

Rapport projet C++ et synthèse d'images

Boids et scène en OpenGL

I. Fonctionnalités du projet

Fonctionnalité	Fonctionnelle	Quasi-fonctionnelle	Non fonctionnelle
Quad	X		
Multi Lights	X		
Multi Shadow		X	
Limites	X		
Carapateur	X		
Collision carapateur	X		
Objets simples	X		
Objets texturés	X		
Modèles 3D (couleurs)	X		
Modèles 3D (textures)			X
Animations *		X	
Caméra 3e personne		X	
Cycle jour/nuit *	X		
Ombre lumière directionnelle	X		
Ombre lumière ponctuelle		X	
Boids	X		
Séparation/alignement/cohésion	X		
ImGui	X		
LODs	X		
Particules *	X		
Manipulation du temps *	X		

* Bonus

Lien du git : https://github.com/MissBidule/OpenGL_TPs_S4

II. Idées initiales du projet

Pour ce projet nous n'avons pas trop eu le temps de nous concerter. Benjamin étant malade c'est Lilou qui a décidé de la direction artistique du projet.

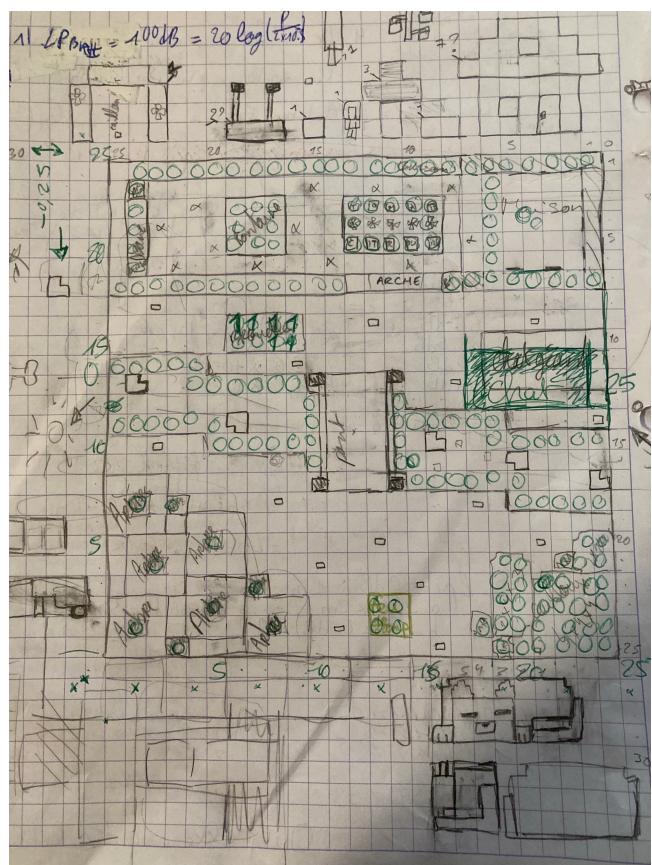
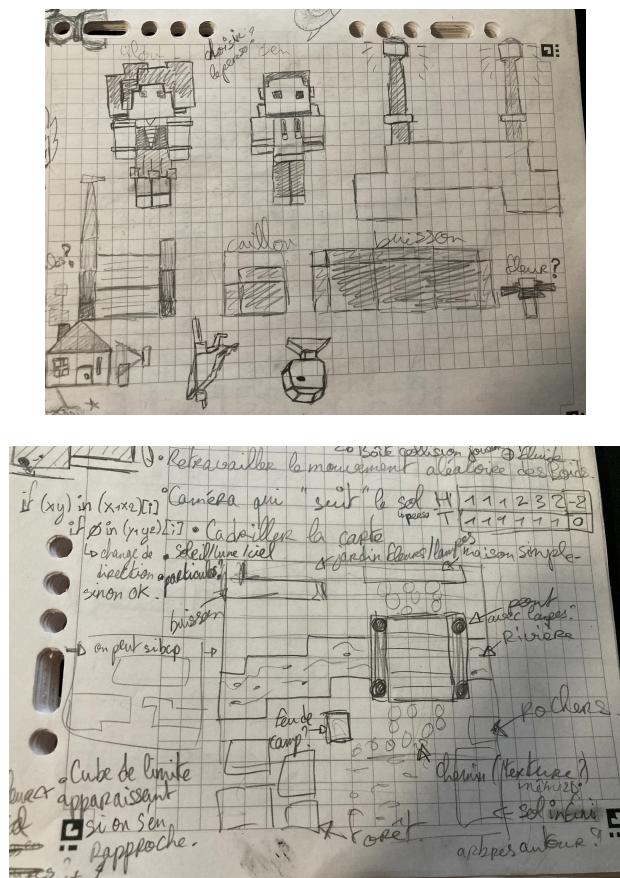
Nous sommes donc partis sur quelque chose de très cubique et accessible côté modélisation. Et pour la cohérence de ses formes simples, nous avons décidé de garder des couleurs et non pas des matières pour habiller les objets.

Les limites sont inspirées de certains battle royal, notamment dans des serveurs Minecraft : des limites qui n'apparaissent en transparence que si le joueur s'en rapproche.

Une rivière et le ciel pour les boids, et un décors en forêt. Le feu de camp est une excuse pour utiliser des particules, et quelques idées farfelues pour habiller le décors comme une brouette ou le chat géant qui dort.

Initialement il y avait 7 lumières de prévues, 4 sur le pont, le soleil, la lune et la lanterne, mais des difficultés techniques nous ont fait changer de direction.

L'idée de changer de personnage pour représenter notre binôme et de manipuler le temps qui passe donne une autre dimension de gestion au projet ; on gère un mini-monde.



III. Pistes de recherches et essais

Pour démarrer le projet il a fallu faire plusieurs tests et recherches avant de commencer la scène en elle-même.

Après son rétablissement Benjamin a repris les Boids du début tandis que Lilou a expérimenté des fonctionnalités avec OpenGL et P6.

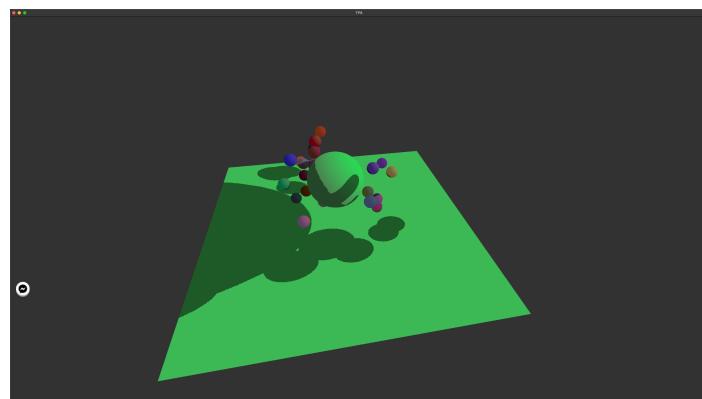
A. OPENGL

Après la réalisation des TPs, nous avons cherché à ajouter les ombres et ce en suivant un tutoriel qui utilise la caméra au point de vue de la lumière pour enregistrer sur une texture les éléments affichés et vus par la lumière. Ce faisant, tous les éléments qui sont cachés par d'autres du point de vue de la lumière se retrouvent dans l'ombre.

Pour la lumière directionnelle l'ombre est écrite depuis un point de vue orthographique. Pour la lumière « spot » et ponctuelle, cela se fait depuis un point de vue perspective.

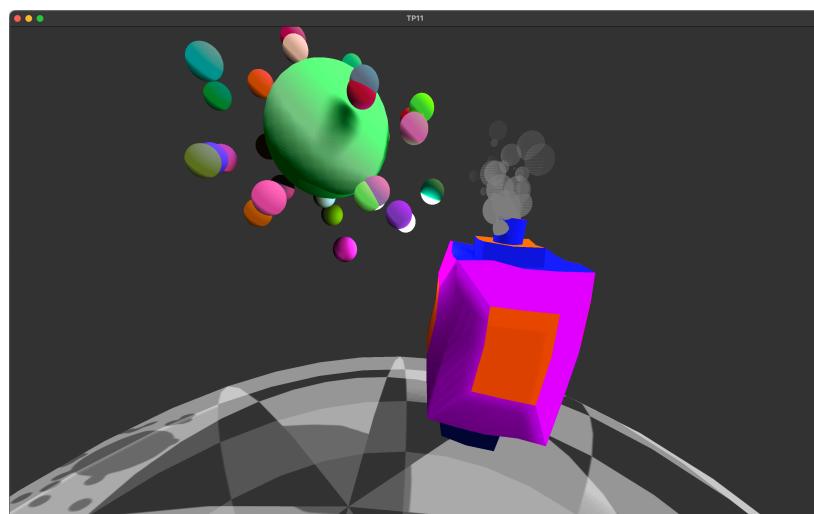
Ensuite, pour la matrice de vue de la lumière, la FreeFlyCamera allait très bien avec la lumière « spot » et directionnelle tandis que la Trackball permet une prise en 6 étapes pour la lumière ponctuelle afin de prendre tous les angles qui entourent cette lumière.

Les Ombres de la lumière directionnelle ont été améliorées d'un flou pour qu'elle soit moins « dure ».

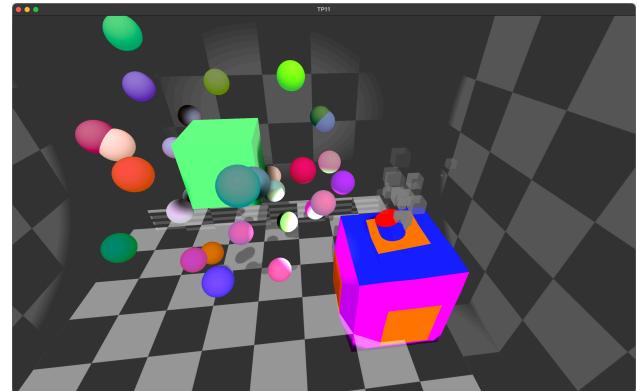
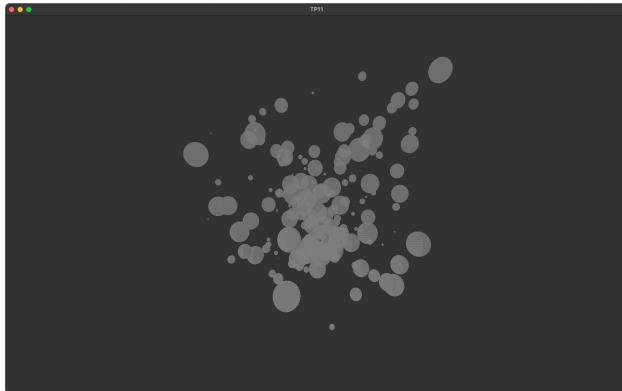


Pour les objets 3Ds, nous avons implémentés la bibliothèque Assimp qui après des difficultés liés à la complexité de la librairie, nous a permis de mettre un objet 3D (d'abord en obj et mtl puis en FBX) dans la scène, puis de l'animer grâce à son armature. Étrangement les objets 3Ds de Blender à OpenGL n'avaient presque jamais la même orientation ou taille de l'un à l'autre. N'ayant pas réussi à résoudre ce problème, j'ai dû adapter mes matrices modèles en conséquence.

Nous avons essayé de sortir les couleurs de Blender sous forme de texture mais n'avons pas vraiment compris comment s'y prendre. De plus Blender n'a pas de couleur ambiante, il a fallu composer avec la lumière diffuse.

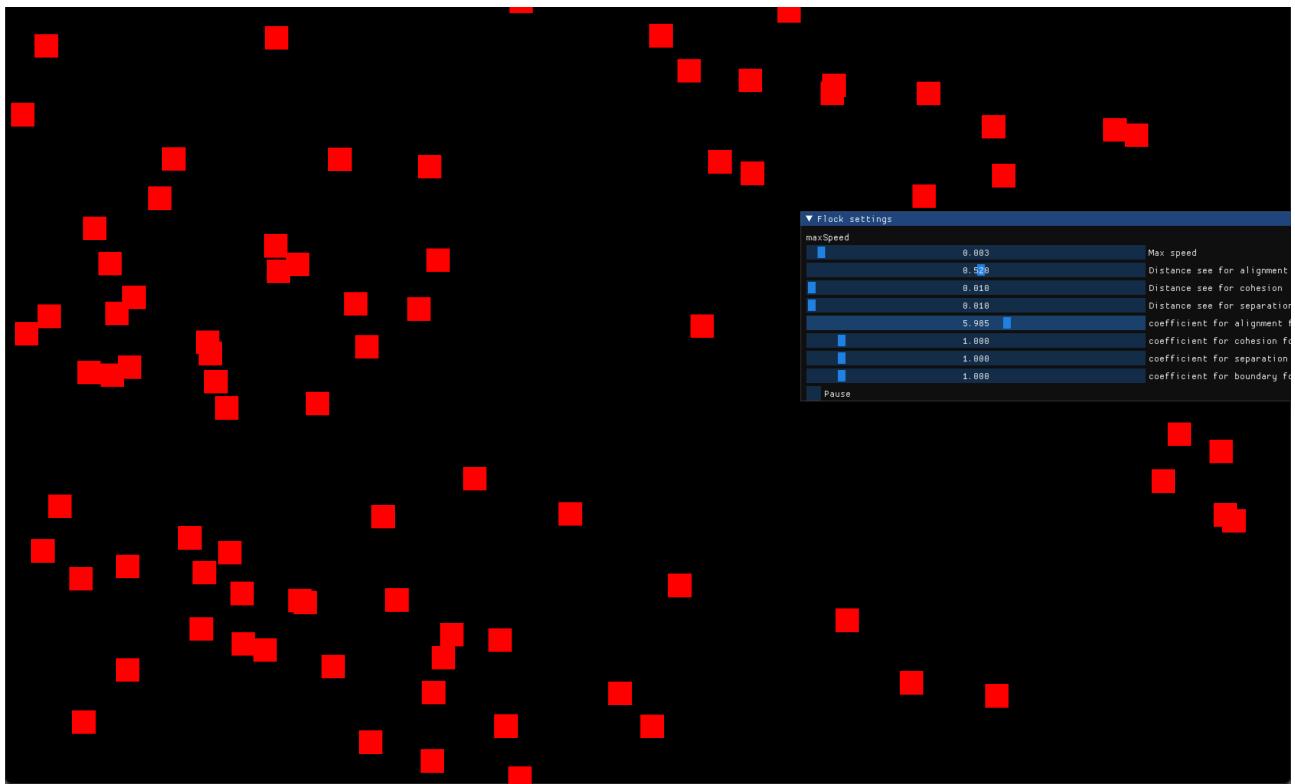


Suite à ça nous avons implémentés les objets texturés simples et les limites sur le principe de la distance et en reprenant un code fait par Lilou en DUT info, nous avons réintégré les particules.



Après un bon nettoyage du code, la scène a pu être implémenté.

B. LES BOIDS

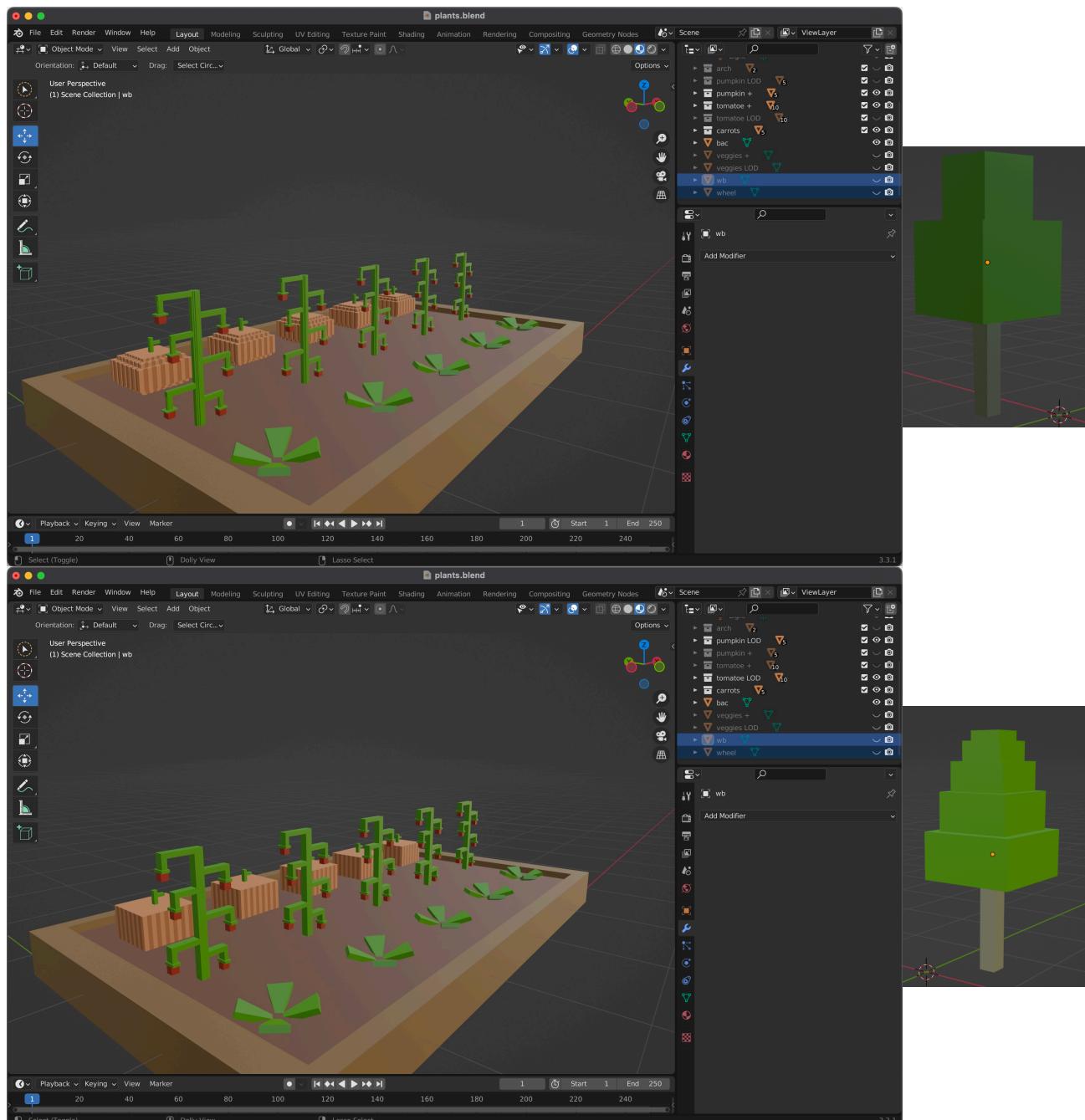


Pour les Boids, j'ai tout d'abord regardé le code qu'avait fait Lilou en classe, étant absent pendant cette période, puis j'ai décidé de regarder les ressources en lignes, je me suis notamment inspiré de la vidéo YouTube code adventure sur les boids. J'ai donc d'abord commencé en essayant d'implémenter les boids à l'aide d'une valeur vitesse et d'une valeur angle, afin d'implémenter les trois règles, mais je me suis vite rendu compte que ce ne serait pas optimale. J'ai donc opté pour un calcul de vecteur de direction que le boïd devrait emprunter pour chaque règle, avant de les

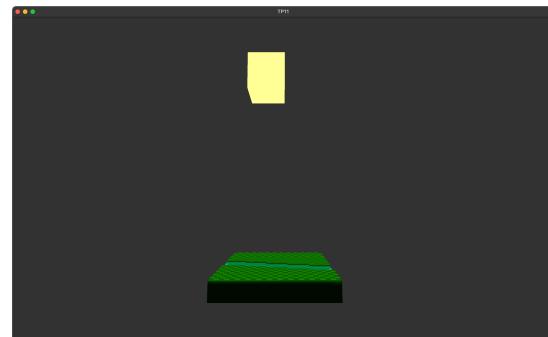
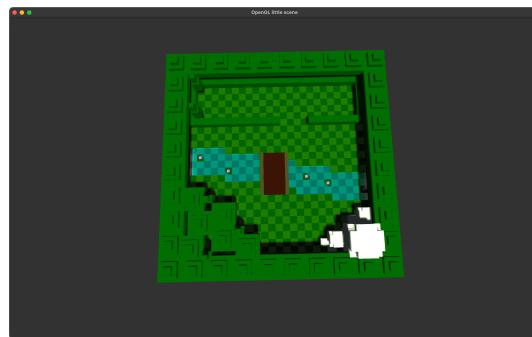
ajouter, chacun avec un coefficient que l'on peut manipuler avec imgui, afin qu'il soit possible de régler la simulation. Exemple si on veut qu'il subisse plus la force d'alignement, la force de cohésion etc... J'ai aussi mis une valeur de distance minimum dans laquelle les boids qui entourent le boïd actuel doivent être, afin qu'il rentre dans le calcul de la règle. J'ai aussi ajouté une règle de bordure afin que les boids ne sortent pas de leur zone délimitée, ainsi qu'une fonction qui limite la vitesse maximum des boids. Dans notre simulation nous avons deux types de Boïds les poissons dans la rivière, et les oiseaux dans le ciel.

IV. Modélisations 3D

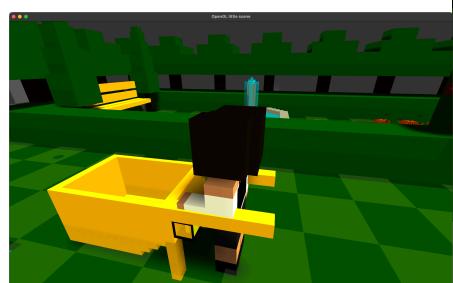
Blender a été le logiciel principal de toutes les modélisations faites sur ce projet. Différents niveaux d'objets ont été créer pour quelques objets qui à partir d'une certaine distance (par défaut) sont changés par leur version simplifiés.



Grâce au plan papier, il a été facile de placer les éléments dans la scène en faisant une translation vers la position désirée. Tous les objets sont des créations personnelles inspirées par la simplicité de modélisation de jeu cubique comme Minecraft.



L'eau et le chat qui dort bougent pour donner un peu de vie à la scène et la cheminée et le feu de camp font des particules de fumées.



Liste d'objets	LODs
Sol + eau	Non
Boîte limites	Non
Arbres (grand)	Non
Arbres (fin)	Oui

Liste d'objets	LODs
Cailloux (gros)	Oui
Trèfles	Oui
Nénuphars	Oui
Pont	Non
Chat géant	Non
Brouette	Non
Poissons	Non
Oiseaux	Non
Maison	Non
Arche	Non
Buissons	Non
Banc	Non
Fontaine	Oui
Fleurs	Oui
Potager	Oui
Feu de camp	Non

V. Structure du projet

Le projet est séparé en 3 morceaux : la scène, les shaders, et les classes.

h++ App.hpp
C++ AppInit.cpp
C++ AppUpdate.cpp
C++ main.cpp

La scène est appelée dans le main via les 3 fichiers App. AppInit s'occupe de l'initialisation des objets et éléments de la scène (tout ce qui n'est lancé qu'une fois) et AppUpdate s'occupe de tout ce qui est rafraîchi à chaque frame de l'application. Le tout est encapsulé dans la classe App.



Dans les shaders, tous les calculs de placement et couleurs sont faits en fonctions de ce que l'on cherche à faire.

Le vertex shader 3D s'occupe des objets simples, 3Dbones des objets animés, particle va gérer les particules de manière simplifiée, et shadiw sert à écrire les données des ombres au bon endroit.

Les fragment shader dirPoslight servent à gérer l'incidence des lumières sur une scène avec et sans texture, limit va calculer et n'afficher les rebords des limites qu'à partir d'une certaine distance, particle fait la même chose que son vertex shader et enfin les deux shadow vont gérer l'écriture de la texture d'ombre différemment selon la lumière.

Les Classes sont beaucoup plus nombreuses donc nous allons nous attarder sur les plus importantes.

LightManager

Crée une lumière et automatise ses fonctionnements comme créer sa shadowMap en fonction de son type, préciser au shader le type d'ombre à afficher, et gérer son placement dans l'espace.

ObjectManager

Crée un objet selon 3 types : simple, texturé ou modélisé, et va permettre l'automatisation de tous les objets avec le même rendu. Il gère l'existence ou non d'un LOD et envoie les variables uniformes au shader.

ParticleManager

Crée des particules à partir d'un endroit précisé par l'utilisateur et rafraîchit les particules selon un cycle de vie. Les particules mortes sont automatiquement renouvelées et limitées selon le besoin.

Character

Gère le personnage à afficher ainsi que la lumière qui lui est attribué. Adapte la rotation du personnage selon la direction dans laquelle il avance. C'est également lui qui est en charge de où le personnage a le droit d'aller sur la carte.

✓ src-common
✓ boids
⌚+ Boid.cpp
⌚+ Boid.hpp
⌚+ Flock.cpp
⌚+ Flock.hpp
⌚+ Tools.cpp
⌚+ Tools.hpp

Boïd

Représente chaque objet boïd, avec ses valeur de positions, de vitesses.

Flock

La classe Flock sert à gérer un groupe de boïd, et à leur appliquer les différentes règles en leur calculant un vecteur vitesse en 3 dimension en fonction de leur position et de la position des autres boïds.

Tools

La classe Tools, contient des méthodes, qui me sert à implémenter les boïds, comme notamment, une méthode de calcul de position aléatoire, utiliser pour le début de la simulation.

VI. Difficultés rencontrées

Lilou :

Impossible de comprendre la logique entre Blender et OpenGL : le modèle peut avoir la bonne taille mais être pivoté de 90° en x et bizarrement, dès qu'il y a une armature le modèle est 100 fois trop grand mais pivoté correctement.

Le modèle ne s'affiche pas si l'armature n'est pas centré dans Blender (à cause du fois 100?).

Si un objet possède une armature, les autres objets qui n'en possèdent pas ne s'affichent pas, j'ai dû afficher les objets touchés par ce problème en 2 morceaux.

La cubeShadowMap a l'air correcte mais affiche des artifices sous certains angles. J'ai pourtant vérifié plusieurs fois l'angle de visu de la ViewMatrix sans succès.

Le fait d'avoir plusieurs Samplers différents sur un même shader crée un « warning » : UNSUPPORTED (log once): POSSIBLE ISSUE: unit 1 GLD_TEXTURE_INDEX_2D is unloadable and bound to sampler type (Float) - using zero texture because texture unloadable

Même si la texture 1 est bien affichée. De plus, impossible d'utiliser ces textures via un tableau à cause d'une « règle » OpenGL qui stipule que l'index doit être un uniform.

Pour contourner cette erreur, je crée toutes les variables une par une et utilise un switch mais encore une fois une autre erreur survient;

Utiliser une shadowMap via un switch qui prend un entier sous forme de variable en paramètre crée du lag. Plus exactement si je veux envoyer l'entier « 1 » via une variable « i » tel que f(i) le programme lag, tandis que si j'appelle directement f(1) tout va pratiquement bien.

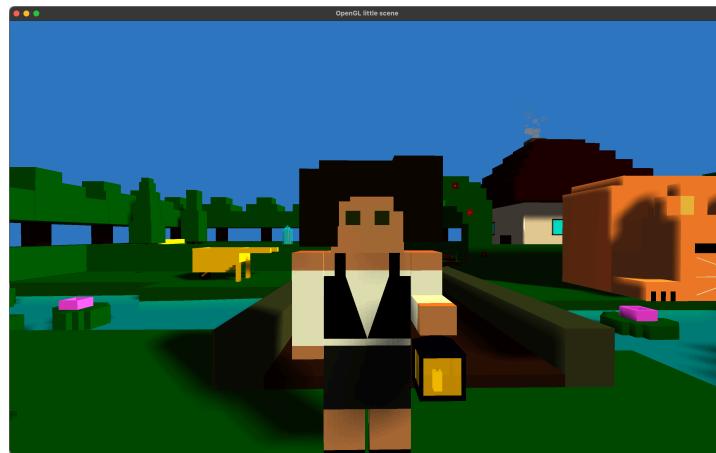
Pour résoudre ce problème j'utilise le fastidieux « Si i = 1 alors j'appelle f(1) » pour tous les cas. De ce fait, je me suis limitée à 3 lumières.

Je n'ai pas trouvé la lumière ambiante sur Blender (il renvoie du blanc), je fais donc k * couleur diffuse.

Les armatures sur des objets larges provoquent des erreurs liés à ma classe SkinnedMesh qui n'accepte que 4 os par vertice. Quand cela arrive je ne mets pas plus de 4 os dans mon animation.

Gérer le déplacement de la Trackball Caméra autour du carapateur. En effet j'ai dû m'adapter en prenant en compte l'angle avant le déplacement de celle-ci afin d'éviter d'avancer uniquement sur l'axe Z.

Je n'ai pas trouvé comment sortir une texture de mes modèles 3D simplement colorés sur Blender. Actuellement, le projet prend chaque partie de l'objet et affiche sa couleur séparément.



Benjamin :

J'ai eu du mal à débuter le code de simulation de boïds, je ne savais pas trop où j'allais, mais une fois que j'ai commencé à utiliser un vecteur pour chaque force, ça s'est beaucoup mieux passé.

La simulation de boïd peut parfois être assez difficile à débugger, il y a plein de paramètres, et de facteurs qui influent sur leur comportement, donc il est assez difficile de trouver l'erreur sans faire de nombreux essais, pour cela ImGui m'a beaucoup aidé, afin de pouvoir modifier les coefficients de chaque force, et les distances d'application de chaque force.

Le temps de make du projet, quand à un certain moment j'avais besoin de rebuild le projet, le temps d'attente est assez long.

Débugger OpenGL, les messages d'erreurs sont quasi inexistant, et il faut donc avancer à tâton.

Mettre en relation les fichiers Fbx de Blender et OpenGL, ils étaient la plupart du temps mal orienté, les tailles étant aussi parfois assez surprenante.

