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

HAI928I - Projet 3D

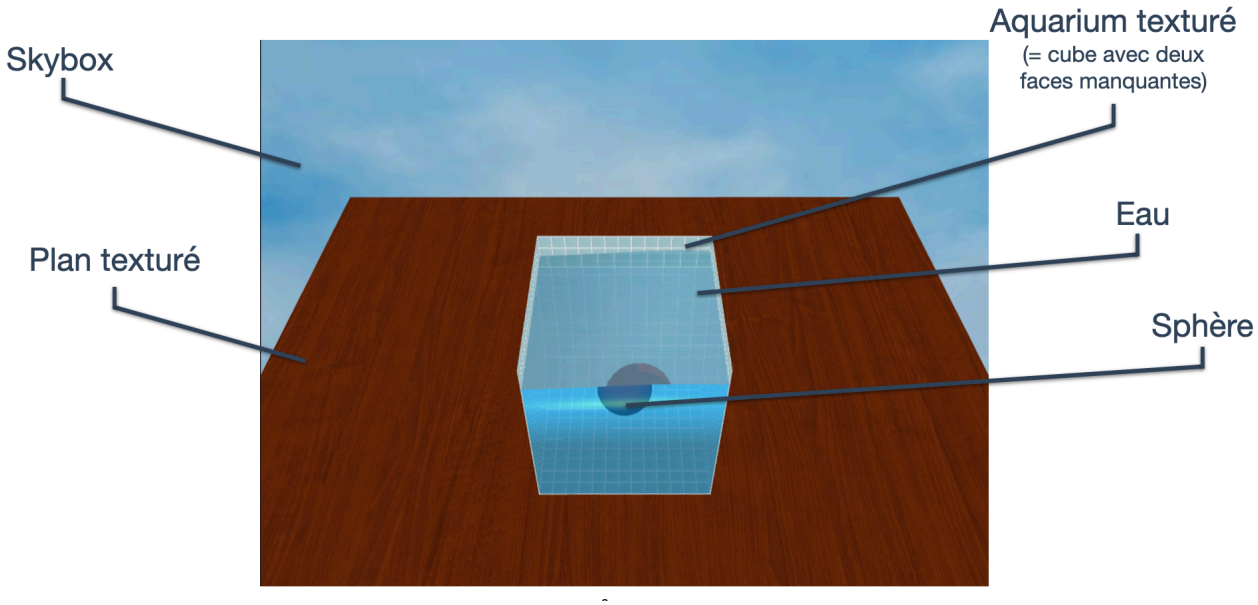
Lien vers le code: https://github.com/Adeleimpa/Aquagraphics
-> Le git contient: le code, la video de démonstration, les slides, le rapport, le readme, la vidéo qui montre le résultat sur les PCs de la fac.

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

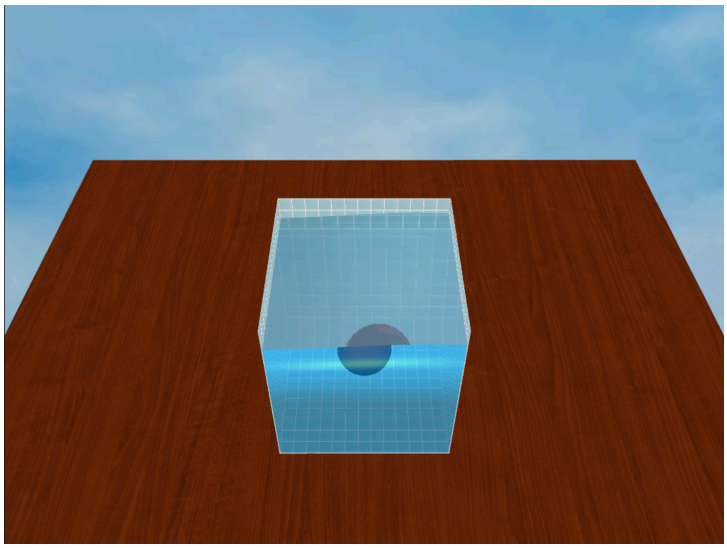


3. Rendu de l'eau

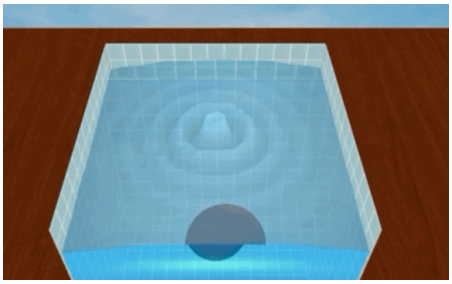
Ce que j'ai implémenté:
- Eclairage de phong (comme vu en cours, directement dans le fragment shader)
- Réflexion (en utilisant une texture liée à un FBO)
- Réfraction (en utilisant une texture liée à un FBO)
- Animation de la surface de l'eau (vague + onde circulaire)

Schéma global pour le rendu de l'eau:
1. Rendu de la scène selon la caméra de reflexion (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 reflexion 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

// Ouput 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

Aquagraphics

▼ Ressources

- Water rendering en C++ et OpenGL (raytraced reflection & refraction)

GitHub - realkushagrakhare/ProjectWater: Realistic water rendering using shaders in OpenGL. Inspired from Evan Wallace's WebGL water rendering.

Realistic water rendering using shaders in OpenGL. Inspired from Evan Wallace's WebGL water rendering. - GitHub - realkushagrakhare/ProjectWater: Realistic water rendering using shaders in ...

https://github.com/realkushagrakhare/ProjectWater

- Water simulation en C et GLSL (gpu)

GitHub - MauriceGit/Water_Simulation: Water-Simulation with real time specular reflection on the waters surface. The reflection is implemented in GL

Water-Simulation with real time specular reflection on the waters surface. The reflection is implemented in GLSL and runs on the GPU and in screen space. The water its

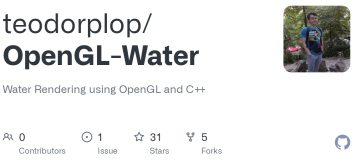
https://github.com/MauriceGit/Water_Simulation

- Water rendering C++ et OpenGL

GitHub - teodorplop/OpenGL-Water: Water Rendering using OpenGL and C++

Water Rendering using OpenGL and C++. Contribute to teodorplop/OpenGL-Water development by creating an account on GitHub.

https://github.com/teodorplop/OpenGL-Water



- Evan Wallace le boss

WebGL Water

Made by Evan Wallace

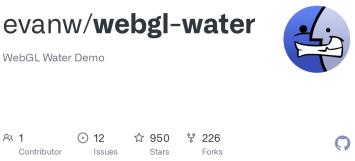
https://madebyevan.com/webgl-water/

- Evan Wallace github

GitHub - evanw/webgl-water at 73eda8be832b649367b25ea5690c1f0181bb56ad

WebGL Water Demo. Contribute to evanw/webgl-water development by creating an account on GitHub.

https://github.com/evanw/webgl-water/tree/73eda8be832b649367b25ea5690c1f0181bb56ad

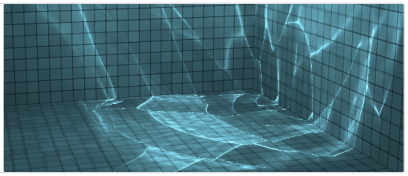


- Les Caustiques par Evan Wallace

Rendering Realtime Caustics in WebGL

I created http://madebyevan.com/webgl-water/ back in 2011 and I've always been meaning to explain how it works. The most interesting aspect…

https://medium.com/@evanwallace/rendering-realtime-caustics-in-webgl-2a99a29a0b2c



- Article "Real time rendering optical effects of water" (C++, OpenGL, GLSL)

https://www.cs.umd.edu/~mount/Indep/Aharon_Turpie/final-rept.pdf

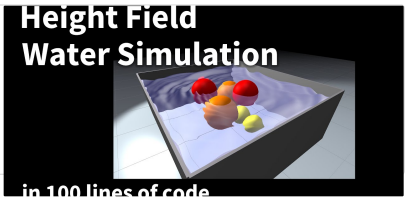
- Video Youtube

20 - How to write a Height-Field Water Simulator with 100 lines of code.

In this tutorial I explain how to simulate water as a height field and its two-way interaction with solid objects.

The demo:

https://www.youtube.com/watch?v=hswBi5wcqAA&list=PLMlwtHzmNCipW34oJ-vxAB3BGeR37l-Gm&index=5

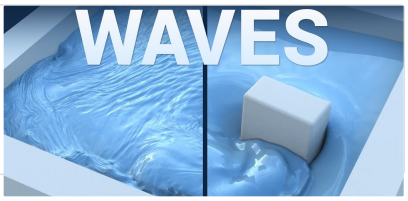


- Video Youtube

Beautiful Fluid Simulations...In Just 40 Seconds! 🤯

❤️ Check out Weights & Biases and sign up for a free demo here: https://wandb.com/papers
❤️ Their mentioned post is available here: https://wandb.ai/wandb/getting-started/reports/Visualize-Debug-Machine-Learning-Models--VmlldzoyNzY5MDk?utm_source=karoly#System-4

https://www.youtube.com/watch?v=LtyvS7NYonw&list=PLMlwtHzmNCipW34oJ-vxAB3BGeR37l-Gm&index=6



- Youtube Playlist de OpenGL Water tutorials

OpenGL Water Tutorials

https://www.youtube.com/playlist?list=PLRIWtICgwaX23jiqVByUs0bqhnalNTNZh

- OpenGL Tutorial Water Waves GPU algorithm

3D C/C++ tutorials - OpenGL 2.1 - Water waves GPU algorithm

http://www.3dcpptutorials.sk/index.php?id=48

- Interactive water surface, light reflection and refraction + caustics

3D C/C++ tutorials - OpenGL 2.1 - Interactive water surface, light reflection and refraction, caustic

First, the water surface 512x512 normal-bump-map (NBM) RGBA32F texture is created using a wave function in the fragment shader. The surface normal is calculated as an optimized cross product of 4 neighboring vertices and is stored in the rgb channels. The surface height (y value) is stored in the alpha channel. The light reflection and refraction effects are a bit tricky. There is one reflection, one refraction and one depth texture. All of them are the size of the screen. The reflection

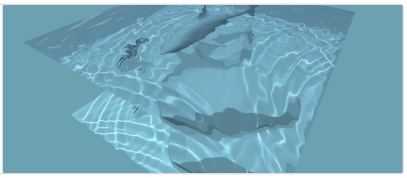
http://www.3dcpptutorials.sk/index.php?id=43

- Real time rendering of water caustics

Real-time rendering of water caustics

In this article, I present an attempt for generalizing caustics computation in real-time using WebGL and ThreeJS. The fact that it is an…

https://medium.com/@martinRenou/real-time-rendering-of-water-caustics-59cda1d74aa



- Water physics in 100 lines of code

https://github.com/matthias-research/pages/blob/master/tenMinutePhysics/20-heightFieldWater.html

- Florentin

GitHub - Flare00/M2-Projet3D-Aquarium: M2 - Projet 3D - Réalisation d'un Aquarium

M2 - Projet 3D - Réalisation d'un Aquarium. Contribute to Flare00/M2-Projet3D-Aquarium development by creating an account on GitHub.

https://github.com/Flare00/M2-Projet3D-Aquarium

- Shaders cookbook

GitHub - PacktPublishing/OpenGL-4-Shading-Language-Cookbook-Third-Edition: OpenGL 4 Shading Language Cookbook - Third Edition, published

OpenGL 4 Shading Language Cookbook - Third Edition, published by Packt - GitHub - PacktPublishing/OpenGL-4-Shading-Language-Cookbook-Third-Edition: OpenGL 4 Shading Language Cookbook - Third Editio...

https://github.com/PacktPublishing/OpenGL-4-Shading-Language-Cookbook-Third-Edition

- Adrien

GitHub - HouleAdrien/Fluids3D-cpp: A cpp fluid simulation that uses the Shallow water equations

A cpp fluid simulation that uses the Shallow water equations - GitHub - HouleAdrien/Fluids3D-cpp: A cpp fluid simulation that uses the Shallow water equations

https://github.com/HouleAdrien/Fluids3D-cpp