Université Claude Bernard Lyon 1

Le monde

Groupe : Eikon Crafters

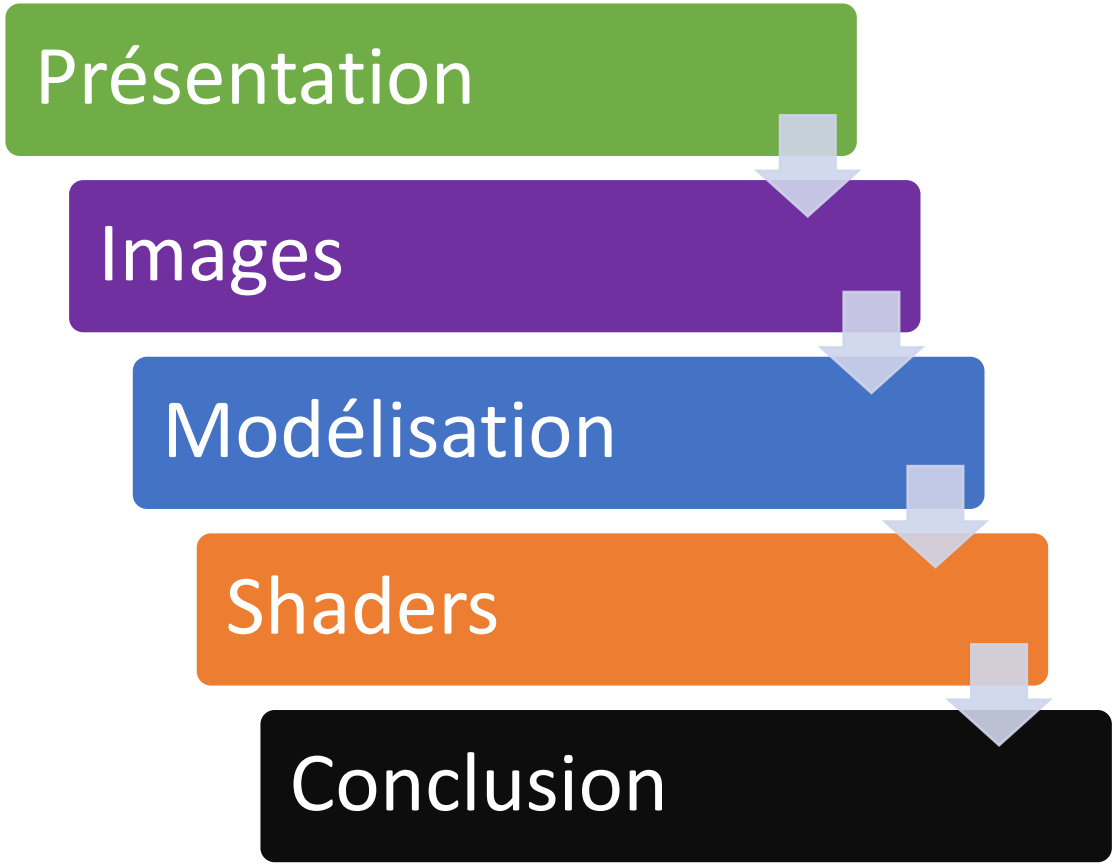
Theme : AM5

SAID KARIM Wassila p2100663

MECHICHE Nessim p2004503

Encadrant: Alexandre MEYER

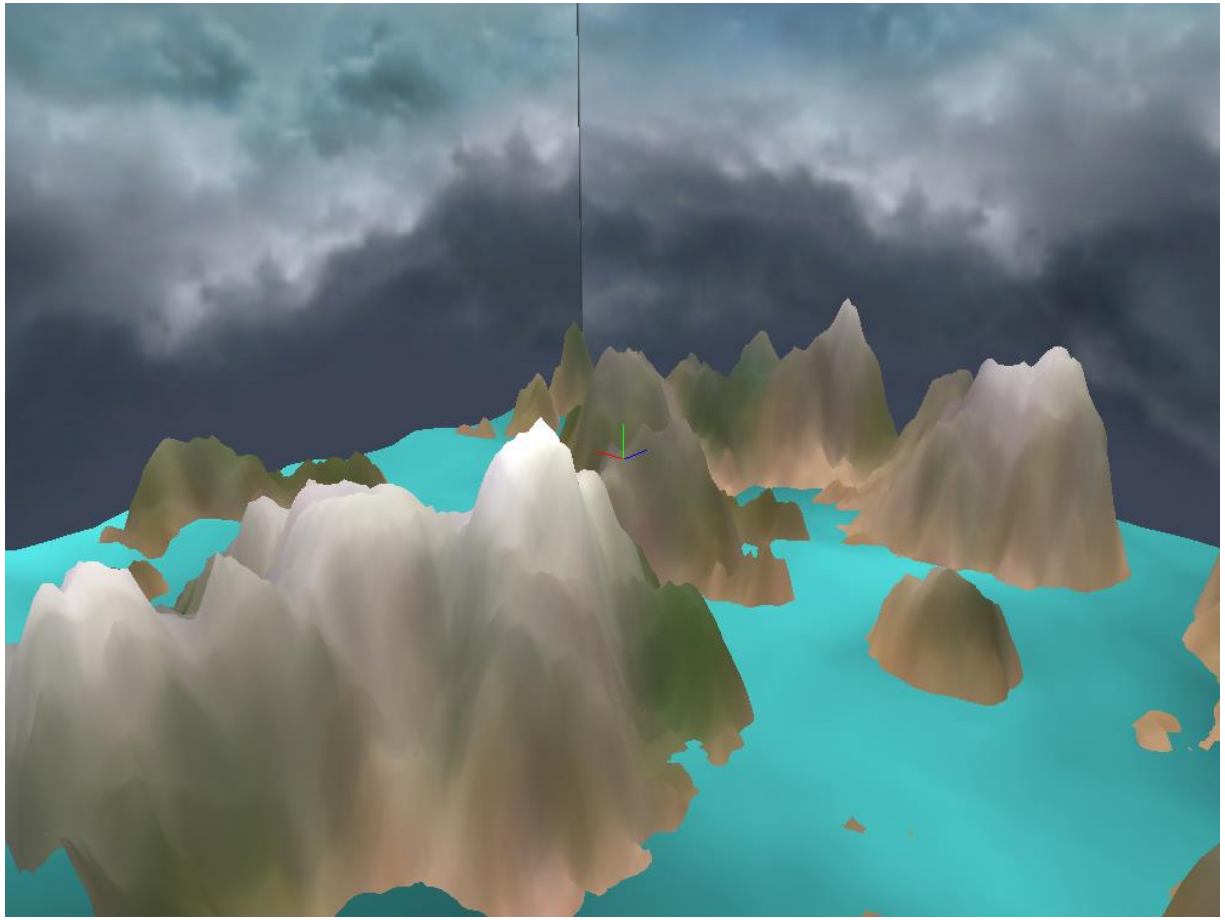
Table des matières



Modéliser une surface d’eau:

- Mouvement/Animation (physique)
- Lumière
- L’affichage

Principes



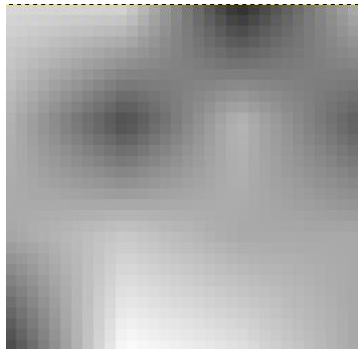
Objectifs

- Individuel :
  - Apprendre un nouveau langage(glsl)
  - Comprendre le concept de la modélisation
  - Savoir présenter/restituer son travail

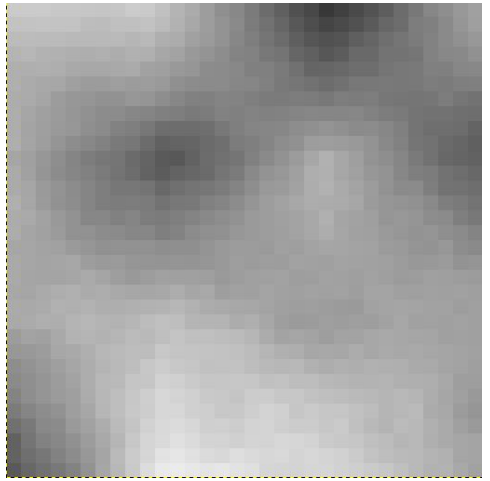
- Projet :
  - Travailler en groupe
  - Acquérir des compétences par soi-même
  - Mener a bien un vrai projet

9 Images généré
« aléatoirement »
3 fonctions principales de Image sont :
Bruit( );
GetPixel(Coordonné x, y);
Concat(Image 1 , 2 ,3 ,4);

image_originel.png



image_aggrandit.png



Final9.png

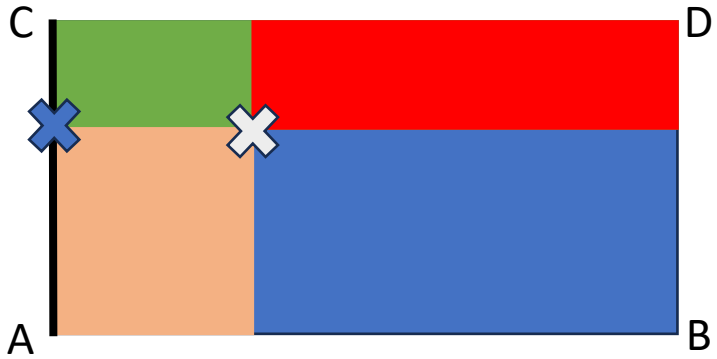
Images

```
void Image::concat(const Image& im4, const Image& im1, const Image& im2,
                   const Image& im3) {
  for (unsigned int i = 0; i < dimx; i++) {
    for (unsigned int j = 0; j < dimy; j++) {
      unsigned char rouge =
          (unsigned char)(((int)im4.getPix(i, j).getRouge() * 0.85 +
                           (int)im1.getPix(i, j).getRouge() * 0.08 +
                           (int)im2.getPix(i, j).getRouge() * 0.045 +
                           (int)im3.getPix(i, j).getRouge() * 0.025));
      setPix(i, j, Pixel(rouge, rouge, rouge));
    }
  }
}
```

: 5-6 semaines

GetPixel(Coordonné x, y);
Interpolation bilinéaire
Taille normalisée
Pourcentage
Génération naturelle

Bruit de Perlin



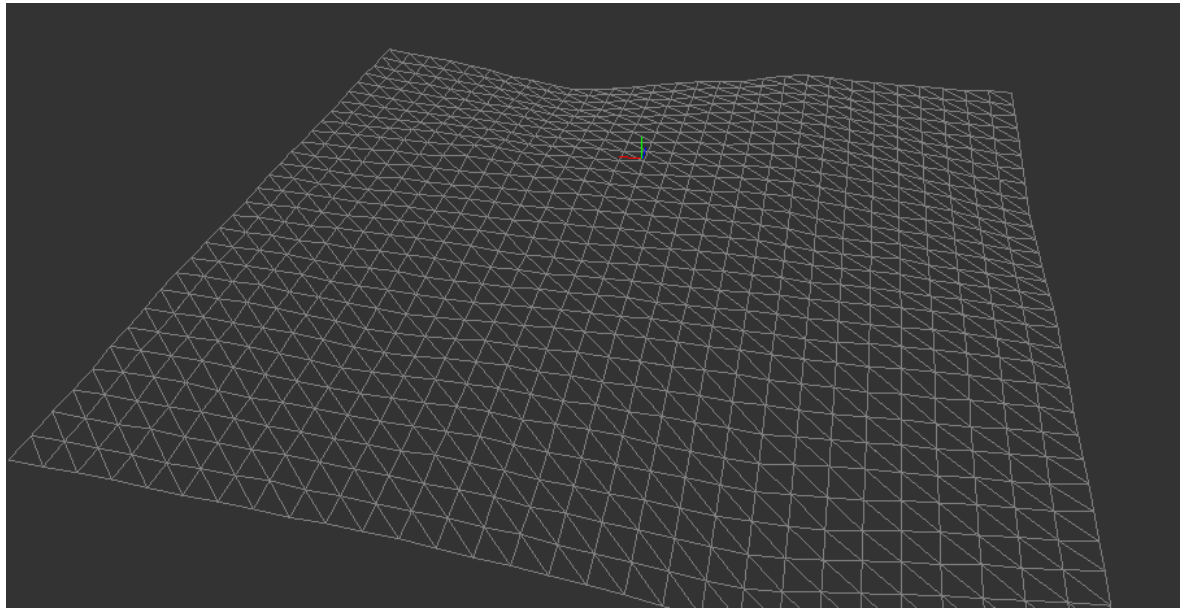

```
const Pixel Image::getPixel(float x, float y) const {
    assert( x>=0.f);
    assert( y>=0.f);
    assert( x<=1.f);
    assert( y<=1.f);
    float x_R = x*(dimx-1);
    float y_R = y*(dimy-1);
    int x_partie_entiere = (int) x_R;
    int y_partie_entiere = (int) y_R;
    float point_A, point_B, point_C, point_D;
    point_A=getPix(x_partie_entiere,y_partie_entiere).getBleu();
    if (y_partie_entiere>=dimy-1)
      point_B = point_A;
    else
      point_B=getPix(x_partie_entiere,y_partie_entiere+1).getBleu();
    if (x_partie_entiere>=dimx-1 || y_partie_entiere>=dimy-1)
      point_C = point_A;
    else
      point_C=getPix(x_partie_entiere+1,y_partie_entiere+1).getBleu();
    if (x_partie_entiere>=dimx-1)
      point_D = point_A;
    else
      point_D=getPix(x_partie_entiere+1,y_partie_entiere).getBleu();
    float x_partie_quotien = x_R - (float) x_partie_entiere;
    float y_partie_quotien = y_R - (float) y_partie_entiere;
    float point_E = point_D*x_partie_quotien + point_A*(1-x_partie_quotien);
    float point_F = point_C*x_partie_quotien + point_B*(1-x_partie_quotien);
    float valeur_inter = point_F*y_partie_quotien + point_E*(1-y_partie_quotien);
    return Pixel((unsigned char)valeur_inter,(unsigned char)valeur_inter,(unsigned char)valeur_inter);
}
```

: 4 semaines

Image

- Init
- Mesh
- Vertex
- Image

Modélisation



```
// Initialise un terrain a partir d'une image pour un mesh en particulier
void ViewerEtudiant::init_terrain(const Image& im, Mesh& m_Objet){
    m_Objet = Mesh(GL_TRIANGLE_STRIP);

    for(int i=1;i<im.width()-2;++i){ // Boucle sur les i
        for(int j=1;j<im.height()-1;++j){ // Boucle sur les j
            m_Objet.normal( terrainNormal(im, i+1, j) );
            m_Objet.texcoord(float(i+1)/float(im.width()),float(j)/float(im.height()));
            m_Objet.vertex( Point(i+1, 2.f*im(i+1, j).r, j) );

            m_Objet.normal( terrainNormal(im, i, j) );
            m_Objet.texcoord(float(i)/float(im.width()),float(j)/float(im.height()));
            m_Objet.vertex( Point(i, 2.f*im(i, j).r, j) );
        }
        m_Objet.restart_strip(); // Affichage en triangle_strip par bande
    }
}
```

: 1-2 semaines

Préparation
Read image
Read texture

Init terrain
Read program => l'affichage

MVP:
Model: Transformation physique
Vu : fenetre
Projection: camera

Uniform

: ~3 semaines

Init()

```
// Appel des images qui serviront de texture et de modeler les objets
m_surface_Alti = read_image("data/terrain/final5.png");
m_surface_texture = read_texture(0, smart_path("data/terrain/final5.png"));
m_terrainAlti= read_image("data/terrain/terrain.png");
m_terrain_texture= read_texture(0,"data/terrain/terrain_texture.png");
m_texture_Decor=read_texture(0,"data/decor.png");

// Initialisations des objets
ViewerEtudiant:: init_cube_map_Decor();
ViewerEtudiant:: init_terrain(m_surface_Alti, m_surface_Eau);
ViewerEtudiant:: init_terrain(m_terrainAlti, m_terrain);

// Appel des programes shaders
m_program_Eau= read_program("data/shaders/surface_Eau.glsl");
m_program_Terrain= read_program("data/shaders/terrain.glsl");
m_program_decor= read_program("data/shaders/cube_map.glsl");
```

Draw()

```
//  parametrer le shader program m_program_Eau
glUseProgram(m_program_Eau);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_surface_texture);
glBindSampler(0, sampler);
program_uniform(m_program_Eau, "mvpMatrix", mvp);
program_uniform(m_program_Eau, "view", view);
GLfloat time= glGetUniformLocation(m_program_Eau, "time");
glUniform1f(time, float(global_time()));
GLuint poslight=glGetUniformLocation(m_program_Eau, "lightCol");
glUniform3f(poslight,lightCol.x,lightCol.y,lightCol.z);

glDrawArrays(GL_TRIANGLES, 0, vertex_count_eau);
m_surface_Eau.draw(m_program_Eau, /* use position */ true, /* use texcoord */ true, /* use normal */ true,
/* use color */ true, /* use material index*/ false);
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
glBindSampler(0, 0);
glUseProgram(0);
glBindVertexArray(0);
```

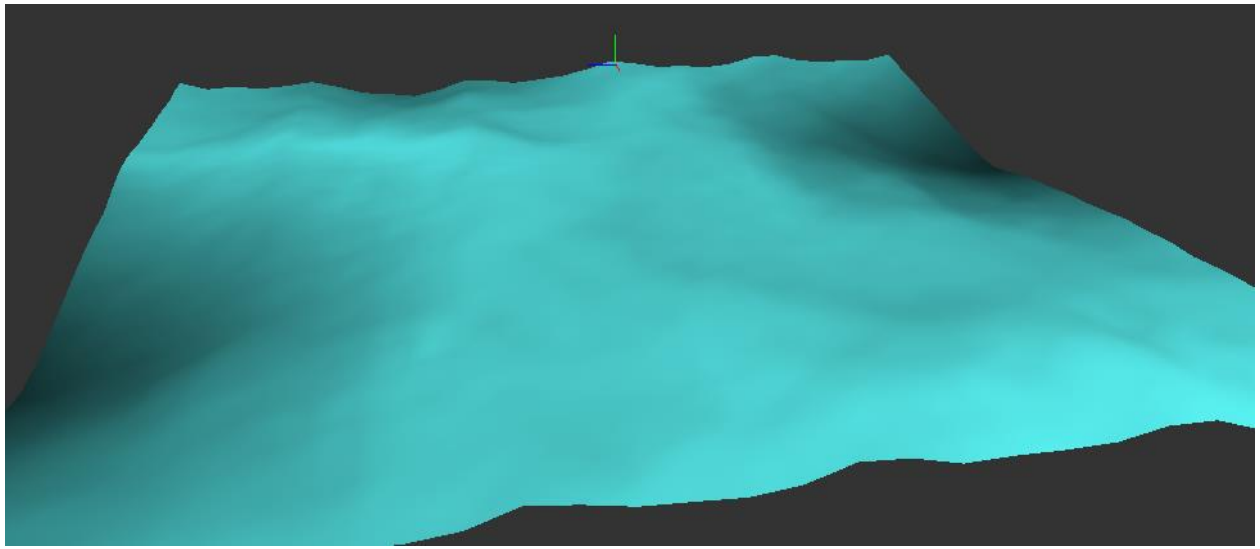
L’apparence de l’objet

Vertex shader
-
Fragment shader
- couleur
- sa réaction avec la lumiere

```
#version 330
#pragma debug(on)
#ifdef VERTEX_SHADER
layout(location = 0) in vec3 position;
layout(location = 1) in vec2 texcoord;
layout(location = 2) in vec3 normal;

uniform mat4 mvpMatrix;
uniform float time;
out vec2 intexcoord;
out vec3 inormal;
out vec3 FragPos;

void main()
{
    vec4 pos = vec4(position, 1);
    intexcoord = texcoord;
    float timePeriod = mod(time,10000000);
    pos = pos + vec4(0, min(sin(3.14*(texcoord.y+0.01)* 2.0 * (timePeriod/(600*3.14))),
    cos(3.14*(texcoord.y+0.01)* 2.0 * (timePeriod/(600*3.14))) )/7, 0, 0);
    gl_Position = mvpMatrix * pos;
    FragPos = vec3(0, 5, 0);
    inormal = normal;
}
#endif
```

```
void main()
{
    vec3 lightPos = vec3(0, 7, 0);
    vec3 norm = normalize(inormal);
    vec3 objectColor = vec3(0.33, 0.9  , 0.90);
    float ambientStrength =2.5;
    float distance = length(lightPos - FragPos)*0.75;
    float attenuation = 1.0 / (distance * distance + 0.1);
    vec3 ambient = ambientStrength * lightCol * attenuation;
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightCol * attenuation;
    // Réflexion spéculaire
    vec3 viewDir = normalize(view - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularColor * spec;
    vec3 result = (ambient + diffuse + specular) * objectColor;
    fragment_color = texture(terrain, intexcoord) * vec4(result, 1.0);
}
#endif
```

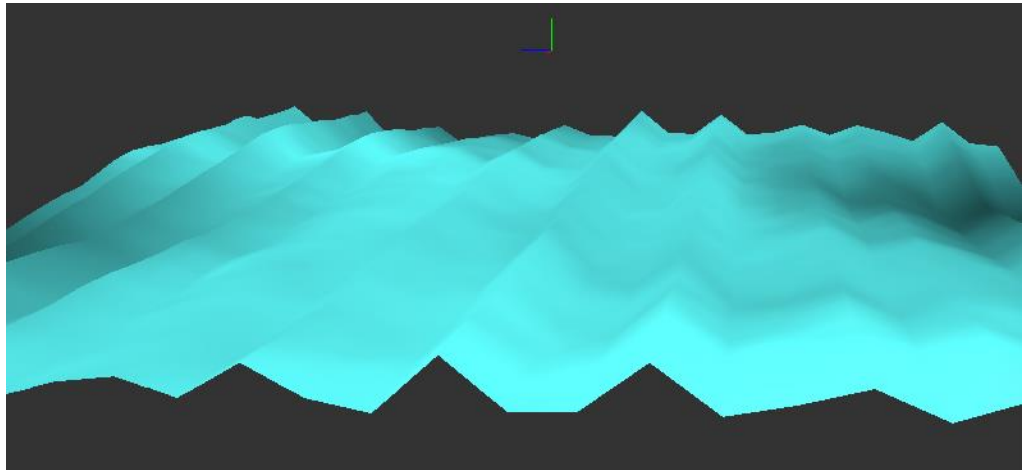
Shader



: 5-6 semaines

- Cos & Sin
- TimePeriod
- Pos

Mouvement



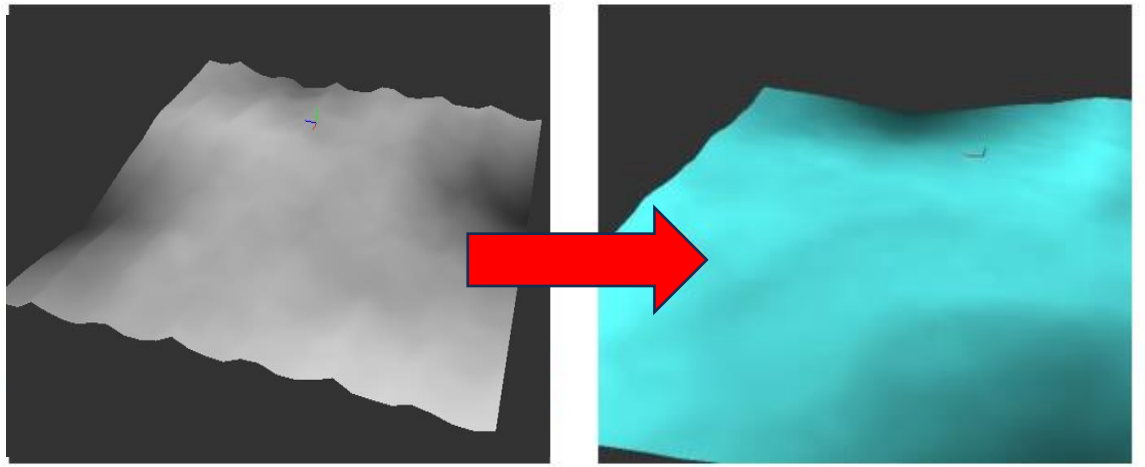
```
#version 330
#pragma debug(on)
#ifdef VERTEX_SHADER
layout(location = 0) in vec3 position;
layout(location = 1) in vec2 texcoord;
layout(location = 2) in vec3 normal;

uniform mat4 mvpMatrix;
uniform float time;
out vec2 intexcoord;
out vec3 inormal;
out vec3 FragPos;

void main()
{
    vec4 pos = vec4(position, 1);
    intexcoord = texcoord;
    float timePeriod = mod(time,10000000);
    pos = pos + vec4(0, min(sin(3.14*(texcoord.y+0.01)* 2.0 * (timePeriod/(600*3.14))),
    cos(3.14*(texcoord.y+0.01)* 2.0 * (timePeriod/(600*3.14))) )/7, 0, 0);
    gl_Position = mvpMatrix * pos;
    FragPos = vec3(0, 5, 0);
    inormal = normal;
}
#endif
```

: 4 semaines

- Couleur de l’objet
- Ambient :
  - lightPos
  - Distance
  - Attenuation
- Speculaire
- Reflective

Lumière

: 2-3 semaines



```
void main()
{
    vec3 lightPos = vec3(0, 7, 0);
    vec3 norm = normalize(inormal);
    vec3 objectColor = vec3(0.33, 0.9  , 0.90);
    float ambientStrength =2.5;
    float distance = length(lightPos - FragPos)*0.75;
    float attenuation = 1.0 / (distance * distance + 0.1);
    vec3 ambient = ambientStrength * lightCol * attenuation;
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightCol * attenuation;
    // Réflexion spéculaire
    vec3 viewDir = normalize(view - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularColor * spec;
    vec3 result = (ambient + diffuse + specular) * objectColor;
    fragment_color = texture(terrain, intexcoord) * vec4(result, 1.0);
}
#endif
```

Difficultés

- Choix de l'interpolation
- Formes de la vague
- Lumière
- Tableau ou pas tableau?
- Interprétation de la documentation non à jour
- Nom du projet

Conclusion

Gestion du Temps et du Travail

domaine de la modélisation 3D et **GLSL**

Combinaison et mise en commun des connaissances

Interprétation de la Documentation

Recherche de ressource sur le web

Débrouillardise

Index

Lien vers le Git :
https://forge.univ-lyon1.fr/p2004503/eikon-crafters/

Video youtube utilisé :
https://youtu.be/Qj_tK_mdRcA?si=Wj2RmfmYDRomMAZT

Liens utiles :
https://theses.hal.science/tel-00319974/file/defense.pdf

ChatGPT :
https://chat.openai.com

La documentation de Gkit :
https://perso.univ-lyon1.fr/jean-claude.iehl/Public/educ/M1IMAGE/html/index.html

code source :
https://perso.liris.cnrs.fr/florence.zara/Web/LIFGraphique.html