Kill-mob

Projet d’algèbre géométrique

Rapport

HARDY Elise

Master 2 Informatique

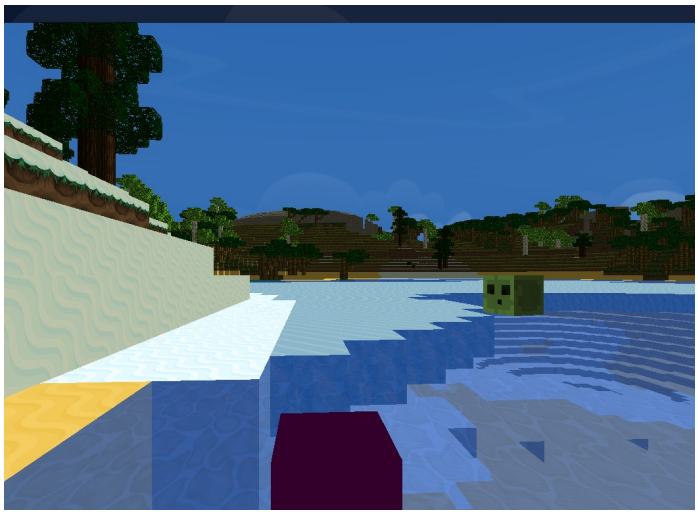
2019 / 2020

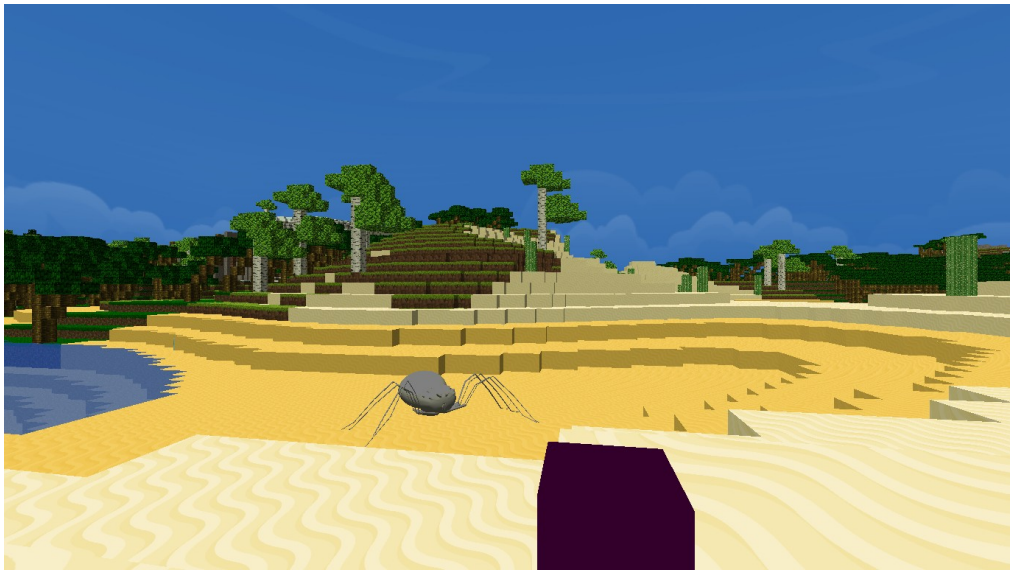
Table of Contents

I. Introduction

Pour le cours de vision par ordinateur algèbre géometrique nous étions libre de choisir le sujet qui nous plaisait. Celui-ci devait néanmoins comporter des notions d'algèbre géométrique.

J'ai choisis de développer un mini-jeux de tir en 3D grâce à la bibliothèque Garamon.





II. Synopsie du jeu

Vous venez de débuter en tant qu’aventurier et on vous a confié comme première mission de débarasser la forêt de Gardonne de ses monstres. Pour cela vous êtes équipé d’un canon à particules (une boite violette magique) et d’une bombe énergétique. Si un monstre vous touche vous perdez des points de vie, si votre vie tombe à zéro vous mourrez.

Attention la forêt de Gardonne est immense. Ne vous perdez pas et bonne chance.

III. Architecture

Pour la création du jeu, j'ai choisis d'utiliser la librairie openGL (elle permet de modéliser en 3D de nombreux objets) et Garamon pour l'algèbre géométrique (configuration c3ga)

J'ai préféré choisir l'algèbre géometrique conforme car je l'a trouve plus simple et que nous avions déjà travaillé dessus en TP.

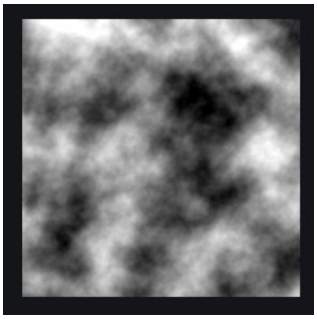
1. Map

La carte du jeu est composée de SuperChunk composé de Chunk eux même composé de cube.

Cette structure permet de pouvoir modifier un grand nombre de cube d'un coup sans devoir tous les parcourir. Nous avons alors une classe ChunkManager qui s'occupe de créer mais aussi de détecter les changements dans la map. La classe s'occupe aussi du rendu des cubes et d'initialiser les shaders.

Pour accélérer le chargement du jeu et les calculs, les cubes non visible, en dehors du champ de la caméra ne sont pas affichés ainsi que les cubes qui se trouvant trop loin.

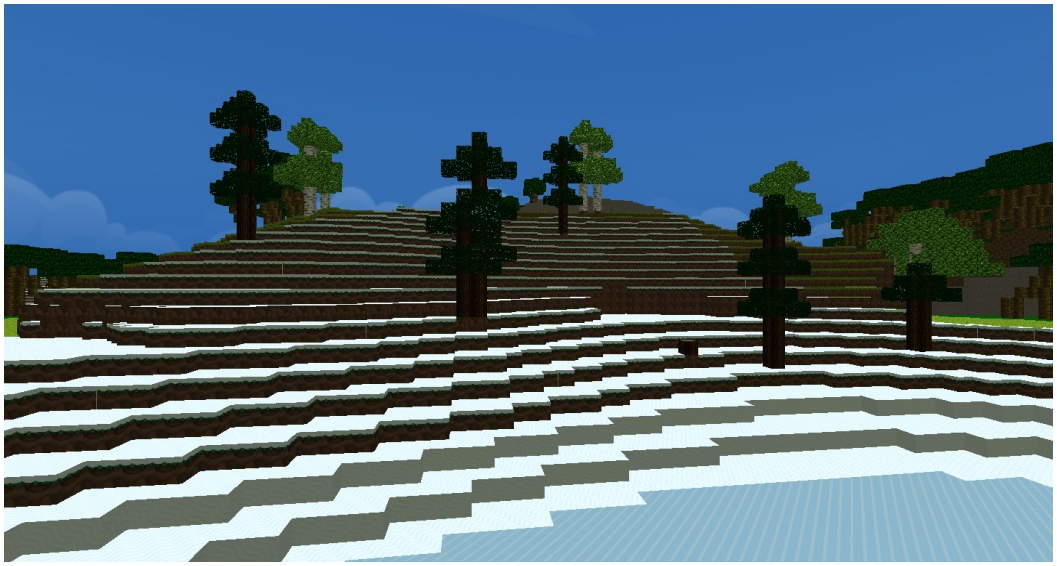
Pour chaque superChunk un bruit de perlin 2D (Le bruit de Perlin est souvent utilisé en image pour créer des effets réalismes dans les textures procédurales, il est speudo-aléatoires) est généré pour construire l'élévation du terrain. Un autre bruit est créé pour modéliser des températures. Le bruit de Perlin est aléatoire donc différent à chaque chargement du jeu mais reste le même pour une position lorsqu'on est dans le jeu. Ainsi si on avance la caméra jusqu'au prochain chunk et que l'on revient en arrière on retrouvera le même terrain à l'identique.



De plus nous avons soustrait un autre bruit au bruit de l'élévation pour créer des falaises car le bruit de Perlin est trop lisse et ne permet pas de changement brute dans l'altitude.

Lors de la construction du terrain, les textures des cubes correspondent à une température et une altitude (la neige tout en haut et une température basse, le sable en bas …). Ce qui permet de créer des micro-climats et de donner à plusieurs textures la même élévation sans forcément se retrouver avec une carte incohérente.

Ensuite on place les élements du décors (arbres et monstres). Les élements du décors sont placés selon des probabilités d'apparition (les sapins sont plus rare que les arbres de la jungle par exemple, un slime à plus de chance d'apparaitre qu'une araignée). Les arbres eux doivent aussi correspondre à un certain climat (les sapins dans la neige, les cactus dans le désert,...).



2. Monstre

Les monstres sont crées à partir d'un fichier obj qui décrient la position de leurs points, leurs normals ainsi que les textures. Ce fichier est organisé selon un format spécial. Pour ce faire un parseur a été créé pour lire ce type de fichier. Cela permet d'ajouter facilement et rapidement d'autres types de monstres dans la carte et de ne pas s'embéter à créer à la main les points du

monstre. Pour ajouter un monstre il suffit d'avoir le fichier obj que l'on trouve facilement dans les modélisateurs 3D et son image pour les textures.

Le déplacements des monstres se fait aléatoirement, on attribue une direction au hasard puis on fait avancer le monstre selon cette direction. On vérifie bien évidemment à chaque fois avant que le monstre peut avancer, qu'il n'y a pas d'obstacle ou que l'élévation n'est pas trop haute.

3. Aventurier

Le personnage se joue avec les touches directionelles (ou z,q, d, s), le personnage peut voir sur le côté grâce à la souris mais si celui-ci veut tirer il doit se trouver en face du canon à particules et appuier sur la touche t.

Pour utiliser la bombe énergétique le joueur doit appuyer sur la touche b. La bombe énérgétique n'est disponible qu'à partir d'un certain nombre de monstres tués et lorsque l'aventurier utilise la bombe celle-ci consomme les monstres tués.

Pour que le jeu soit plus rapide à tester une touche de triche à été ajouté : la touche c.

Elle permet de déplacer la caméra en hauteur (ctrl pour le bas et espace pour le haut) et de passer le jour ou la nuit avec la lettre e.

4. Managers

Le jeu est composé de nombreuses classes manager, chacun doit s'occuper d'une partie du fonctionnement du jeu. Ce qui permet une indépendalité entre chaque classe.

La classe InputManager permet ainsi de gérer exclusivement toutes les actions des touches et de la souris commes les déplacement du joueur, les tirs,…

Le WindowManager lui va s'occuper des créations de la fenêtre du jeu, de changer sa taille pour la mettre en pleine écran, de la rafraichir,…

La classe ConfigManager gère tous ce qui peut se configurer comme le nombre de frames, l'occlusion culling (n'affiche pas les objets qui se trouvent derrière un autre) mais aussi tous ce qui concerne les informations du debug et l'activation du cheat code.

Cette classe gère aussi la skybox, représenté par un cube tous autour de la scène, qui change selon la position du soleil, et la lumière du jeu qui augmente au diminue selon le moment de la journée.

IV. Algèbre géometrique

1. Monstres

La forêt de Gardonne regorge de monstres en tout genre, dont trois types différents.

- Les slimes

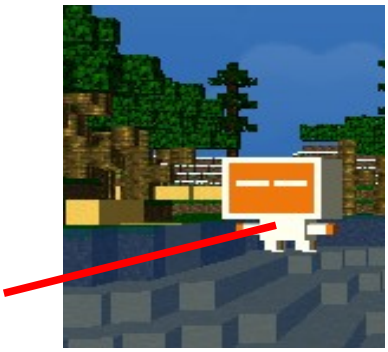
- les robots

- les araignés

Les monstres sont représentés en algèbre géometrique par une sphère duale de rayon variable selon la taille du monstre.

La position du monstre correspond à un point au centre de la sphère duale.

Les robots possèdent en plus des autres monstres un rayon laser (représenté par une ligne). Ce rayon laser suit la direction des robots.

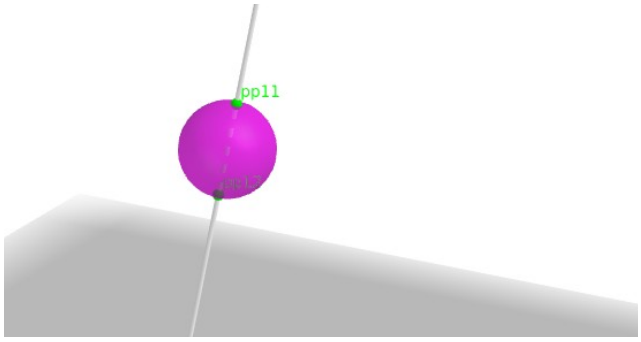


2. Aventurier

L'aventurier est lui aussi représenté par une sphère duale centrée en sa position.

A chaque boucle du jeu on vérifie si le joueur n'est pas entré en colision avec un monstre ou avec un tir de laser d'un robot.

Si l’aventurier est touché par le laser d’un robot, le jeu s’arrête de suite, tandis que si le joueur rentre dans un monstre il perd juste quelque points de vie. Lorsque sa vie est à zéro le jeu s’arrête.



3. Armes

L’aventurier possède deux types d’armes pour lutter contre les monstres.

- un canon à particules
- une bombe énergétique

Les armes commes les autres élements sont représentés en algèbre géométrique par une sphère duale centrée pour le canon sur le cube violet et pour la bombe sur la position de l’aventurier.

Lorsque l’aventurier tire avec son canon, on vérifie si la sphère du monstre et celle du canon s’intersectent.

Pour cela on utilise la formule suivante :

cercle_intersection = (sphereDualMonstrer ^ sphereDualArme).dual()

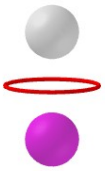
Ensuite on calcul le rayon du cercle : **rayon = sqrt(cercle_intersection * cercle_intersection)**

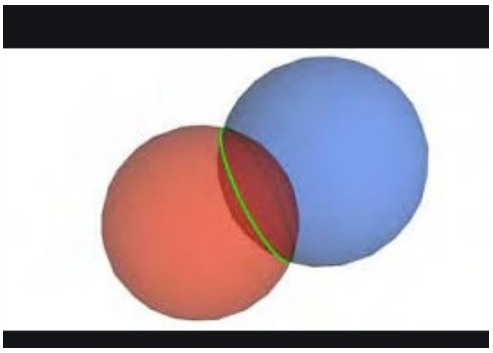
puis on calcul : **(rayon/ (rayon* E_inf)) * (rayon/ (rayon* E_inf))**

Tous les E_x corresponde au donnée de la métrique.

Ainsi **E_inf** correspond à l’infinie.

On vérifie alors que le résultat soit supérieur à zéro dans les réels. Si c’est le cas alors les deux sphères s’intersectent.





Cette vérification est répété sur la sphère du canon que l'on translate de 0,6 vers l'avant jusqu'à dix fois pour modéliser un tir.

Pour la translation nous utilisons la formule suivante :

T = (1,0 + (-1,0/2,0)*t * E_inf

t = vecteur de translation

que l'on applique sur l'objet : **objet = T * objet * T.inverse()**

Pour la bombe en revanche la sphère ne translate pas mais est dilatée.

On dilate la sphère pour l'agrandir d'un facteur de 0,5.

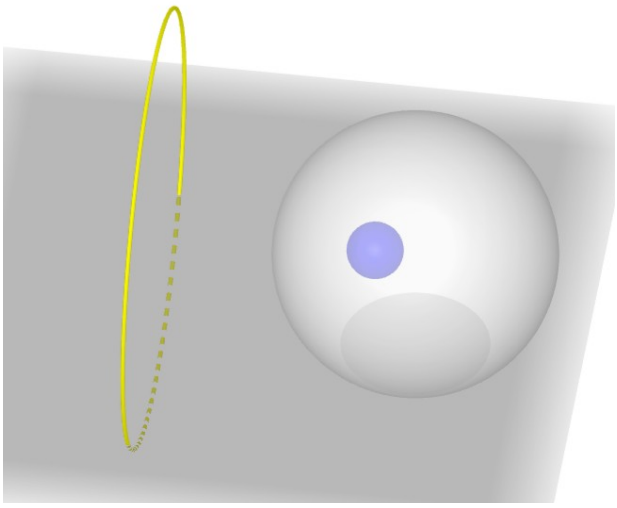
On utilise comme formule:

D = (1- (1,0 -s / (1,0 + s) * (-E_0inf)

s = facteur d'agrandissement

puis on applique la dilatation sur l'objet : **objet = D * objet * D.inverse()**

Pour la bombe j'ai du ajouter une condition d'intersection car j'ai remarqué que lorsqu'une sphère est contenue dans une autre alors l'intersection des deux est toujours dans les imaginaires.



Donc on ajoute la condition suivante :

on vérifie aussi que la position du centre de la sphère du monstre est à l'interieur du rayon de la sphère de la bombe.

4. Tool

Pour utiliser la librairie Garamon, j'ai du redéfinir des outils pour créer les objets plus facilement comme lors du TP.

Une fonction pour créer un point, une sphère duale, les translations, dilateur mais aussi une fonction pour calculer le centre et le rayon d'une sphère duale.

Fonction pour un point :

```
template<typename T>
c3ga::Mvec<T> point(const T &x, const T &y, const T &z){

   c3ga::Mvec<T> mv;
   mv[c3ga::E1] = x;
   mv[c3ga::E2] = y;
   mv[c3ga::E3] = z;
   mv[c3ga::Ei] = 0.5 * (x*x + y*y + z*z);
   mv[c3ga::E0] = 1.0;

   return mv;
}
```

Fonction pour une sphère duale :

```
template<typename T>
c3ga::Mvec<T> dualSphere(const T &centerX, const T &centerY, const T &centerZ,
const T &radius){
   c3ga::Mvec<T> dualSphere = point(centerX,centerY,centerZ);
   dualSphere[c3ga::Ei] -= 0.5*radius;
   return dualSphere;
}
```

Fonction pour le centre et le rayon d'une sphère duale :

```
template<typename T>
void radiusAndCenterFromDualSphere(const c3ga::Mvec<T> &dualSphere, T
&radius, c3ga::Mvec<T> &center){
   radius = (dualSphere | dualSphere) / dualSphere[c3ga::E0];
   center = dualSphere / dualSphere[c3ga::E0];
}
```

L'utilisation des templates dans le code permet ainsi de les utiliser avec n'importe quel type de données et on évite alors de redéfinir les fonctions.

III. Conclusion

La possibilité de choisir le sujet du projet par nous même, nous a permis de travailler sur un projet qui nous intérressait vraiment. J’ai ainsi choisis un jeu car à la fin on a un visuel et que c’est intéressent à développer. De plus nous avons pu mettre en pratique les notions abordés lors du cours d’algèbre géométrique et lors du TP.

Notamment, les translations, scales, intersections mais aussi utiliser le dual. J’ai ainsi pu voir les limites de la géometrie algébrique comme pour l’intersection de deux sphères avec l’un contenue dans l’autre.

Ce projet nous a aussi permit d’utiliser la librairie Garamon développé par l’universitée.