire pour l’axe Y.



La sphère de la boule de feu lancée par le dragon

En plus de déplacer la boule de feu après le premier tiers, il faut, avant le deuxième tiers, avoir refermer la bouche du dragon. Pour cela, nous procédons de la même façon que pour l’ouvrir : avec une équation définissant la variable contrôlant la rotation pour l’ouverture de la bouche lors de la phase de dessin de l’objet. Cependant, cette équation est bien différente car au tier l’angle doit être le maximum pour coller à son ouverture juste avant et au deuxième tiers être à 0. Nous allons donc prendre les 2/3 de l’angle maximum auquel nous soustrayons ce même maximum multiplié par le temps normalisé, le tout multiplié par 3 car tout s’effectue en un tiers de la durée.
Le dernier tiers est consacré uniquement au déplacement de la boule de feu.

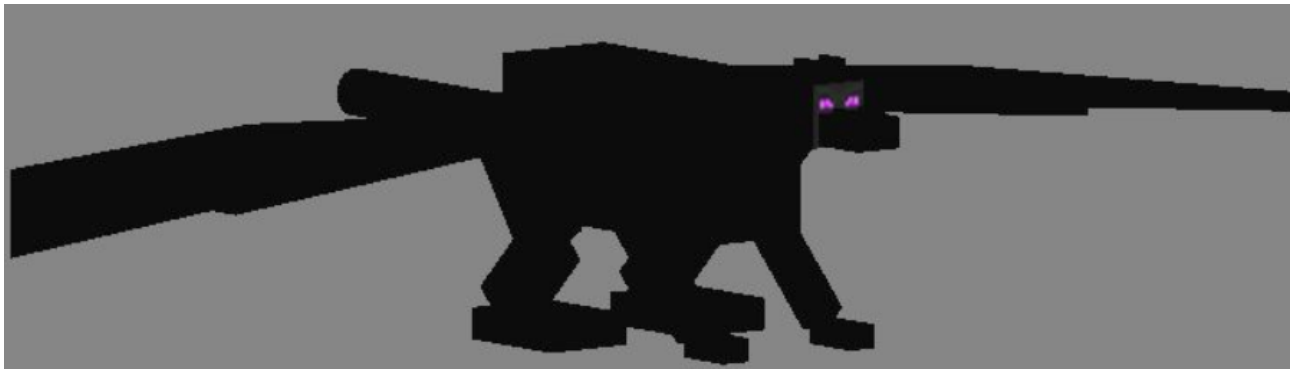
Pour finir, nous allons tester si le temps normalisé est supérieur à 1 et si c'est le cas c'est que l'animation est terminée et nous allons l'arrêter en passant le booléen animFireBall à faux et en désactivant la source lumineuse de la boule de feu.

III. Les lumières

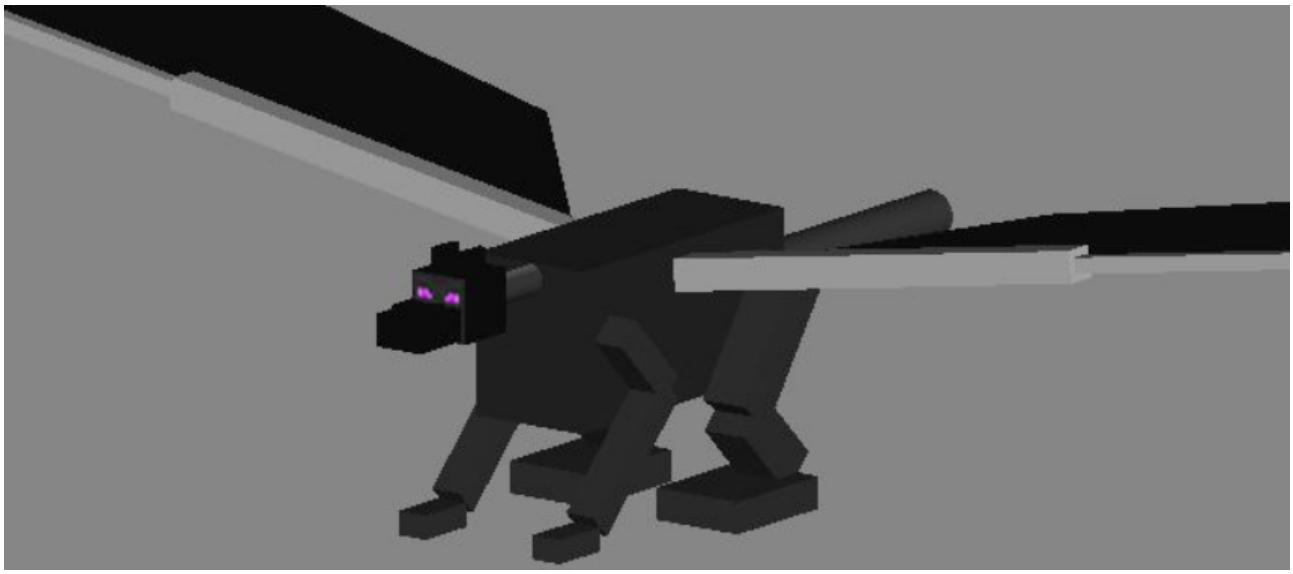
Pour qu'il soit possible d'admirer le dragon de manière plus réaliste, nous avons ajouté deux sources lumineuses dans la scène : le soleil et la boule de feu de l'animation.

1. Le soleil

Le rôle du soleil est d'illuminer toute la scène, ce que nous réalisons en utilisant la GL_LIGHT0 comme éclairage diffusant une lumière blanche (SPECULAR et DIFFUSE). Elle nous permet d'obtenir un rendu plus réaliste lors du déplacement autour du dragon et sans elle, nous ne pouvons pas distinguer les variations de couleur sur la créature, qui sont définies non pas avec glColor mais avec glMaterial.



Le dragon sans lumière



Le dragon avec lumière, côté exposé au soleil

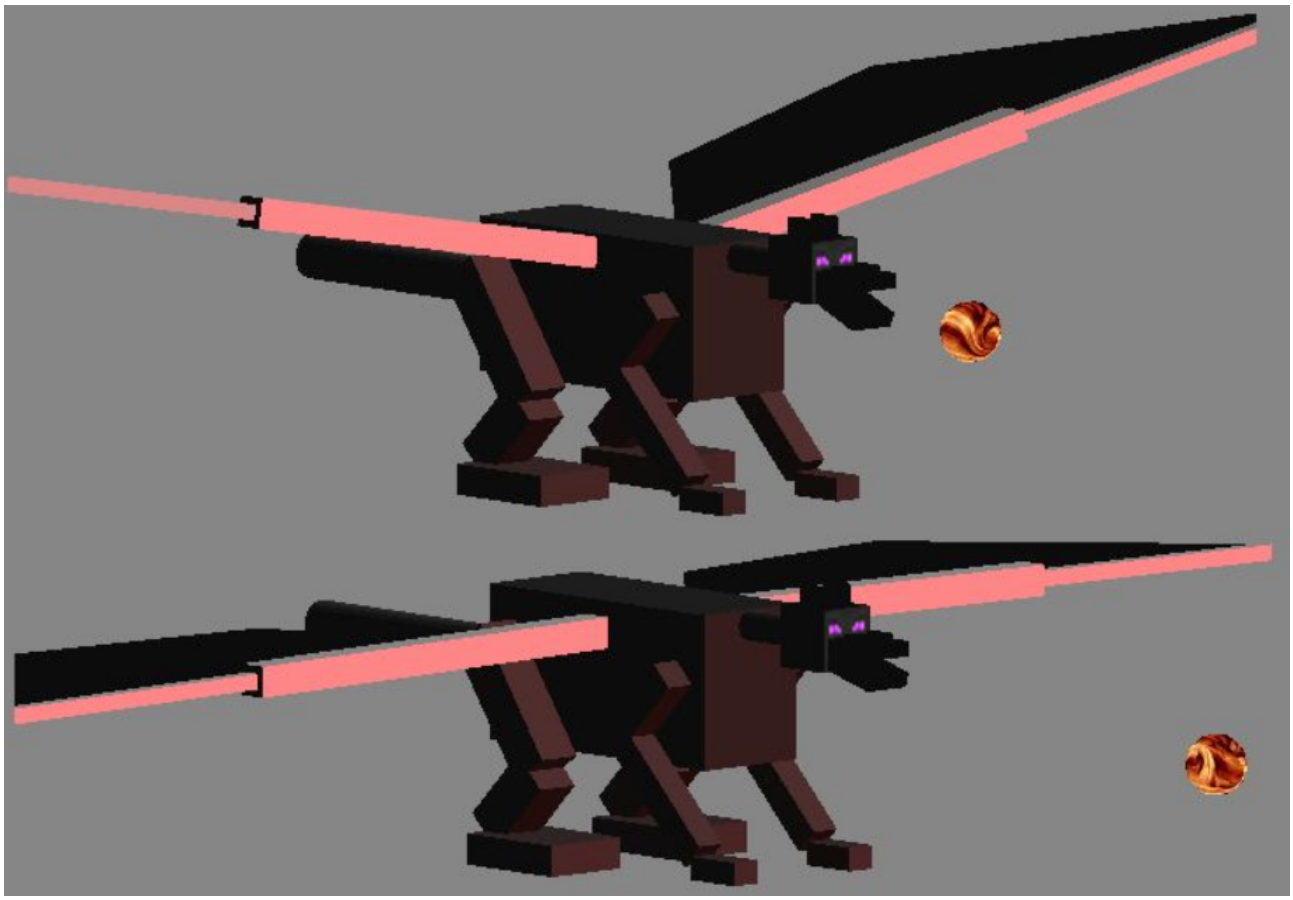
Le dragon avec lumière, côté caché de la lumière

2. La boule de feu

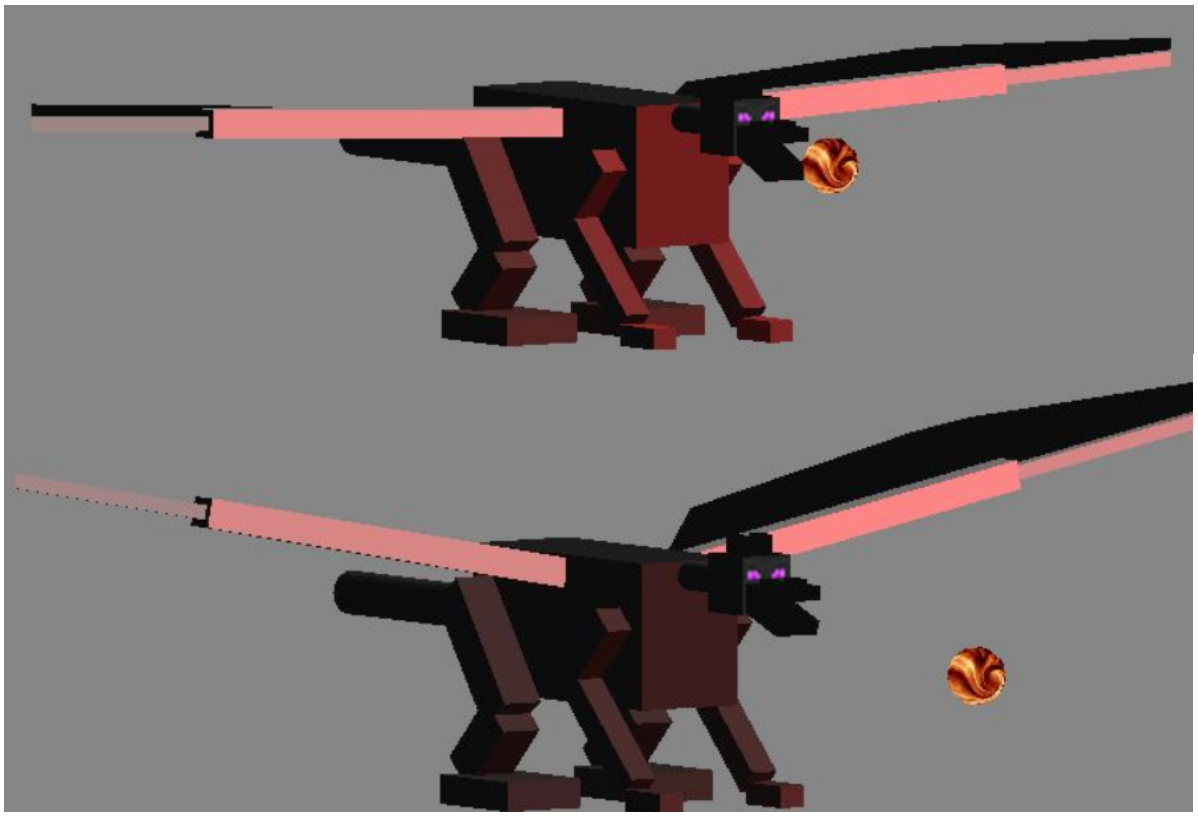
Afin de rendre plus réaliste l'animation du lancer de la boule de feu, nous avons ajouté une source lumineuse qui suit la sphère représentant la boule. La boule éclaire dans toutes les directions d'une lumière rouge (propriété DIFFUSE) pour représenter le feu qui brulerait et a pour effet d'illuminer la bouche, les ailes et le buste du dragon d'une couleur rougeâtre.
Pour la séparer du soleil, nous utilisons la GL_LIGHT1 et sa configuration est un peu particulière : elle prend en compte une atténuation de la lumière en fonction de la distance, ce qui donne un effet réduisant l'illumination rougeâtre projetée par la boule de feu à mesure qu'elle s'éloigne de son lanceur.

Pour réaliser cet effet, nous utilisons les paramètres GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION et GL_QUADRATIC_ATTENUATION dont les valeurs sont mises à respectivement 0, 0.75 et 1.
Dans le cas de lumières, rien de mieux que des images pour illustrer nos propos alors nous allons voir l'effet de la lumière telle que nous l'avons paramétrée, en comparant sans lumière et sans atténuation.

Lancer de la boule de feu sans lumière



Lancer de la boule de feu avec lumière mais sans atténuation



Lancer de la boule de feu avec lumière et atténuation