Individual Study IV : Cloud Simulation

Objective

Figure 1 : clouds

Clouds are essential elements in both game development and film production for creating natural outdoor environments. In modern game development, clouds are often rendered using pre-computed data or textures. While this approach is efficient and visually appealing, it lacks dynamic realism and real-time interaction. Therefore, I aim to implement a cloud simulation using the ray marching technique in OpenGL to achieve more realistic and procedurally generated cloud visuals.

Method

Research

OpenGL

Figure 2 : OpenGL Logo

OpenGL (Open Graphics Library) is a cross-language, cross-platform API for rendering 2D and 3D graphics.The API is typically used to interact with a graphics processing unit (GPU).

It provides a programmable pipeline that enables developers to write custom vertex and fragment shaders for real-time rendering.

To understand how to use openGL properly, I study from https://learnopengl.com about the basics of OpenGL before implementing cloud rendering.

There are 4 more libraries included in the project to facilitate the developer.

1. **ImGui** : A bloat-free graphical user interface library for C++
2. **GLFW** : A simple API for creating windows, contexts and surfaces, receiving input and events.
3. **Glade** : A multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs.
4. **GLM** : A header-only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications.

Signed Distance Function

Signed Distance Functions (SDF) are mathematical representations to describe how far the point is from the surface of the object. This is crucial in raymarching in the next topic.

In the context of cloud simulation, SDFs can be adapted to represent the boundaries of volumetric density fields, helping to shape cloud volumes.

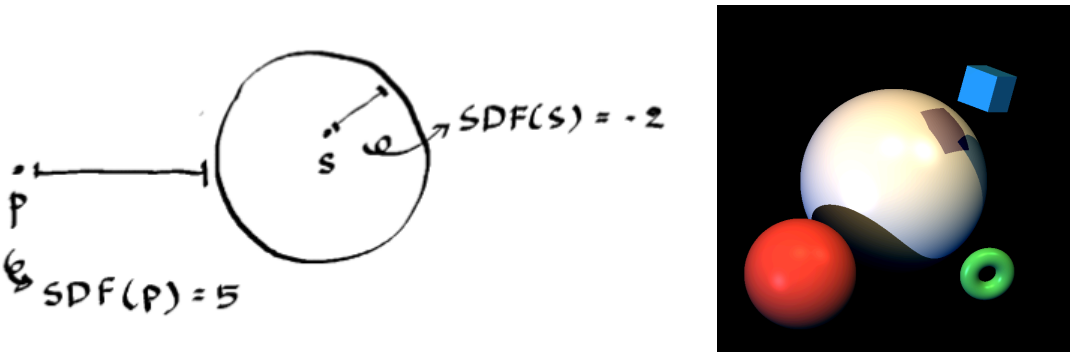

Figure 3,4 : SDF

Raymarching

Ray marching is a rendering technique used for visualizing volumetric data. It works by casting “rays” from the camera into the scene and “marching” through the space with the desired step size, hence the name “raymarching”.

The advantage of SDF can be combined with Ray marching to march more efficiently to the object since SDF can calculate the distance between ray and object’s surface. In addition, I use SDF to march inside the object until the SDF says the ray is out of the object.

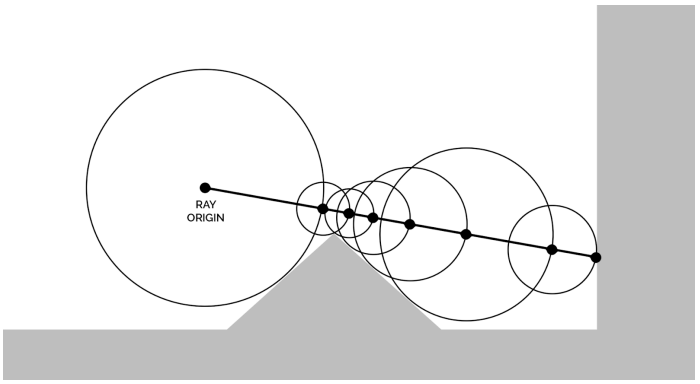

Figure 5 : Ray marching

Fractal Brownian Motion

Fractal Brownian Motion (FBM) is a method for generating noise by combining multiple layers (octaves) of noise at different frequencies and amplitudes. FBM is commonly used to create the organic or fluffy look. It helps define the shape of the cloud volume by varying the density in a natural pattern.

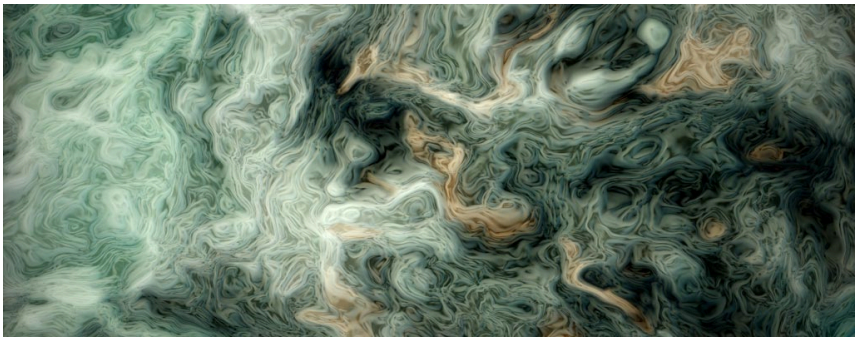

Figure 6 : example of FBM noise

Voronoi noise

Voronoi noise is a type of procedural noise based on the distance to the nearest feature point in a grid. When combined with FBM, it can create fluffy noise, which looks like cloud

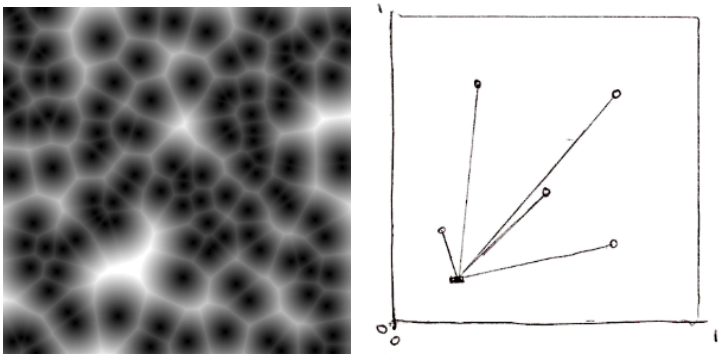

Figure 7,8 : example of Voronoi noise

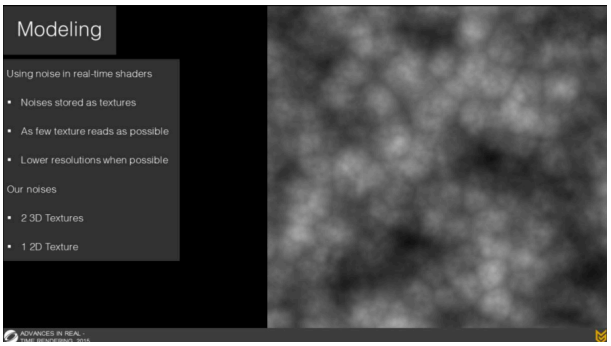

Figure 9 : example of Voronoi noise in FBM

Beers's Law

Beer's Law (or Beer-Lambert Law) is a physical principle describing the attenuation of light as it passes through a medium. In cloud rendering, it helps simulate how light is absorbed and scattered inside the cloud volume, contributing to the soft shadow and light falloff effect that gives clouds their realistic appearance.

$$Beer = e^{-density \cdot absorption}$$

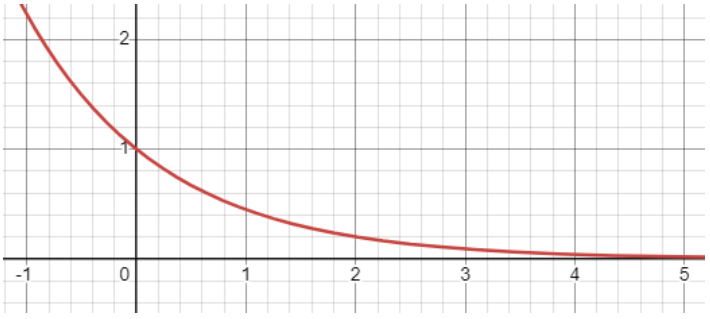

Figure 10 : graph of Beer's equation

Beers-Powder's Law

An extension of Beer's Law, the Beer's-Power Law introduces from "The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn" to simulate where the edge of the cloud can gather the scattered light from surrounding less than the cloud that is further inside.

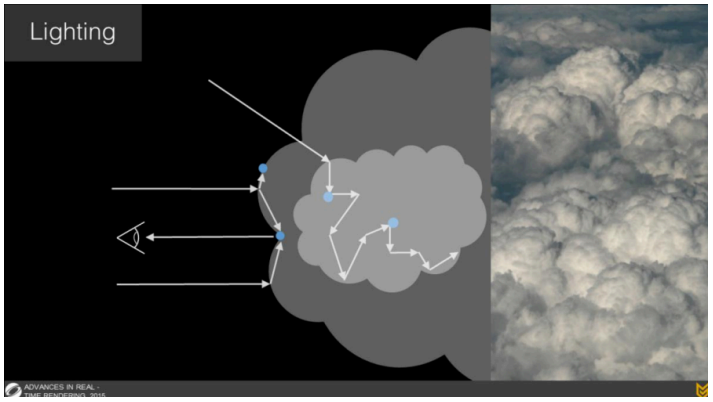

Figure 11 : real-life example of Beer-Powder's Law

$$Powder = 1 - e^{density \cdot 2}$$
$$attenuation = Beers \cdot Powder$$

Figure 12 : graph of Beer-Powder's equation

Implement

Setup project

I create a basic openGL program to start the project with feature like

1. VAO VBO
2. Shader
3. Camera

The object that live in this step is

1. Camera
2. Cube represent as a sun
3. Quad that attach to the camera

Ray marching

In the quad shader, I implement the raymarching with the SDF of the box as a boundary of the cloud.

```
void main(){
    …
    // raymarch
    const float tmax = 10.0;
    float t = 0.0;
    for( int i=0; i<STEP; i++ )
    {
        vec3 pos = ro + t*rd;
        float h = map(pos);
        if( h<min_distance || t>tmax ) break;
        t += h;
    }

    vec3 col = vec3(0.0);
```

```
    float alpha = 0.0;
    if(t<tmax){
        col = vec3(1.0);
        alpha = 1.0;

        vec3 pos = ro + t*rd;

        Cloud(pos, rd, col, alpha);
        col = mix(_buttomColor.xyz, col, length(col)).xyz;
    }

    FragColor = vec4(col,alpha);

}
```

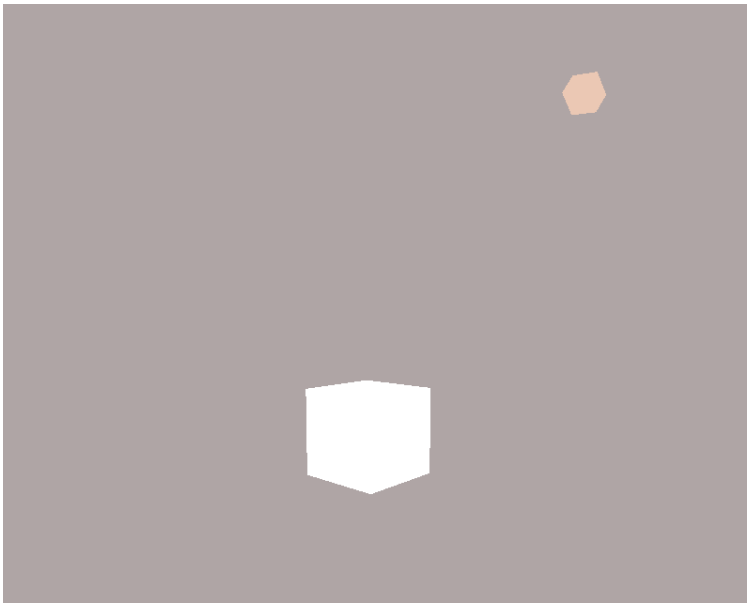

Figure 13 : scene with box from raymarching as cloud boundary(white) and box from traditional rendering as a sun(orange)

Noise

As I mentioned earlier, I use Voronoi noise with FBM. I chose to implement the Voronoi noise in the CPU side as a 3D texture then pass this into quad shader.

quad shade is use noise texture to advance them to FBM noise

```
float sampleNoise(vec3 _pos){
      float value = 0.0;
    float amplitude = _initialFbmAmplitude;
    float st = _noiseSize;
    float octave = _FbmOctave;
```

```
    // Loop of octaves
    for (int i = 0; i < octave; i++) {
            vec3 rp = 1.0 - (_pos- vec3(st))/(st*2.0);
            float t = texture(_noiseSampler, rp).r;
        value += amplitude * t;

        st *= .5;
        amplitude *= .5;
    }
    value = pow(value, _cutTexture) ;
    return smoothstep(0.01, 0.5, value)* value;
}
```

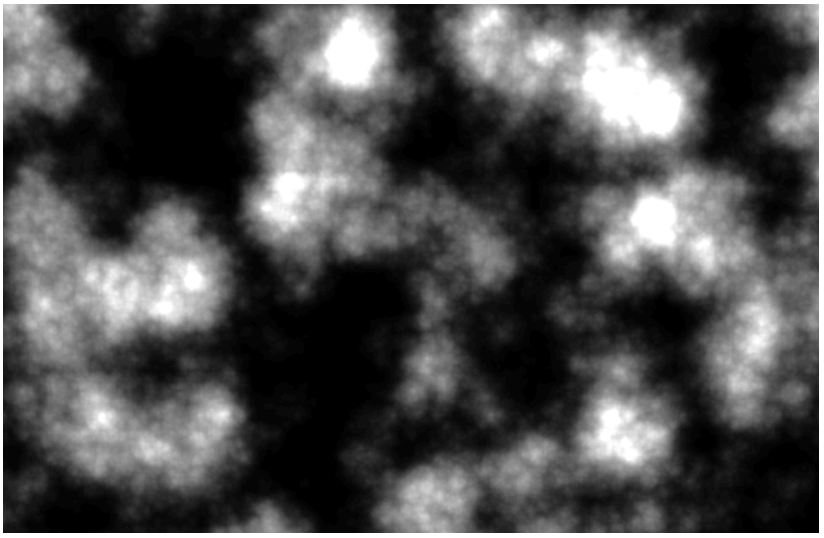

Figure 14 : result of the FBM noise for cloud's density

Cloud

After the ray hits the boundary of the cloud volume, it needs to continue marching inside to sample the cloud's density and accumulate the light that can pass through the volume. This process requires **two ray marching steps**:

1. **Primary ray**: This ray starts from the camera and marches through the cloud. Its main role is to sample the cloud's density and accumulate light along the view direction. This process uses Beer's law to calculate transmittance of each point using totalDensity that accumulates along the way.

```
void Cloud(vec3 ro, vec3 rd, out vec3 col, out float alpha){

    alpha = 1.0;
    col = vec3(0.0);

    float transmittance= 1.0;
```

```
    vec3 lightEnergy = vec3(0.0);

    vec3 lightDir = normalize(lightPosition);

    float t = 0.0;
    for( int i=0; i<STEP; i++ )
    {
        vec3 pos = ro + t*rd;
        float h = map(pos);
        if( h>min_distance) break;

        float density = 0.0;
            density = sampleDensity(pos) ;

        if(density > 0.0){
                  float lightTransmittance = lightMarching(pos); //
light for each point.
            lightEnergy += density * transmittance * lightTransmittance;
            transmittance *= exp(-density * walk_in_distance *
_absorption);
        }
        t += walk_in_distance;
        i++;
    }
    // lightEnergy = clamp(lightEnergy, vec3(0.0), vec3(1.0));
    vec3 cloudCol = lightEnergy * _lightColor;
    col = cloudCol + (_skyColor.xyz * transmittance);
    alpha = 1.0 - transmittance;
}
```

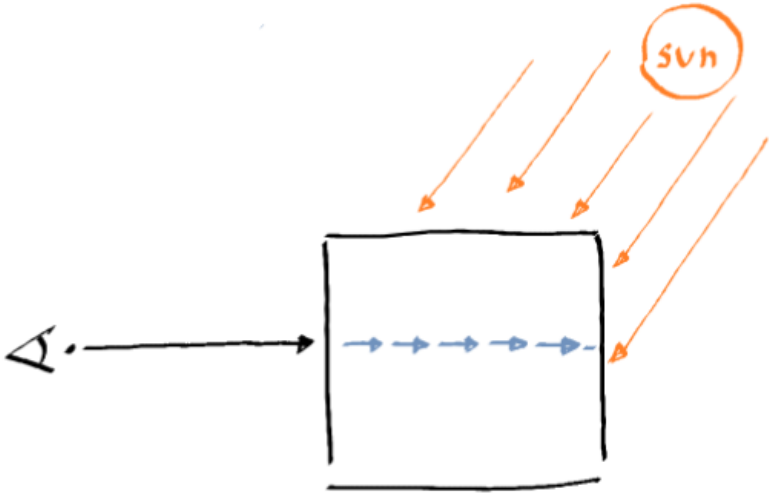

Figure 15 : How primary ray travel

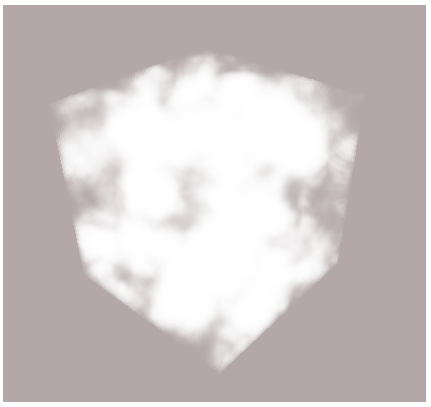

Figure 16 : Cloud with only primary ray

2. **Shadow ray**: At each sample point along the primary ray, a secondary ray is cast toward the light source. This shadow ray determines how much light can reach the sample point by checking for density along its path. This process uses Beer-Powder's Laws to calculate the light transmittance.

```
float lightMarching(vec3 ro){
    vec3 lightDir = normalize(lightPosition);

    float t = 0.0;
    float totalDensity = 0.0;
    for( int i=0; i<STEP; i++ )
    {
        vec3 pos = ro + t*lightDir;
        float h = map(pos);
        if( h>min_distance) break;

        float density = 1.0;
        density = sampleDensity(pos);

        totalDensity += density * walk_light_distance;
        t += walk_light_distance;
        i++;
    }
    float transmittance = exp(-totalDensity* _scatter);
    transmittance *= Powder(totalDensity);
    return transmittance;
}
```

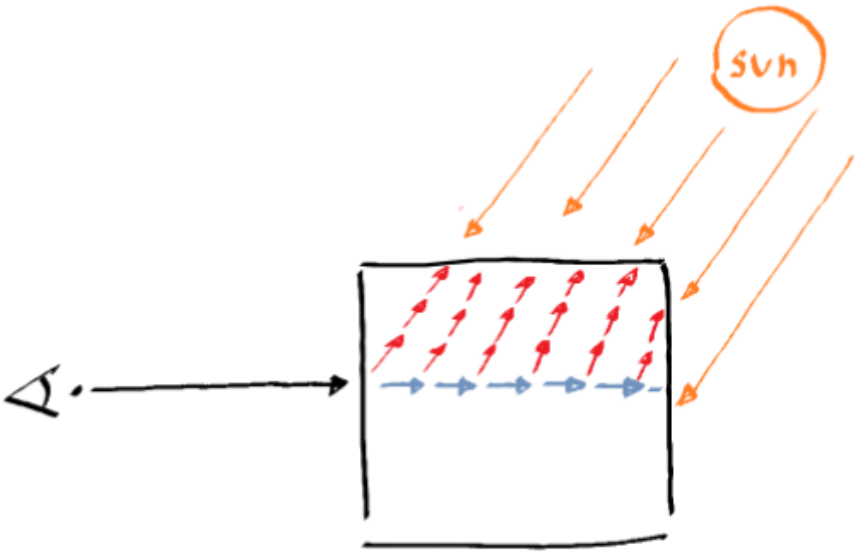

Figure 17 : How primary and shadow ray travel

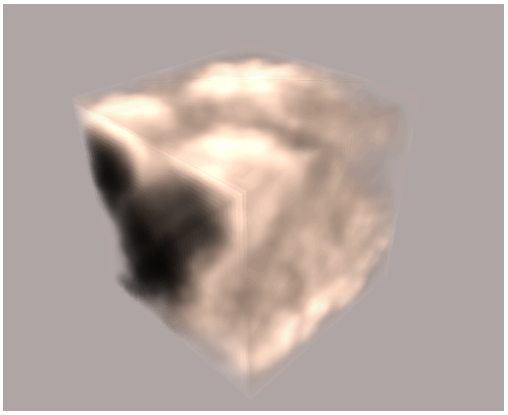

Figure 18 : Cloud with primary and shadow ray

These two ray marches work together to simulate realistic light scattering and soft shadows inside the cloud.

Adjustable Parameter

For artistic purposes, parameters are essential to implement because this lets others use these tools easily. Therefore, I use ImGui to implement the UI for parameter adjustment. There are 13 parameter divide into 4 groups

1. **Shape**
    1. Radius : adjust radius of the boundary

2. Absorption : adjust how light is absorbed through clouds affects both color and transparency.
3. Scatter : adjust how light is scattered through clouds and affect only color.

2. FBM

1. Cut Texture : adjust the lower threshold of the noise
2. Initial Amplitude : adjust amplitude of the FBM.
3. Octave : adjust how many layers of noise.
4. Noise Size : adjust how large noise texture is.

3. Color

1. Ambient light Color : adjust the Ambient light color

4. Sky

1. Sky Color : adjust the color of the sky
2. Sun Angle : adjust the angle of the sun

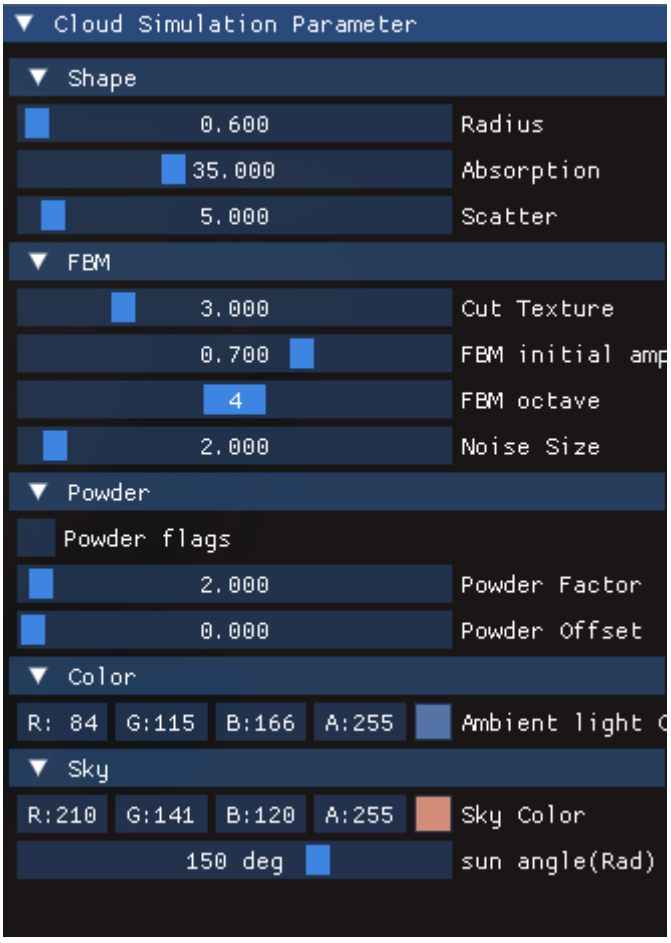

Figure 19 : image of UI for adjusting parameter

result

Github : https://github.com/Rujipas-Thongpao/CloudSim

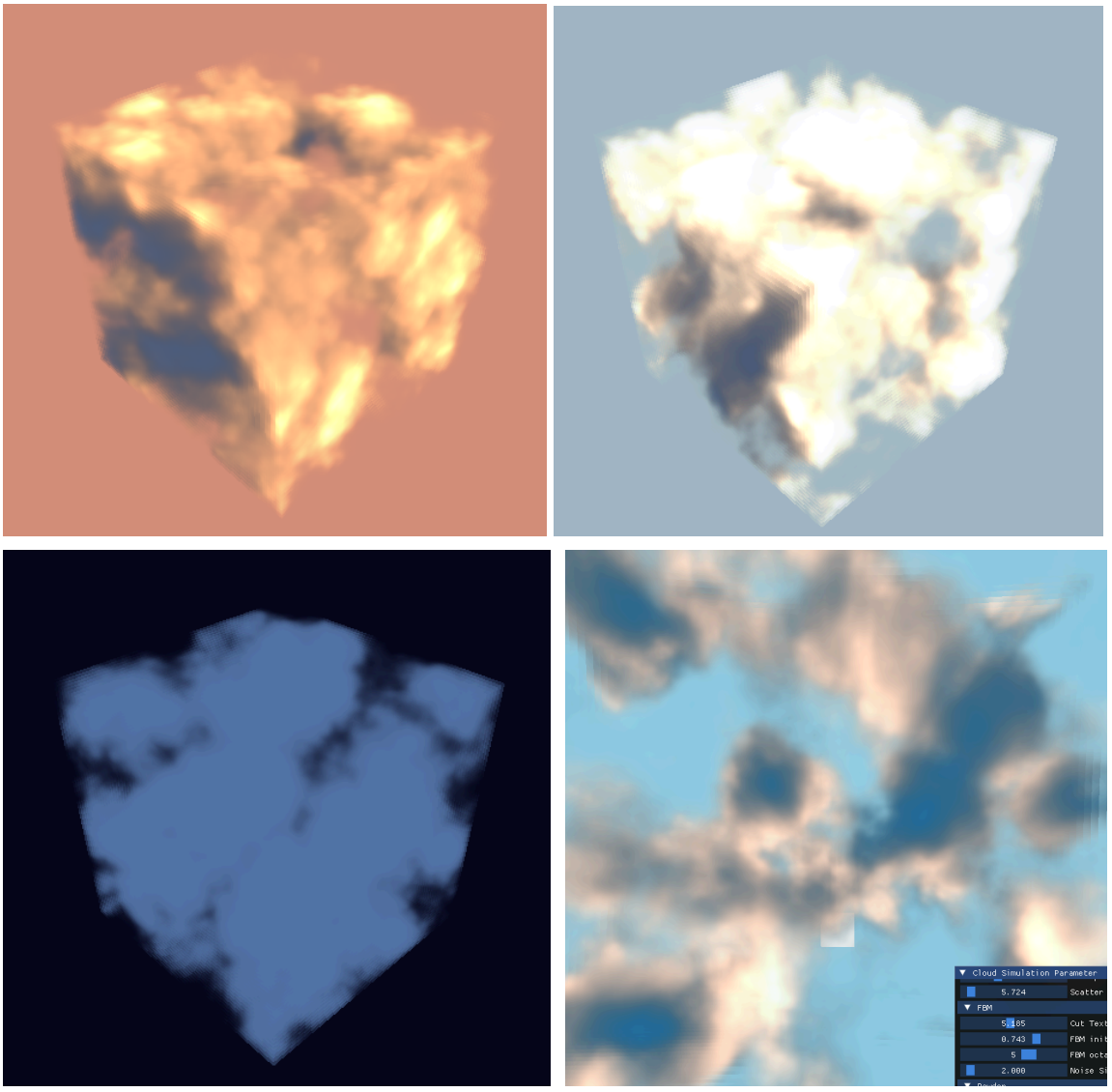

Figure 20,21,22,23 : Result of the clouds simulation in day-light cycle

Obstacle/Solution

1. How perspective is calculated because it's different from traditional rendering and Ray Marching rendering, but I consult with Inigo Quilez who is a professional in the graphic programming community.
2. Optimization is one of the problems until the end of this project because there are lots of calculations in 1 iteration. Therefore, I try to set the boundary as small as possible while keeping the resolution of the noise good, because the noise has lower computation compared to ray marching steps.
3. There is a decent amount of information on the internet, but it's not enough for the newer to come and implement it directly. My approach is to find as much information as possible and combine them into my own understanding.