Université de Montpellier
Master 2 - IMAGINE

Projet 3D

Rapport

Aquarium

Étudiant :
M. Florentin DENIS

Année : 2022 - 2023

Table des matières

Introduction

Vous pourrez trouver à ce lien Github le code source du projet, ainsi que la documentation Doxygen et le moyen de compiler / exécuter le projet :
https://github.com/Flare00/M2-Projet3D-Aquarium

Vous pourrez également trouver la documentation en ligne à ce lien :
https://www.flareden.fr/Aquarium3D/index.html

Les Librairies utilisées sont :

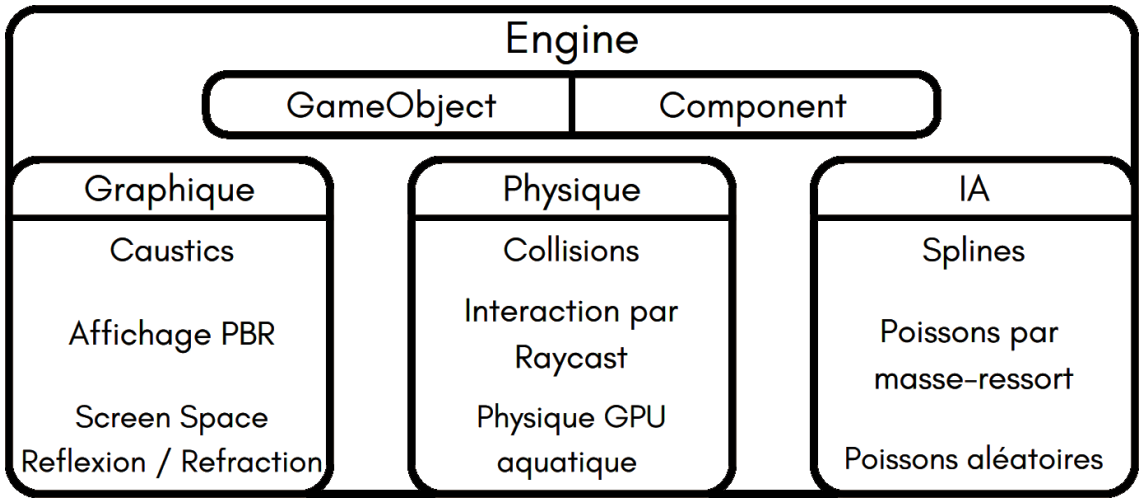
— GLEW - 2.2.0
— GLFW - 3.3.8
— GLM - 0.9.9.8
— OpenFBX
— stb image

Version d'OpenGL : 4.3
Version CMake : 3.21

Chapitre 1

Architecture

Le projet est composé d'une architecture ECS, nous avons GameObject qui est l'Entity, Component, les composants, et enfin, Graphique, Physique et IA, les systèmes. Engine sert à gérer et ordonner les différents systèmes.



Tout le code compilé est stocké sous "/aquarium/sources/".
Tout le code Shader est stocké sous "/assets/Shaders/".

Chapitre 2

Fonctionnalités (Code)

2.1 Spline - Catmull-Rom

```
class Spline {
private:
  std::vector<glm::vec3> points;
  std::vector<float> keyframeTime;
  float totalLength;
  bool cyclic;
public:
  Spline() {}
  Spline(std::vector<glm::vec3> points, bool cyclic = true)
  {
    this->points = points;
    this->cyclic = cyclic;

    if (cyclic) {
      this->points.push_back(points[0]);
    }
    else {
      this->points.insert(this->points.begin(), this->points[0]);
      this->points.push_back(points[points.size() - 1]);
    }

    ComputeKeyframeTimes();
  }

  /// <summary>
  /// Use a given progression to return the current points with his own progression.
  /// </summary>
  /// <param name="progression"></param>
  /// <returns>Return the pair of points index and his progression</returns>
  std::pair<size_t, float> FindSegment(float progression) {
    size_t index = 0;
    while (index < keyframeTime.size() - 2 && keyframeTime[index + 1] <= progression)
      index++;

    if (cyclic) {
      index = index % keyframeTime.size();
    }
    float time = (progression - keyframeTime[index]) / (keyframeTime[(index + 1) %
↪  keyframeTime.size()] - keyframeTime[index]);

    return { index,time };
  }

  /// <summary>
```

```
44    /// Catmull-Rom function, give the current position on the spline using a progression value
      ↪  between 0 and 1.
45    /// </summary>
46    /// <param name="t">Progression on the spline (between 0 and 1)</param>
47    /// <returns>The 3D coordinate of the progression on the spline.</returns>
48    glm::vec3 Interpolate(float progression) {
49      auto [index, time] = FindSegment(progression);
50
51      if (cyclic) {
52        index = index % points.size();
53      }
54      else {
55        if (index == 0) index++;
56        if (index == points.size() - 1) index--;
57      }
58      return 0.5f * ((2.f * points[index]) + //Control Point 0
59        (-points[(index - 1 + points.size()) % points.size()] + points[(index + 1) % points.size()])
      ↪  * time + //Control Point 1
60        (2.f * points[(index - 1 + points.size()) % points.size()] - 5.f * points[index] + 4.f *
      ↪  points[(index + 1) % points.size()] - points[(index + 2) % points.size()]) * time * time +
      ↪  //Control Point 2
61        (-points[(index - 1 + points.size()) % points.size()] + 3.f * points[index] - 3.f *
      ↪  points[(index + 1) % points.size()] + points[(index + 2) % points.size()]) * time * time *
      ↪  time); //Control Point 3
62    }
63
64    /// <summary>
65    /// Compute the Keyframe time of each point, based on the length of each segment.
66    /// </summary>
67    void ComputeKeyframeTimes() {
68      if (keyframeTime.size() != points.size())
69        keyframeTime.resize(points.size());
70
71      //Calcul la longueur totale
72      totalLength = 0;
73      for (size_t i = 1; i < points.size(); i++)
74        totalLength += glm::length(points[i] - points[i - 1]);
75
76
77      //Calcul la longueur actuelle et assigne au keyframeTime sa valeur entre 0 et 1.
78      double current_length = 0;
79      keyframeTime[0] = 0;
80      for (size_t i = 1; i < points.size(); i++) {
81        current_length += glm::length(points[i] - points[i - 1]);
82        keyframeTime[i] = current_length / totalLength;
83      }
84    }
```

2.2 FishBank - Masse-Ressort

```
class FishBank : public Component, public EngineBehavior
{
protected:
  Spline* spline;
  ModelInstanced* fish;
  std::vector<std::vector<size_t>> links;
  std::vector<glm::vec3> velocities;
  double avancement = 0.0;
  double speed = 0.02;
  float distRepos;
  float k = 0.5f; //Raideur
public:

  /// <summary>
  /// Create a FishBank
  /// </summary>
  /// <param name="fish">The model of the fish</param>
  /// <param name="spline">The spline that the Fish bank follow</param>
  /// <param name="distRepos">The rest distance between fish</param>
  /// <param name="nbLayer">The number of layer for the construction</param>
  /// <param name="initialAvancement">The default start position on the spline.</param>
  FishBank(ModelInstanced* fish, Spline* spline, float distRepos, int nbLayer, double
↪  initialAvancement = 0.0f) {
    this->fish = fish;
    this->spline = spline;
    this->avancement = initialAvancement;
    this->distRepos = distRepos;

    //Genere les points et les liens entre les points.
    this->fish->SetPositions(ModelGenerator::MultiLayerSphere(this->links, nbLayer, distRepos));

    //Initialise la position en utilisant l'interpolation de la spline à l'avancement donnée.
    glm::vec3 init = spline->Interpolate(this->avancement);
    std::vector<glm::vec3> pos = this->fish->GetPositions();
    glm::vec3 d =  init - pos[0];

    //Déplace l'ensemble des points à la position précédemment calculée.
    for (size_t i = 0, max = pos.size(); i < max; i++) {
      pos[i] += d;
      velocities.push_back(glm::vec3(0));
    }

    //Set les points sur le model.
    this->fish->SetPositions(pos);
  }

  /// <summary>
  /// The loop fonction to update positions.
  /// </summary>
  /// <param name="deltaT"></param>
  void loop(double deltaT) override {
    this->avancement += deltaT * this->speed;
    if (this->avancement >= 1.0) {
      this->avancement = 0.0;
    }
    UpdatePositions(deltaT);
```

```
56    }
57
58    /// <summary>
59    /// Update the position of the central fish, and move the fish bank.
60    /// Also update the distance between elements.
61    /// </summary>
62    void UpdatePositions(double deltaT) {
63      //Récupère les points
64      std::vector<glm::vec3> positions = this->fish->GetPositions();
65      //Trouve la position grace à la Spline;
66      glm::vec3 tmp = spline->Interpolate(this->avancement);
67      glm::vec3 dir = tmp - positions[0];
68      // met la valeur de la position au point centrale.
69      positions[0] = tmp;
70      //Parcours les points
71      for (int i = 1, max = this->links.size(); i < max; i++) {
72        glm::vec3 force(0.0f);
73        //Pour tout points parcours ses voisins
74        for (int j = 0, maxJ = this->links[i].size(); j < maxJ; j++) {
75          //Calcul la force, en se basant sur la rigidité k
76          //la différence entre la distance actuelle, et celle de repos
77          //et la normal de direction entre le point courant et le voisins courant.
78          glm::vec3 np = positions[links[i][j]] - positions[i];
79          float d = glm::length(np);
80          force += k * (d - distRepos) * glm::normalize(np);
81        }
82        //Applique la force à la velocité, ainsi que la vélocité à la position.
83        this->velocities[i] += force * (float)deltaT;
84        positions[i] += velocities[i] * (float)deltaT;
85        //Ajoute en partie la distance parcouru par le point centrale, pour eviter que les poissons
      ↪  disparaisse instantanément.
86        positions[i] += dir * 0.75f;
87      }
88
89      //Set for instanciation
90      this->fish->SetPositions(positions);
91    }
92  };
93
```

2.3 Eau GPU

2.3.1 water.vert

```
#version 430

layout(location = 0) in vec3 aPos;
uniform vec2 halfSize;
out vec2 coord;

void main(){
  coord = (aPos.xz+vec2(1)) / 2.0;
  gl_Position =  vec4(aPos.x, aPos.z, aPos.y, 1.0);
}
```

2.3.2 water.frag

```
#version 430
const float PI = 3.141592;

out vec4 color;
in vec2 coord;
uniform sampler2D tex;
uniform float deltaTime;
uniform vec2 deltaMove;

void main(){
    //parse data
    vec4 data = texture(tex, coord);
    vec2 dx = vec2(deltaMove.x, 0.0);
    vec2 dy = vec2(0.0, deltaMove.y);
    //Compute the average movement
    float average = (
        texture(tex, coord - dx).r +
        texture(tex, coord - dy).r +
        texture(tex, coord + dx).r +
        texture(tex, coord + dy).r
    ) * 0.25;

    // change velocity to average
    data.g += (average - data.r) * 2.0;
    // attenuate the velocity a little so waves do not last forever
    data.g *= 1.0 - (deltaTime * 0.1);
    // move the vertex along the velocity
    data.r += data.g;

    // compute the normal
    vec3 ndx = vec3(deltaMove.x, texture(tex, vec2(coord.x + deltaMove.x, coord.y)).r - data.r,
↪  0.0);
    vec3 ndy = vec3(0.0, texture(tex, vec2(coord.x, coord.y + deltaMove.y)).r - data.r,
↪  deltaMove.y);
    data.ba = normalize(cross(ndy, ndx)).xz;
    //Set the data
    color = data;
}
```

2.3.3 drop.frag

Vertex Shader : water.vert

```
#version 430
const float PI = 3.141592;

out vec4 color;
in vec2 coord;

uniform sampler2D tex;
uniform vec2 center;
uniform float radius;
uniform float strength;

void main(){
    vec4 data = texture(tex, coord);
    //Add the drop.
    float d = max(0.0 , 1.0 - length(center - coord) / radius);
    d = 0.5 - (0.5 * cos(d * PI));
    if(d > 0.01){
        data.r = d * strength;
    }
    color = data;
}
```

2.3.4 caustics.vert

```
#version 430
layout(location = 0) in vec3 aPos;

uniform vec2 halfSize;
out vec2 coord;

void main(){
  coord = (aPos.xz+vec2(1)) / 2.0;
  gl_Position =  vec4(aPos.x, aPos.z, aPos.y, 1.0);
}
```

2.3.5 caustics.frag

```
#version 430
layout (location = 0) out vec4 gAlbedo;
layout (location = 1) out vec4 gPosition;
layout (location = 2) out vec4 gNormal;
layout (location = 3) out vec4 gMetalnness;
uniform sampler2D waterData;
uniform vec3 lightDirection;
in vec2 coord;

void main(){
    vec4 d = texture(waterData, coord);
    vec3 normal = vec3(d.b, sqrt(1.0 - dot(d.ba, d.ba)), d.a);
    vec3 reflection = reflect(lightDirection, normal);
    float caustic = pow(clamp(dot(reflection, vec3(0, 0, 1)), 0.0, 1.0), 2.0);
    gAlbedo = vec4(vec3(1,1,1) * caustic, 1.0f);
}
```

2.4 Screen Space Reflexion / Refraction (SSR)

Fichier : pbr.frag

```
//Si utilise le prérendu (deuxième draw)
if(u_use_pre_render == 1 && Metal > 0.01){
    //récupère le rayon depuis le repère camera;
    vec3 incomingRay = normalize(PointCoord.xyz);
    //Calcul la normale du points
    vec3 normal = normalize(Norm);
    //Calcul la reflection et refraction
    vec3 reflectedRay = reflect(incomingRay, normal);
    vec3 refractedRay = refract(incomingRay, normal, 1.33);
    //En deduit un fresnel entre 0 et 1
    float fresnel = mix(0.5, 1.0, pow(1.0 - dot(normal, -incomingRay), 3.0));
    //Récupère les couleurs du rayon réfléchi et refracté
    vec4 refractColor = texture(t_pre_render, refractedRay.xy);
    vec4 reflectColor = texture(t_pre_render, reflectedRay.xy);
    //melange les deux couleurs si besoin
    vec4 mixed =  mix(refractColor, reflectColor, fresnel);
    //Mélange la couleur trouvée avec la couleur globale, selon le taux de Metalness.
    gAlbedo = mix(gAlbedo, mixed, Metal);
}
```

Chapitre 3

Difficultés rencontrées

— Je n'ai pas réussi à mettre en place à temps un Octree prenant en compte des bounding box, ni à implémenter de la physique de collision.

— L'ajout d'une goutte d'eau dans le même shader que les calculs emmenés l'eau à prendre des valeurs absurde et à monter dans le ciel très vite, j'ai donc du séparer l'ajout de gouttelette et le calcul de la physique de l'eau dans deux shaders différents.

— Lors de la première tentative d'implémentation du SSR (Screen Space Reflexion), la méthode que j'avais suivi demandais beaucoup de ressource, assez pour causer un arrêt de la carte graphique, la nouvelle méthode simplifiée est bien moins gourmande, même si moins précise.

— Le système de masse-ressort, qui suit un point sur la Spline à tendance à sortir de sa trajectoire et même de l'aquarium si le mouvement est trop rapide.

— Le rendu différé puis plaqué sur un simple quad pour l'affichage à empêcher toutes interactions simple de lancé de rayon, j'ai donc décidé de définir une caméra de rendu principal par défaut.

Chapitre 4

Références

— OpenGameArt, https://opengameart.org/,consulté en ligne.

— LearnOpenGL, https://learnopengl.com/, consulté en ligne.

— LENGYEL Eric, *Mathematics for 3D Game Programming and Computer Graphics*, 2002.

— LETTIER David, "3D Game Shaders For Beginners", https://lettier.github.io/3d-game-shaders-for-beginners/screen-space-refraction.html, 2019, consulté en ligne.

— SZAUER Gabor, *Game Physics Cookbook*, 2017.

— WALLACE Evan, "WebGL Water", https://madebyevan.com/webgl-water/, consulté en ligne.