HAI928I
Projet Jeux 3D

Moteur de jeu

BARDIN Melvin
JAFFRET Laurine

Sommaire

- Environnement et Personnage
- Fonctionnalité
- Graphe de scène
- Bruit de Perlin
  - Bref récapitulatif
  - Le terrain
  - Les arbres
  - les nuages
- La pipeline OpenGl
  - Vertex shader
  - Tessellation Control shader
  - Tessellation evaluation shader
  - Geometry shader
  - Fragment shader
- Les animations
  - de maillages
  - de Textures
- La physique et la boucle de rendu
  - la gravité et perte cinétique
  - les collisions
    - AABB
    - Objets - Terrain
    - Objets - object
- Démonstration
- Amélioration possible

Environnement et personnage

- Décors
  - Terrain infini
  - Arbre
  - Nuage
  - Soleil
  - Boite
- Joueur (le truc qui ressemble à un avion)

- Vue à la 3ème personne

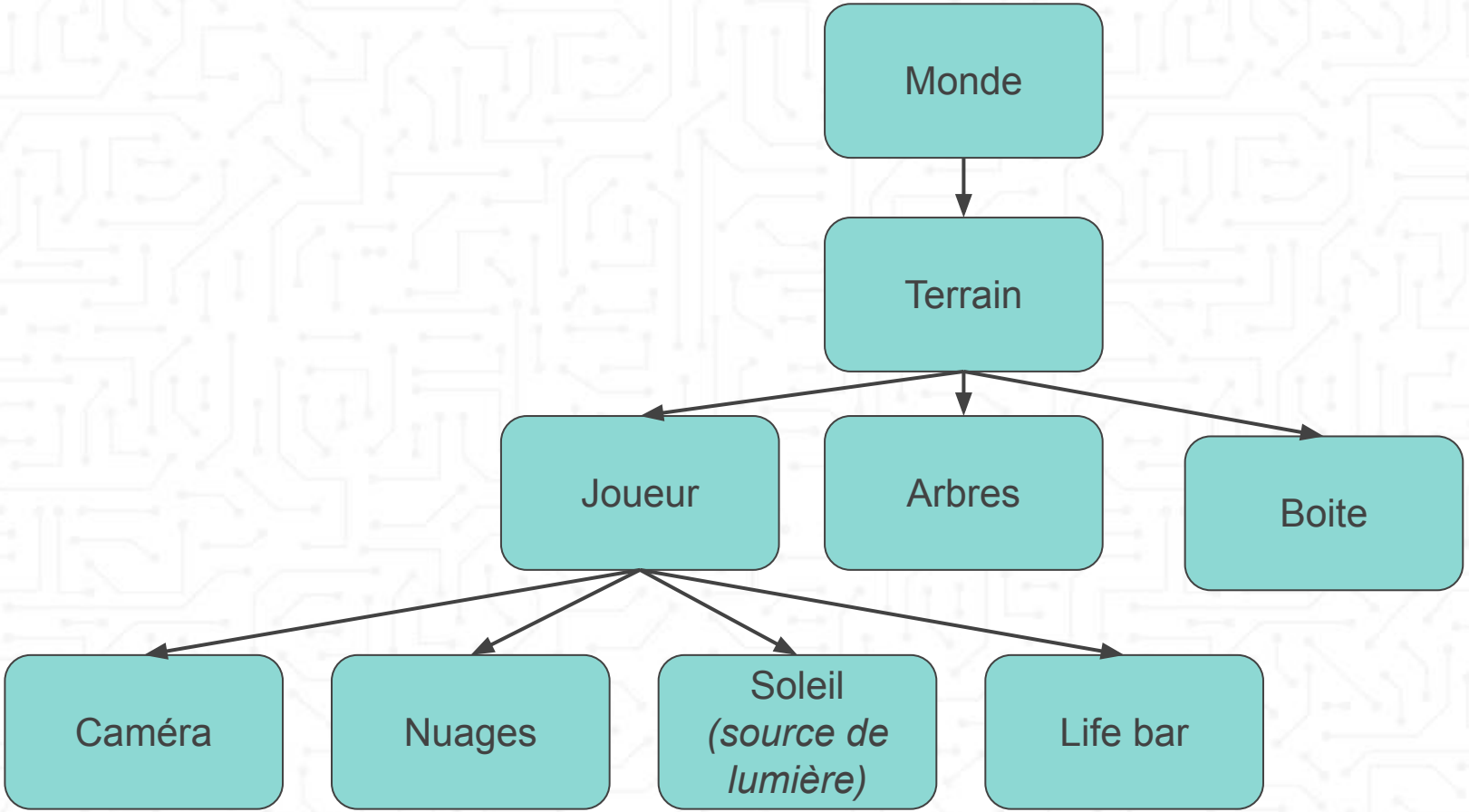
Fonctionnalités

- Se déplacer avec les touches du clavier
  - o : Avancer
  - l : Reculer
  - k : Pivoter droite
  - m : Pivoter gauche
- Modifier vue de la caméra
  - z q s d, et pivoté caméra avec la souris

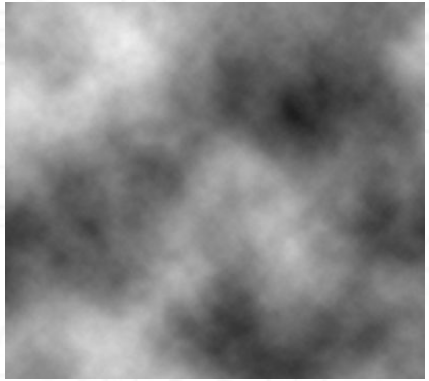
- Génération procédurale du terrain d'arbre et nuage infinie en temps réel en fonction de la position du joueur
- animation du terrain

- Gravité :
  - Collisions
  - Perte cinétique

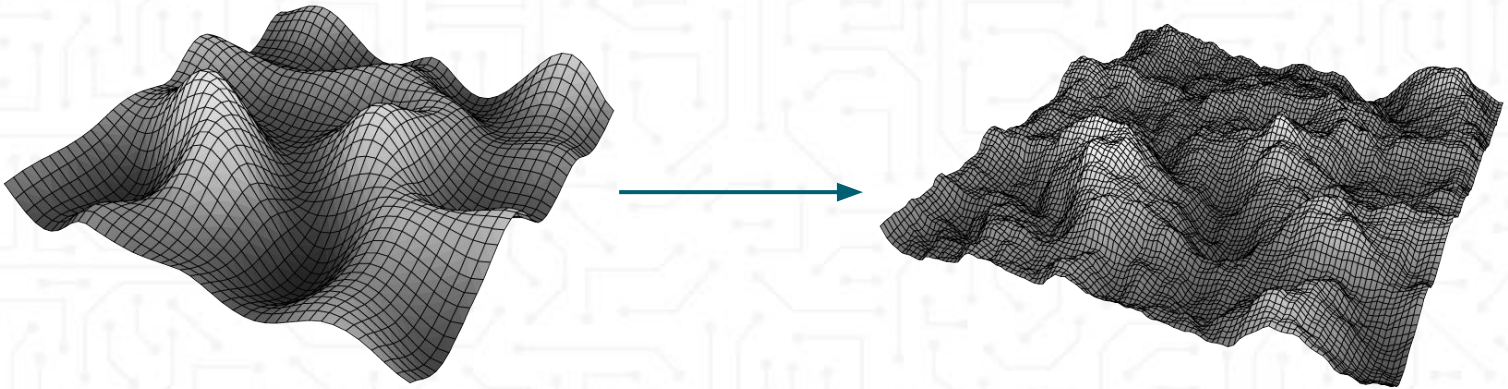
Graphe de scène



Le bruit de Perlin
Bref récapitulatif

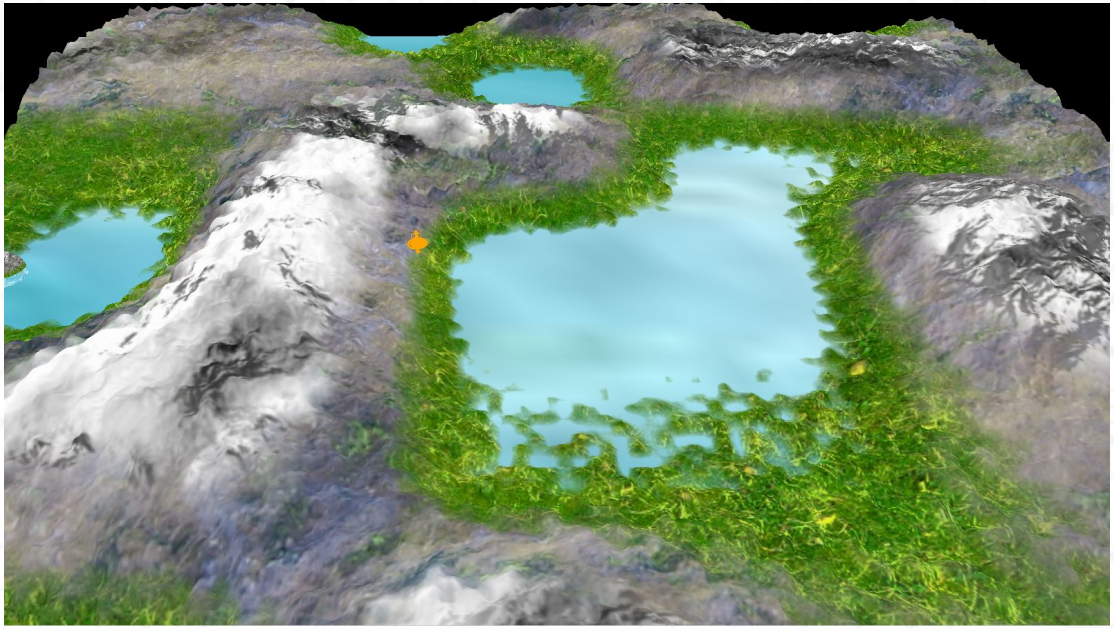


- Dans les grandes lignes
  - Pseudo aléatoire,
  - Utilise une grille de vecteur de gradient
  - Calculs sur le GPU rapide
  - Combinaison de plusieurs bruits de Perlin



Le bruit de Perlin
Le terrain

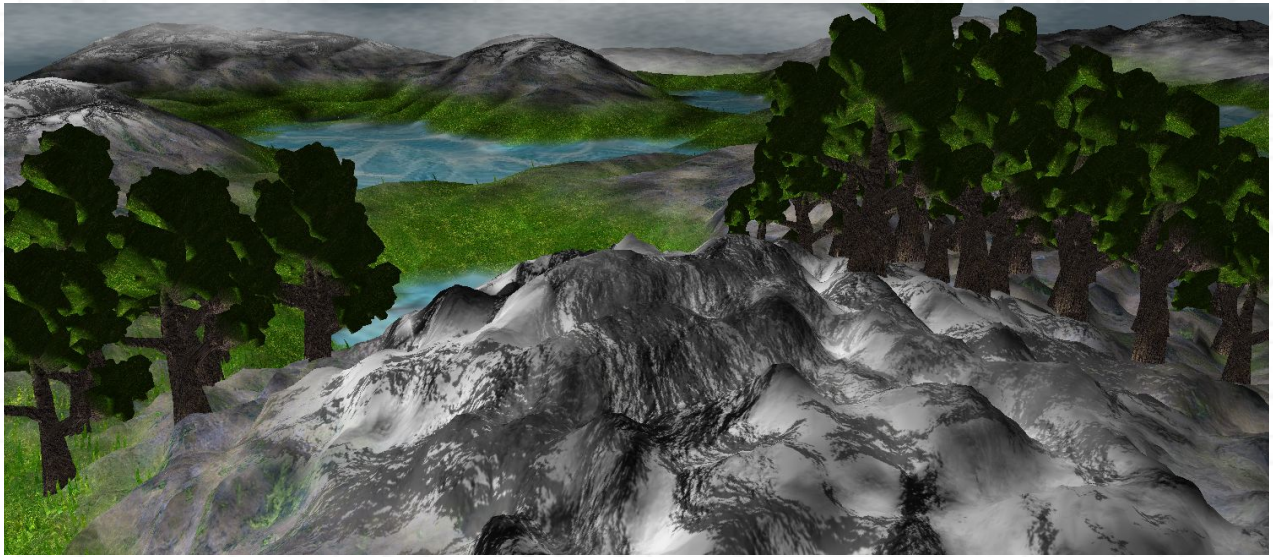
- Terrain infini
- Se génère dynamiquement lorsque le joueur se déplace
- Combinaison de 8 bruits de Perlin



Le bruit de Perlin
Les Arbres

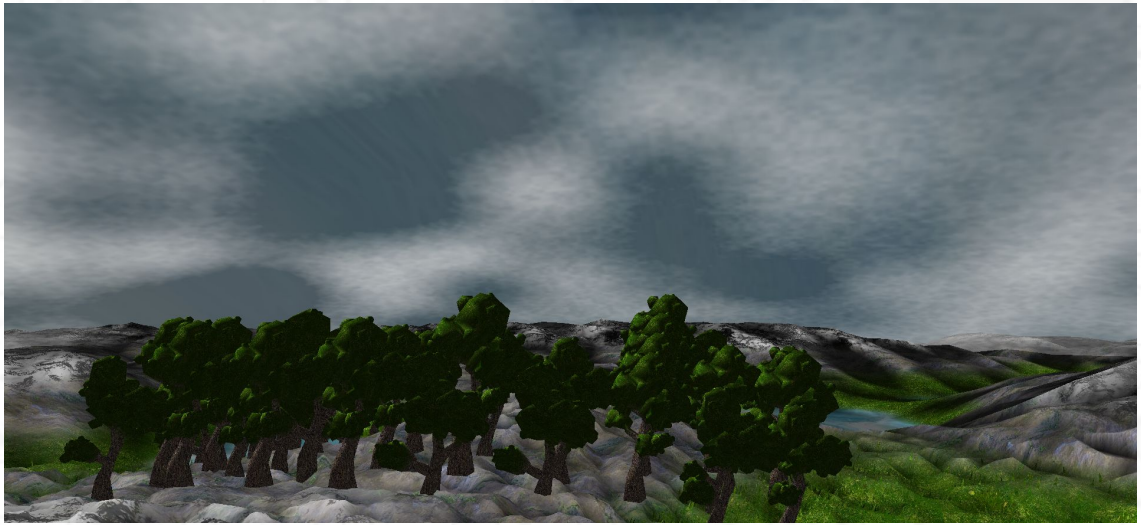
- Nombre d'arbre "infini"
- Se génère aussi dynamiquement lorsque le joueur se déplace
- porté réduite d'apparition des arbres par rapport à la distance de construction du terrain

Condition d'apparition d'arbre:
- grille invisible
- seuillage du Bruit
- condition de hauteur avec récupération du bruit du terrain
- bruit pour l'orientation/inclinaison de l'arbre



Le bruit de Perlin
Les Nuages

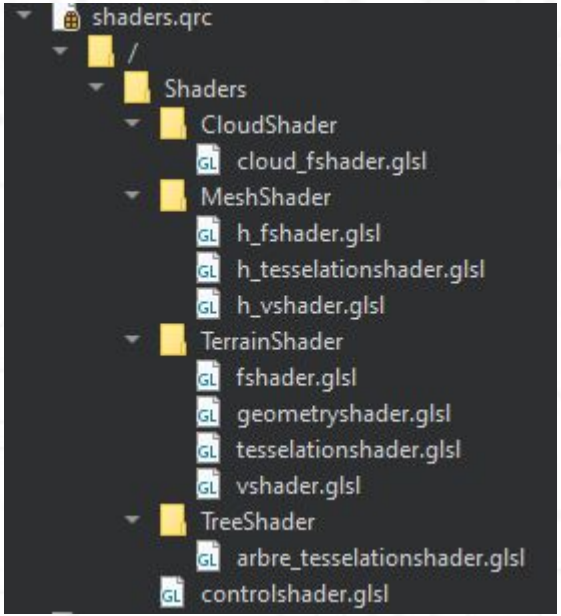
- plain incurvé survolant tout le terrain
- initialement blanc
- le bruit fait varier la transparence

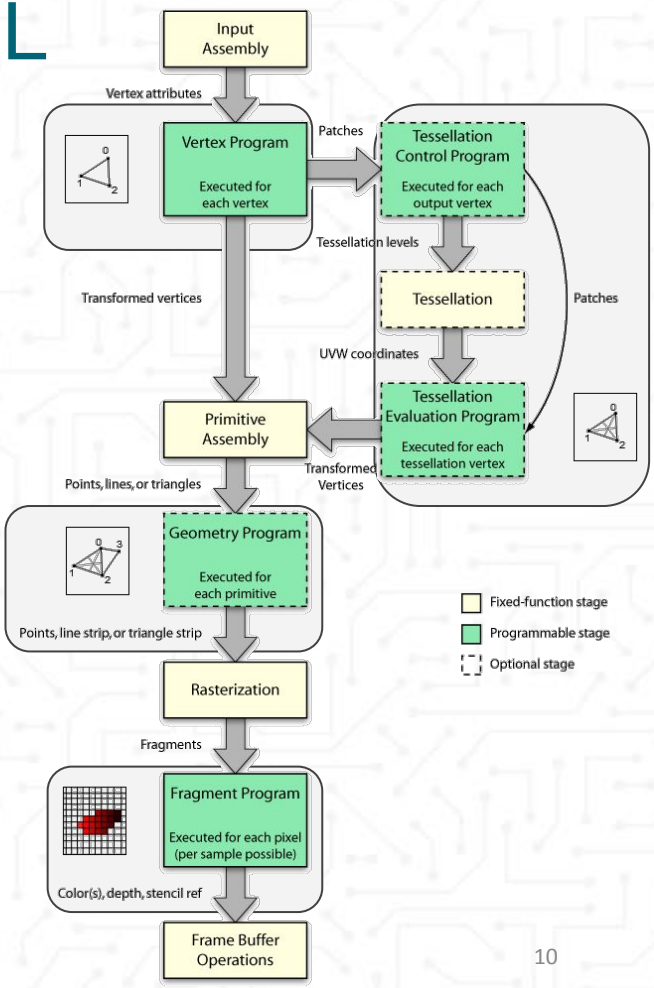


Pipeline OpenGL

Différents shaders en fonction du besoin

- Le terrain
- Les meshs
- Les nuages
- Les arbres





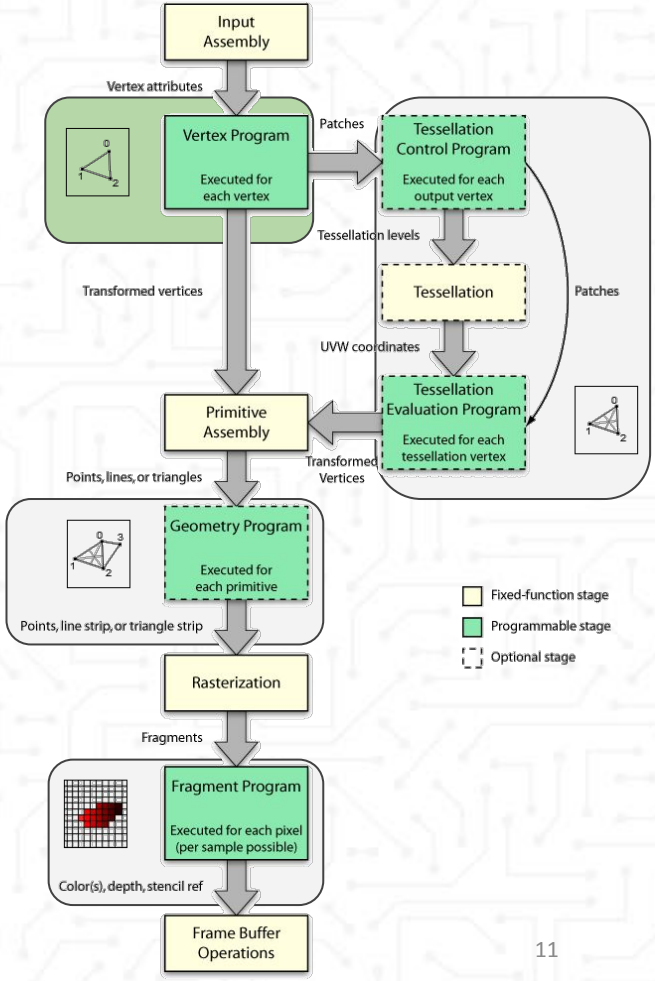
Vertex Shader

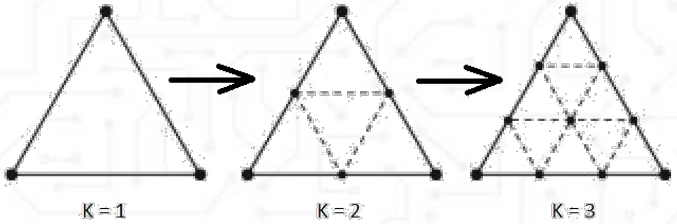
Nos Vertex Shaders se contente seulement de transmettre les données à la couche suivante

```
uniform mat4  transform_Matrix;
in vec4 a_position;
in vec2 a_texcoord;
in vec3 a_normal;
out vec2 v_texcoord;
out vec3 v_position;
out vec3 v_normal;
out vec3 FragPos;
uniform sampler2D texture;

//! [0]
void main()
{
    FragPos = vec3(transform_Matrix * vec4(a_position.xyz, 1.0));
    v_texcoord = a_texcoord;
    v_position = vec3(a_position.xyz);
    v_normal = a_normal;
}
```

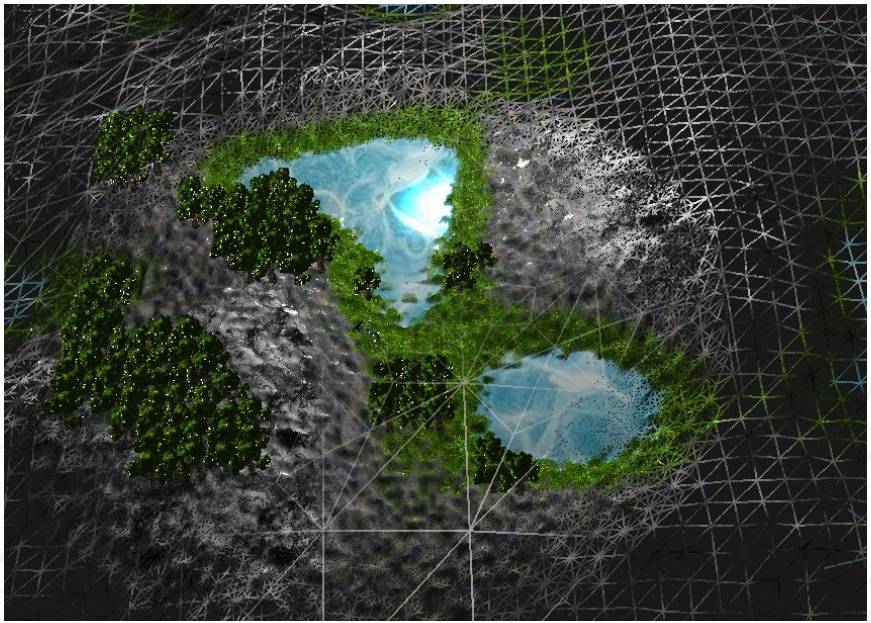
Vertex Shader pour les maillage, nuages et arbre

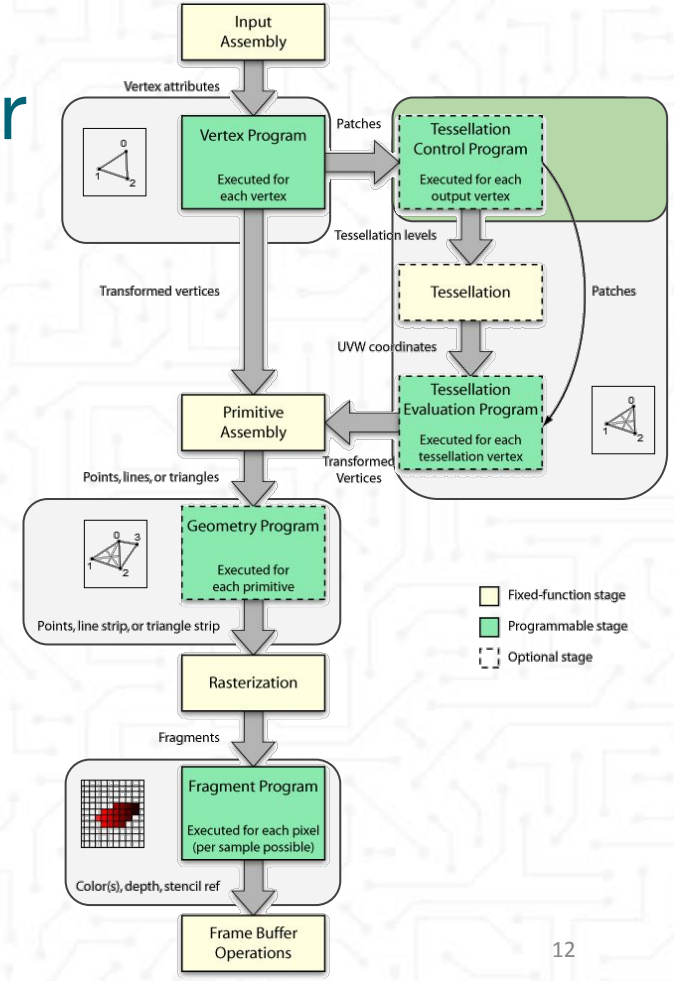




Tessellation Control shader

Le tessellation control shader consiste à indiquer le nombre de subdivision à apporter à chaque patch

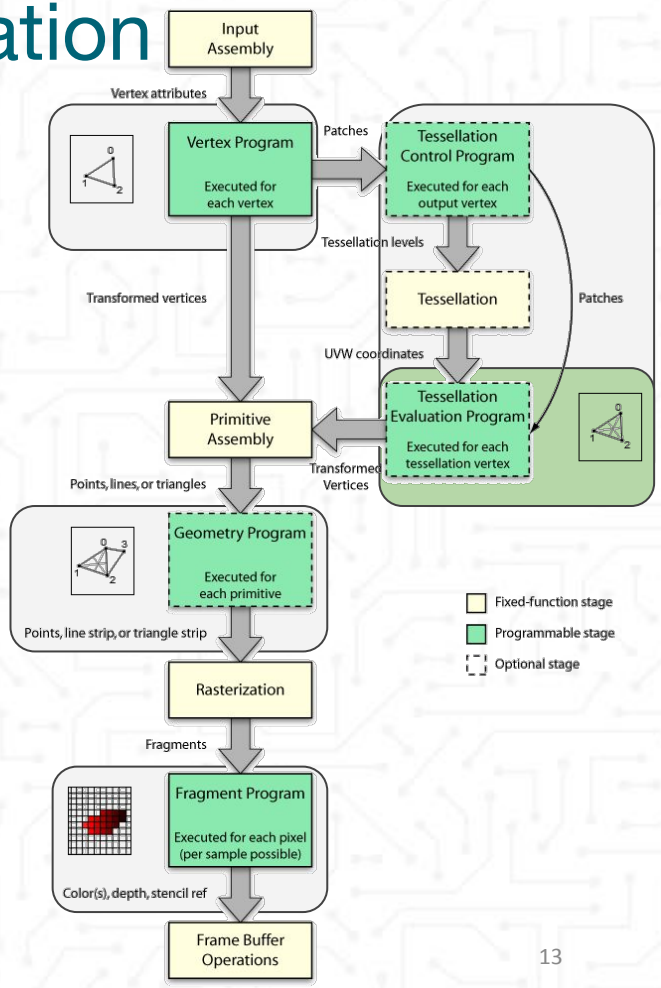




Tessellation Evaluation shader

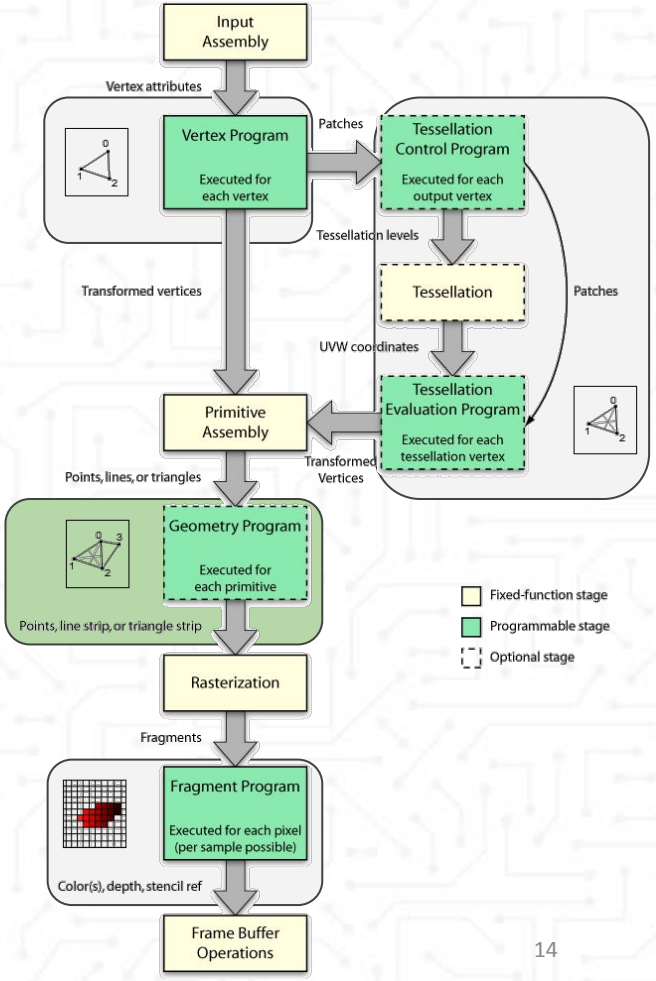
Le tessellation Evaluation shader consiste à interpoler les nouveau triangles

- Terrain
  - Calculs du bruit de Perlin
  - Animation de l'eau
- Meshs , nuages
  - Interpolation classique
- Arbres
  - Interpolation classique
  - Animation des feuilles



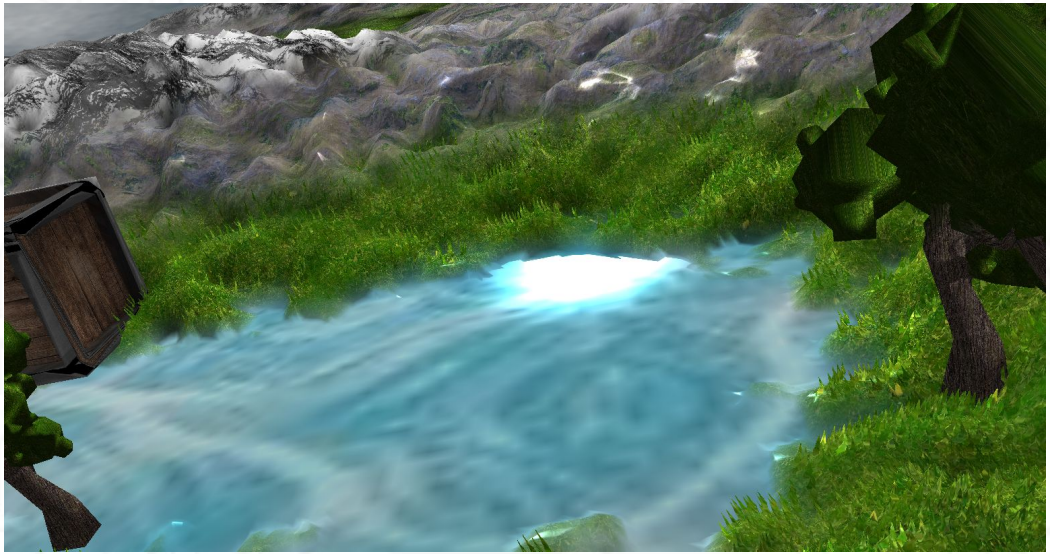
Geometry shader

Le geometry shader consiste à ajouter des point/triangle à chaque triangle

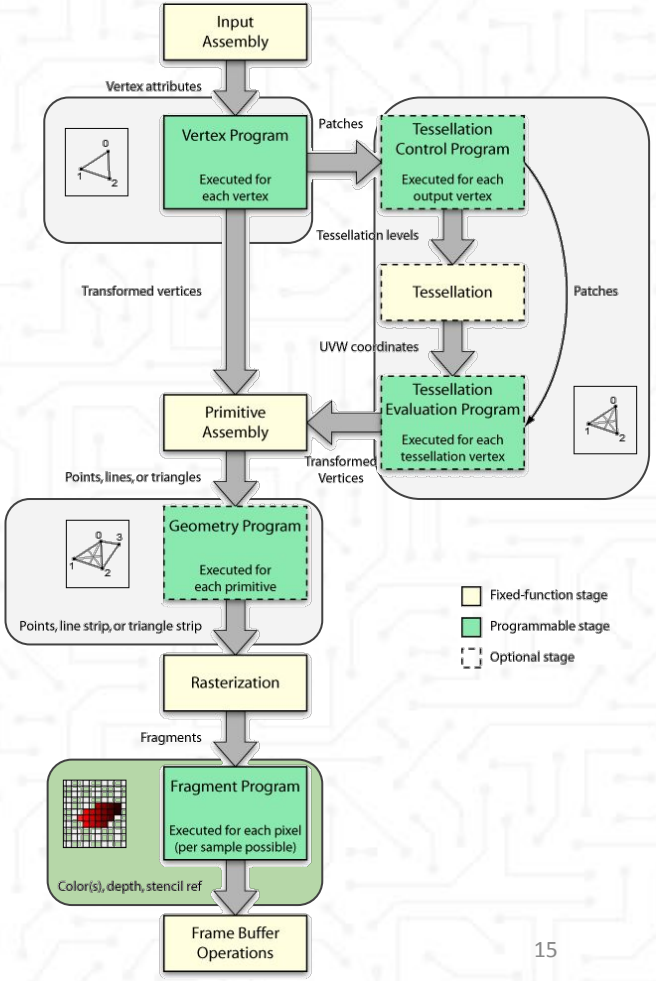


Fragment shader

- Modèle de phong



- Animations des textures (l'eau, feuillage des arbres)



Animation

- Toutes les animations sont gérées dans les shaders
- Une variable de temps + sinusoïde = Animation

- Pour les arbres: la hauteur influence l'oscillation

Oscillation plus forte à
la cime de l'arbre

Animation

- Toutes les animations sont gérées dans les shaders
- Une variable de temps + sinusoïde = Animation

- Pour l’eau: on ajoute un biais (la coord y) afin de donner un effet de vague qui se déplace

Animation

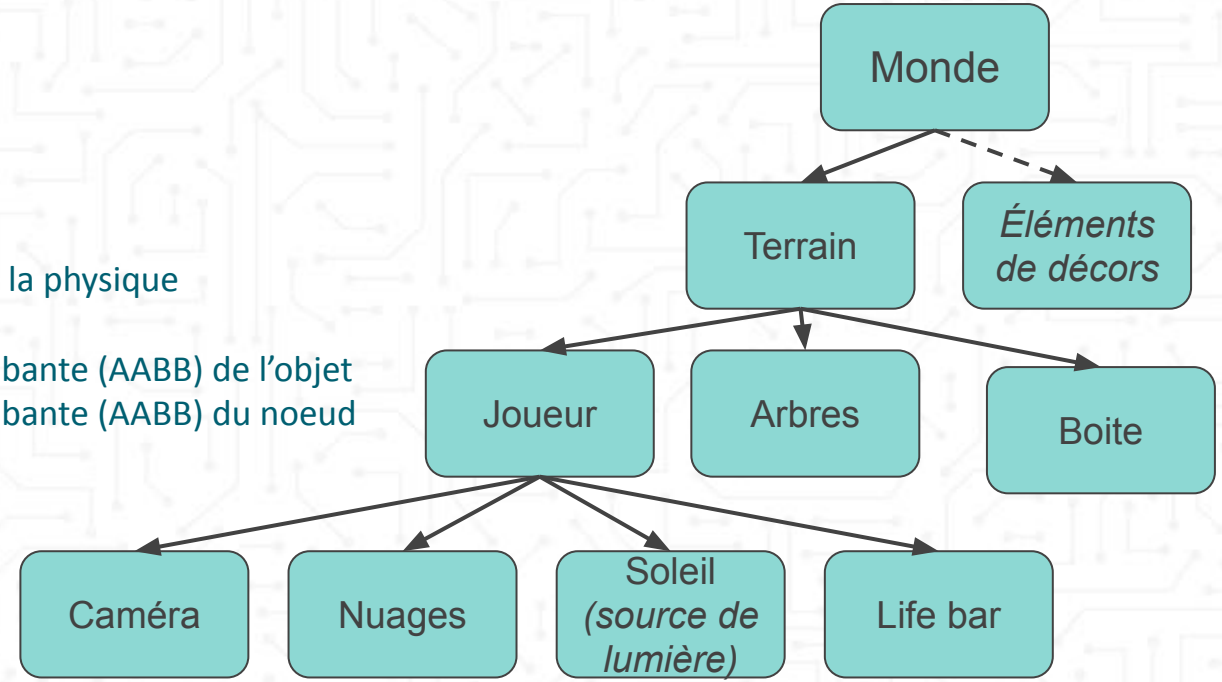
- Toutes les animations sont gérées dans les shaders
- Une variable de temps + sinusoïde = Animation

- Pour les texture:
  - somme de deux animation à une vitesse d’oscillation différente

La physique et la boucle de rendu

La boucle de rendu :

- Actualise les positions
- Gestion des collisions, de la physique
- rendu (draw)
- Actualise les boîtes englobante (AABB) de l'objet
- Actualise les boîtes englobante (AABB) du noeud



La physique et la boucle de rendu
la gravité et perte cinétique

Perte cinétique et gravité deux fonction géométrique
- problème : deltaTemps

La gravité est définie par:

```
QVector3D(0,0,-(pow(1.1,(deltaTime/facteurGravite)))/10);
```

```
float facteurGravite =300;
float facteurCynetique = 50;
```

La perte cinétique est définie par:

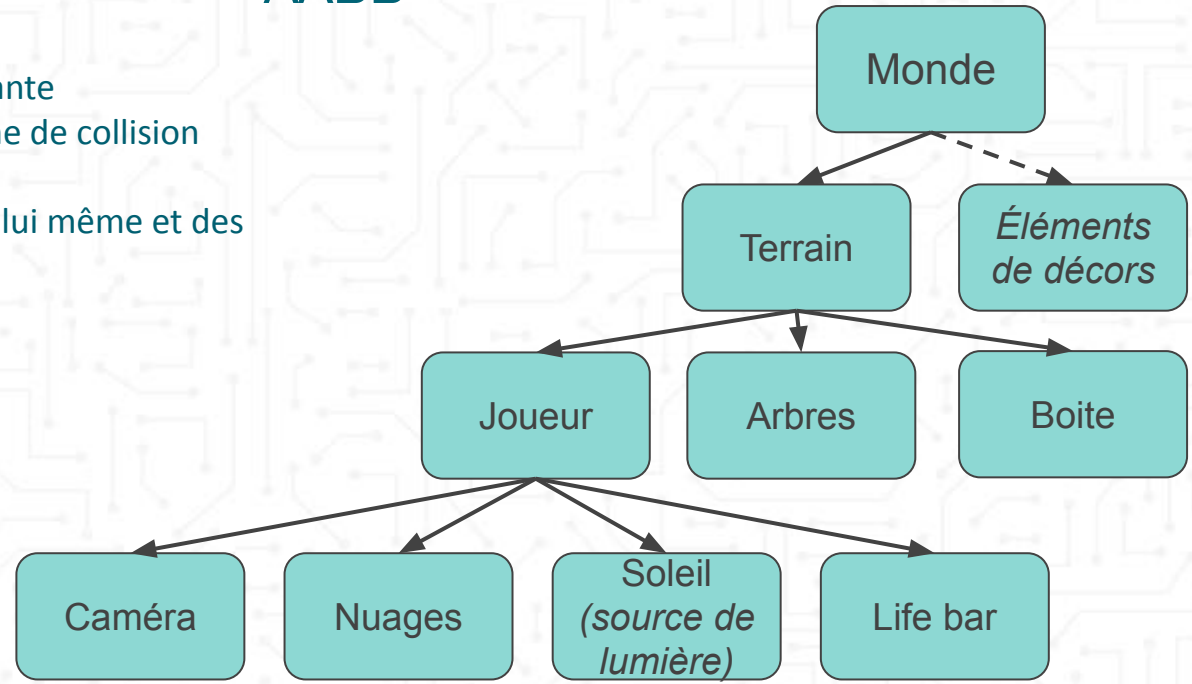
```
pow(0.9,deltaTime/facteurCynetique)
```

Elle est appliqué sur x et y, et sur z si z >0

La physique et la boucle de rendu

AABB

- Hiérarchie de boite englobante
  - optimiser la recherche de collision
- AABB du noeud
  - englobe les AABB de lui même et des enfants
- AABB du mesh



La physique et la boucle de rendu

Collision: objet – terrain

la difficulté:

- Les deux GameObject on un repère différent

solution:

- Convertir les deux GameObject dans le repère monde

Détection de la collision:

- bruit de perlin au coordonnée de l'objet
- si hauteur de l'objet < hauteur du terrain alors collision

Réaction à la collision:

- recalage
  - déplacement de l'objet sur la surface
- réflexion
  - éviter toutes erreurs ⟶ vecteur direction toujours avec z > 0

La physique et la boucle de rendu

Collision: objet – Objet

la difficulté:

- Les deux GameObject on un repère différent

solution:

- Convertir les deux GameObject dans le repère monde

Détection de la collision:

- entre les AABB

Réaction à la collision:

- réflexion
  - éviter toutes erreurs, ne rien faire si:
    - produit scalaire entre vecteur réfléchie et le vecteur BA > 0
    - avec A le centre de l'objet courant
    - avec B le centre de l'objet subissant la collision
- l'objet B est repoussé

Démonstration

Amélioration possible

- Plus de fonctionnalités de jeu
  - Objets à ramasser
  - Ecran de fin
  - Condition de victoire et d'échec
- Nager, plan ajouter fond marin
- Physique plus réaliste
  - Amélioration des collisions de l'objet (rebonds)
  - Gestion multi-collision plus efficace
- Capturer la souris
  - Caméra guidé par la souris
- Cacher les objet gênant à la caméra (cachant le personnage)

Merci !
Des questions ?

Bibliographie

- Bruit de Perlin *(https://fr.wikipedia.org/wiki/Bruit_de_Perlin)*
- Opengl tutorial *(http://www.opengl-tutorial.org)*
- Learn Opengl *(https://learnopengl.com/)*
- OpenGL in Qt 5.1 *(https://www.kdab.com/opengl-in-qt-5-1-part-5/)*

Melvin BARDIN : melvin.bardin@etu.umontpellier.fr
Laurine JAFFRET : laurine.jaffret@etu.umontpellier.fr