



Rendu 3D de l'eau (C++, OpenGL)

HAI928I - Projet 3D

Lien vers le code: <https://github.com/Adeleimpa/Aquagraphics>

-> Le git contient: le code, la vidéo de démonstration, les slides, le rapport, le readme, la vidéo qui montre le résultat sur les PCs de la fac.

1. README

How to run code ? (assuming you have everything to use OpenGL)

I run it on Ubuntu.

OpenGL version string: 4.0 (Compatibility profile) Mesa 22.2.5

Cmake version 3.22.1

1. Download code on your computer
3. Open terminal window in folder 'Code' of Aquagraphics
4. Remove 'build' folder
5. mkdir build
6. cd build
7. cmake ..
8. cd ..

Once you've done that you can you compile and run like that:

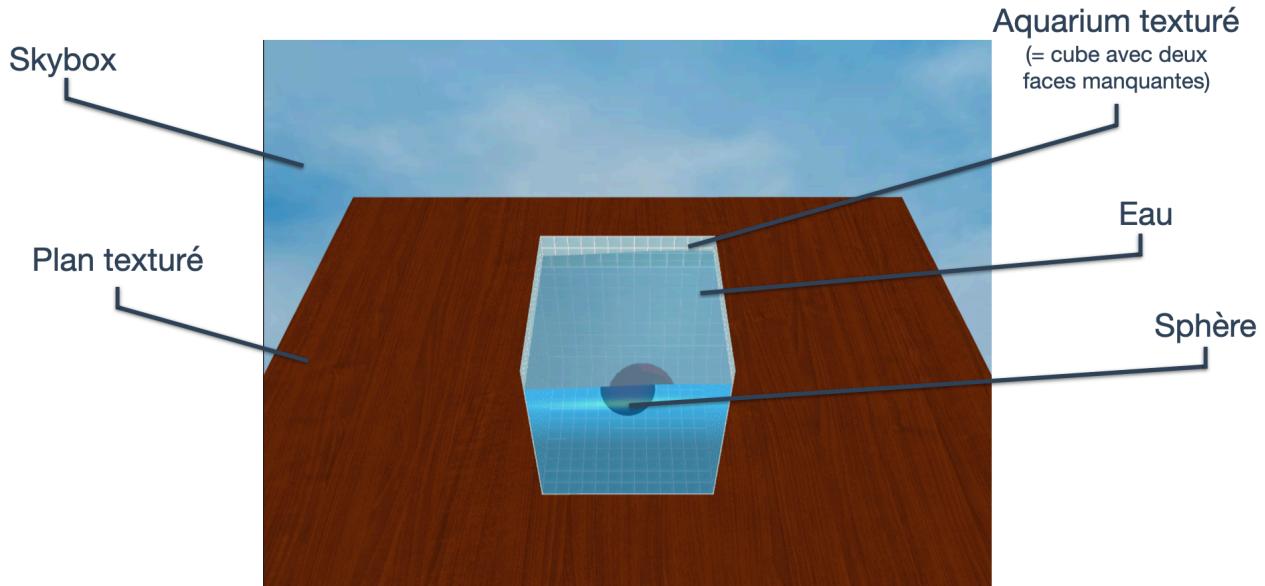
9. cmake --build build
10. cd waterRendering
11. ./waterRendering

Commands in the app:

- Press c to move the camera around the scene
- Press space bar to make a droplet fall onto the surface
- Use the arrows to move in the scene
- Press W to visualize the wires of the mesh

Important: mon code compile sur les PCs du bat16 mais il y a un léger bug qui empêche d'utiliser les flèches pour se déplacer dans la scène et qui fait buger la surface de l'eau (impossible de savoir pourquoi). Vous trouverez dans le git une vidéo qui montre la différence entre la démo et ce qu'il se passe sur les PCs de la fac. Normalement la vidéo de démonstration est suffisante pour visualiser les résultats de ce projet. J'espère sincèrement que cela suffira.

2. Contenu de la scène



3. Rendu de l'eau

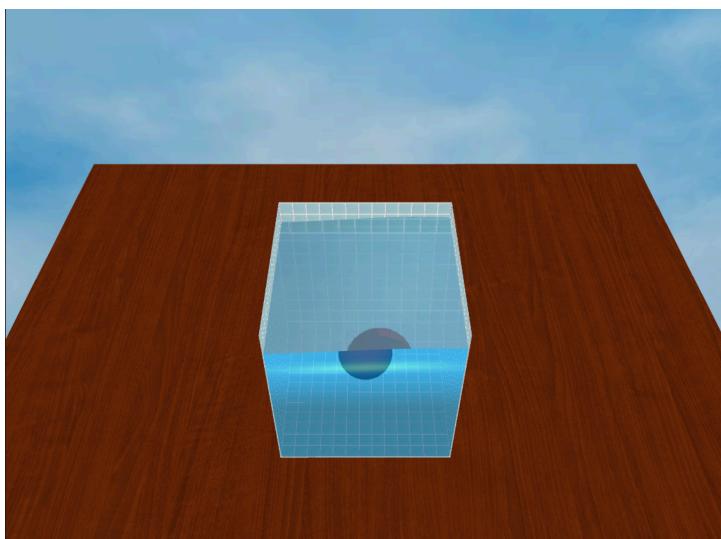
Ce que j'ai implémenté:

- Eclairage de phong (comme vu en cours, directement dans le fragment shader)
- Réflexion (en utilisant une texture liée à un FBO)
- Réfraction (en utilisant une texture liée à un FBO)
- Animation de la surface de l'eau (vague + onde circulaire)

Schéma global pour le rendu de l'eau:

1. Rendu de la scène selon la caméra de réflexion (en utilisant un clipping plane)
2. Rendu de la scène selon la caméra de réfraction (en utilisant un clipping plane)
3. Rendu de la scène dans le default buffer en utilisant les textures de réflexion et de refraction qu'on plaque à la surface de l'eau

Résultat:



+ aperçu de l'onde à la surface



4. Fragment shader

```
#version 330 core

// Output data
out vec4 FragColor;

in vec2 coord_txt;

uniform sampler2D skybox_txt;
uniform int isSkybox;
uniform sampler2D wood_txt;
uniform int isPlane;
uniform int isAquarium;
uniform sampler2D tile_txt;
uniform sampler2D texture_nrs;

uniform int isSphere;

uniform vec3 objectColor;
uniform vec3 k_a;
uniform vec3 k_d;
uniform vec3 k_s;
uniform float transparency;

// light data
uniform vec3 lightPos;
uniform vec3 I_a;
uniform vec3 I_d;
uniform vec3 I_s;

// camera position
uniform vec3 viewPos;

in vec3 fragNormal;
in vec3 FragPos;

// reflection, refraction
uniform sampler2D reflectionTexture;
uniform sampler2D refractionTexture;
uniform int isWater;
in vec4 clipSpace;

// receive height of water from vertex shader
in float height_water;

void main() {
    // light calculations
    vec3 ambient = I_a * k_a;

    vec3 norm = normalize(fragNormal);
    vec3 lightDir = normalize(lightPos - FragPos);

    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * I_d * k_d;

    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = I_s * spec * k_s;

    //vec3 result = ambient * objectColor;
    //vec3 result = (ambient + diffuse) * objectColor;
    vec3 result = (ambient + diffuse + specular) * objectColor;

    // refraction, reflection
    vec2 ndc = (clipSpace.xy / clipSpace.w) / 2.0 + 0.5; // normalized device coords
    vec2 reflectionTxtCoordinates = vec2(ndc.x + 0.12, -ndc.y + 0.12);
    vec2 refractionTxtCoordinates = vec2(ndc.x + 0.12, ndc.y + 0.12);

    if(isSkybox == 1){
        FragColor = texture(skybox_txt, coord_txt) * vec4(objectColor, 0.0);
    } else if (isPlane == 1){
        FragColor = texture(wood_txt, coord_txt) * vec4(objectColor, 0.0);
    } else if(isAquarium == 1){
        FragColor = texture(tile_txt, coord_txt) * vec4(objectColor, 0.0);
    } else if (isWater == 1){
        vec3 upDirection = vec3(0.0, 1.0, 0.0); // direction points to top

        if (normalize(fragNormal) == upDirection) { // if normal points to top (i.e. is top face)
            //mix both reflection and refraction
            vec4 reflectionColor = texture(reflectionTexture, reflectionTxtCoordinates) * vec4(1.0, 1.0, 1.0, 0.0);
            vec4 refractionColor = texture(refractionTexture, refractionTxtCoordinates) * vec4(1.0, 1.0, 1.0, 0.0);

            vec4 mix_refr_refl = mix(reflectionColor, refractionColor, 0.3);
            FragColor = mix_refr_refl * vec4(objectColor, 0.0);
            FragColor = height_water * FragColor;

        } else {
            FragColor = vec4(result, transparency); // Apply object color
        }
    } else{
        FragColor = vec4(result, transparency); // phong color
    }
}
```

5. Vertex shader

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertices_position_modelspace;

// Input normal data
layout (location = 1) in vec3 vertexNormal;

// uniform transformations matrices Model View Projection
// Values that stay constant for the whole mesh.
uniform mat4 model_matrix;
uniform mat4 view_matrix; // camera
uniform mat4 proj_matrix;

mat4 transformation_matrix;

// texture
out vec2 coord_txt;
layout(location = 2) in vec2 coord;

// send normals to fragment shader
out vec3 fragNormal;

// send fragment position to fragment shader
out vec3 FragPos;

// clip space coordinates for refraction
out vec4 clipSpace;

uniform vec4 clipping_plane;

// send height of water to fragment shader
out float height_water;

void main(){
    transformation_matrix = proj_matrix * view_matrix * model_matrix; // MVP but inverted! (order matters)
    gl_ClipDistance[0] = dot(vec4(vertices_position_modelspace, 1.0), clipping_plane);
    clipSpace = transformation_matrix * vec4(vertices_position_modelspace, 1.0);
    gl_Position = clipSpace;
    FragPos = vec3(model_matrix * vec4(vertices_position_modelspace, 1.0));
    fragNormal = vertexNormal;
    coord_txt = coord; // texture
    height_water = vertices_position_modelspace.y;
}
```

6. Perspectives

Ce que j'aurais aimé avoir dans mon résultat final et qui selon moi n'est pas 100% abouti:

- Que la goutte qui tombe et produit une onde puisse tomber n'importe où dans l'aquarium plutôt qu'uniquement au milieu
- Que l'onde produite par la goutte puisse rebondir sur les parois de l'aquarium
- Que plusieurs gouttes puissent tomber en même temps en faisant les ondes se superposer
- Pouvoir déplacer la sphère dans l'aquarium et qu'elle produise une vague à la surface
- Ajouter les caustiques, ç'aurait rendu vraiment trop bien

7. Difficultés rencontrées

- L'utilisation des Frame Buffer Objects pour la réflexion et la réfraction
- Ne pas pouvoir poser de questions aux professeurs pendant les vacances