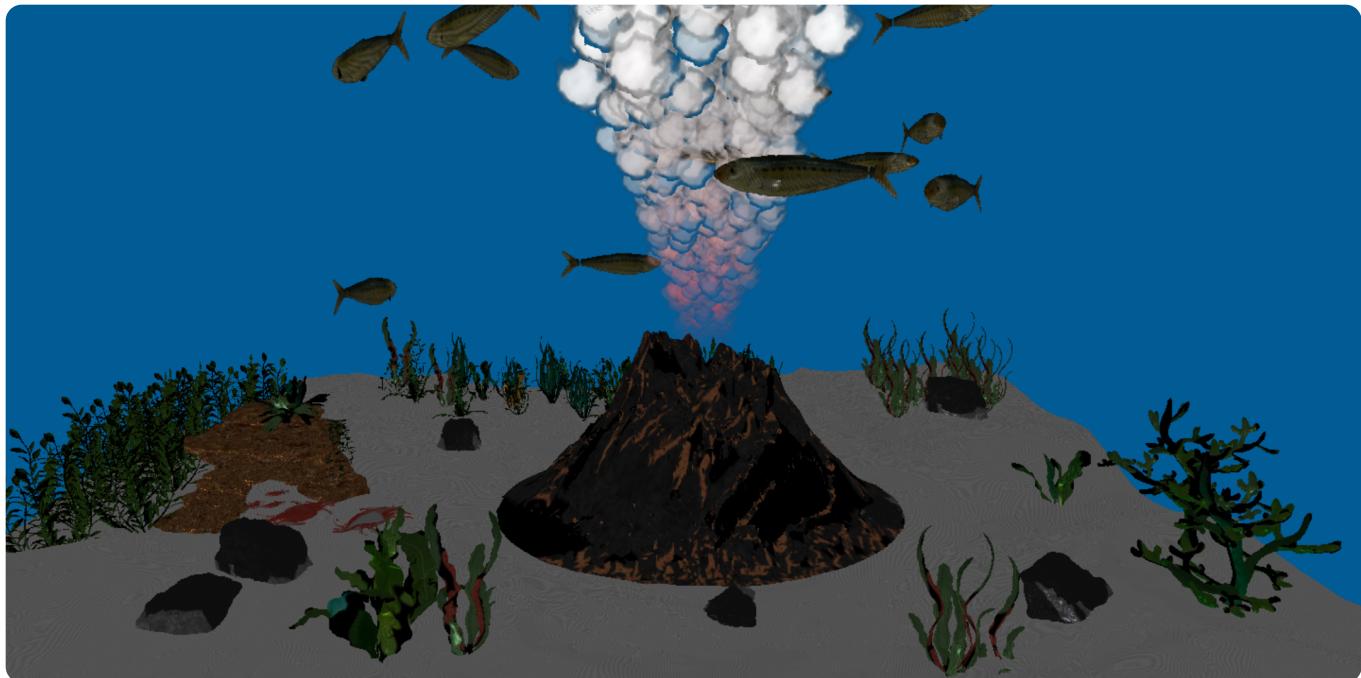
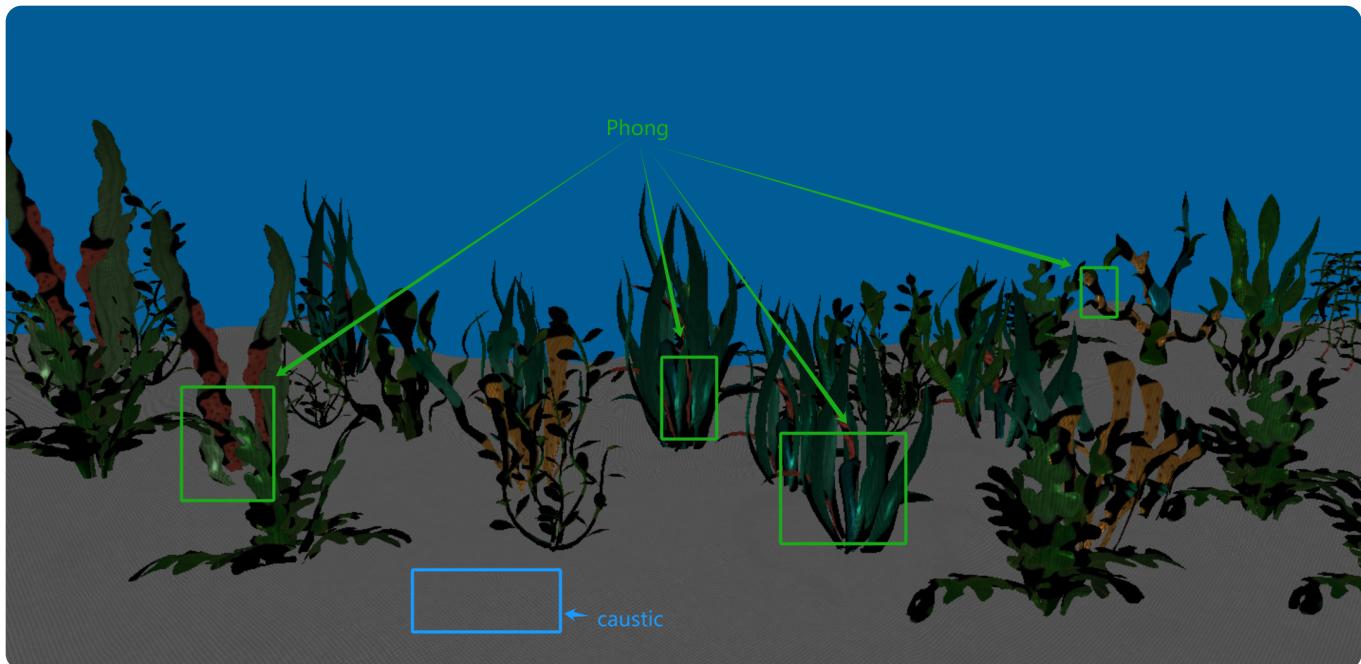


Underwater Volcano



Lighting, Shading and Texture Mapping:



1. A **ambient** lighting that is blue, simulates the color in the ocean. Which will illuminate all objects equally.

```
vec3 ambient = lightAmbient;
```

2. A **diffuse** lighting depends on the angle between the **keep-changing-direction sunlight** and the surface normal. It simulates the light scattered in many directions when it hits a rough surface. It also combines with **caustic effect** that simulate the underwater caustics.

```
void updateIllumination() // called each frame
{
    // Oscillate light position along an axis
    lightPosition = glm::vec3(
        10.0f * sin(timeInSeconds * 2.0), // Faster horizontal oscillation
        15.0f + sin(timeInSeconds * 2.0) * 2.0f, // Smaller vertical oscillation
        10.0f * cos(timeInSeconds * 2.0) // Faster horizontal oscillation
    );

    // Calculate light direction based on the position of the light
    lightDirection = glm::normalize(lightPosition - glm::vec3(0.0f));
}
```

```
// fragmeng shader:
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(-lightDirection);
float diff = max(dot(norm, lightDir), 0.2);
float causticEffect = caustics(FragPos) * causticIntensity;
vec3 diffuse = lightDiffuse * diff * causticEffect;
```

3. A **specular lighting** simulates the bright spot of light that appears on shiny surfaces. It depends on the view direction, light direction, and surface normal.

```
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 64);
vec3 specular = vec3(0.8, 0.8, 1.0) * spec;
```

4. Light **attenuation** simulates the reduction of light intensity over distance, which is particularly important in underwater scenes.

```
float depth = abs(FragPos.y - 50);
float attenuation = exp(-depth * depthFalloff);
vec3 attenuatedLight = (ambient + diffuse + specular) * attenuation;
```

5. **Texture mapping**: load and map vertex to corresponding texture part

```
vec4 texColor = texture(texture1, TexCoords);
```

6. Combine all the lighting together:

```
const vec3 shallowColor = vec3(0.0f, 0.4f, 0.8f); // Bright blue
const vec3 deepColor = vec3(0.0f, 0.1f, 0.3f); // Dark blue
vec3 waterColor = mix(shallowColor, deepColor, depth / 100.0f);
vec3 finalLight = mix(waterColor, attenuatedLight, attenuation);
FragColor = vec4(finalLight * texColor.rgb, 1.0);
```

Advanced Feature: Particle System

To better simulate the smoke from volcano, I build the whole particle system which would spawn several particles each frame For each particle:

- Has velocity, position and lifetime. A velocity is mostly upward and with some random horizontal offsets, and will be slightly changed each frame to make it more realistic
- Being updated in each frame before their life time ends

```
// initialize or reset a particle
for (int i = 0; i < NewParticlePerFrame; ++i) {

    // ...
    // ... search for unused particles

    if (particles[particleIndex].lifetime <= 0.0f)
    {
        // Reset a particle
        Particle& p = particles[particleIndex];
        // start from crater of the volcano
        p.position = startPosition + glm::vec3(horizontalDist(rng), 0.0f,
                                                horizontalDist(rng));
        p.velocity = glm::vec3(horizontalDist(rng), verticalDist(rng),
                               horizontalDist(rng)); // Mostly upward velocity
        p.lifetime = PARTICLE_LIFETIME;
    }
}

// Update all alive particles
```

```

const float dampingFactor = 0.75f; // Slow down speed
for (auto& p : particles) {
    if (p.lifetime > 0.0f) {

        // randomize the velocity
        float randomX = ((rand() % 100) / 100.0f - 0.5f) * 0.01f;
        float randomZ = ((rand() % 100) / 100.0f - 0.5f) * 0.01f;
        p.velocity.x += randomX;
        p.velocity.z += randomZ;

        p.position += p.velocity * deltaTime * dampingFactor;
        p.lifetime -= deltaTime;
    }
}

```

- Smoke texture mapping to make it look more realistic.

```

// Particle quad vertices
float quadVertices[] = {
    // Positions      // Texture Coords
    -0.05f, -0.05f, 0.0f, 0.0f, 0.0f,
    0.05f, -0.05f, 0.0f, 1.0f, 0.0f,
    -0.05f, 0.05f, 0.0f, 0.0f, 1.0f,
    -0.05f, 0.05f, 0.0f, 0.0f, 1.0f,
    0.05f, -0.05f, 0.0f, 1.0f, 0.0f,
    0.05f, 0.05f, 0.0f, 1.0f, 1.0f,
};

glBindBuffer(GL_ARRAY_BUFFER, particleVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(quadVertices), quadVertices, GL_STATIC_DRAW

// vertex shader:
gl_Position = projection * view * vec4(vertex * currentScale + instanceOffset, 1

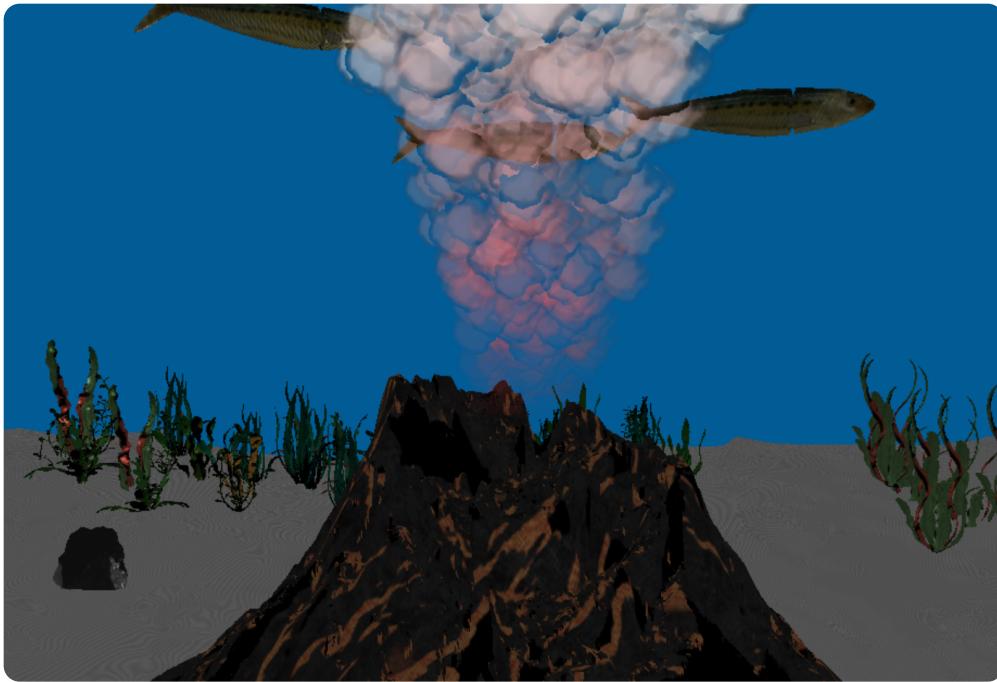
// fragment shader:
vec4 texColor = texture(particleTexture, TexCoords) * vec4(1.0, 1.0, 1.0, alpha)
if (texColor.a < 0.1) discard;
FragColor = vec4((texColor.rgb * particleColor.rgb + redLight) * alpha, texColor

```

- The alpha and scale of particle is changing each frame to simulate the boba gets bigger and more transparent as it floats up.

```
float currentScale = mix(50, 150, pow(instanceLifetimePercent, 1.5));
float alpha = pow(LifetimePercent, 0.5); // Fade out faster near the end
```

- A **red directional light** is also added to simulate the light from crater of the volcano, and be faded as it goes up.



```
// Calculate direction to volcano
vec3 toVolcano = normalize(FragPos - volcanoPosition);

// Calculate alignment (cosine of angle)
float alignment = dot(toVolcano, redLightDirection);

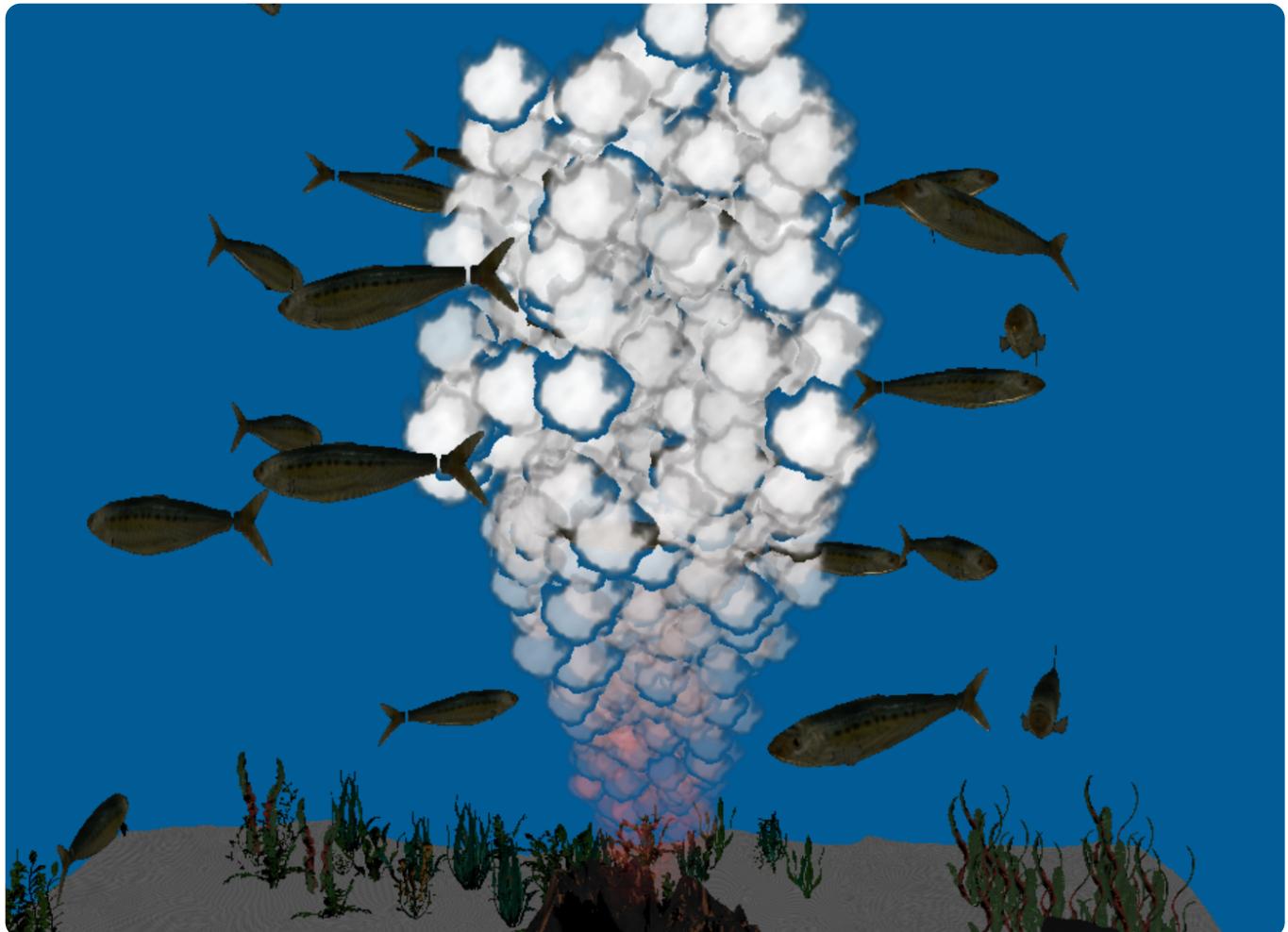
// Beam effect using smoothstep
float beamEffect = smoothstep(1.0 - beamWidth, 1.0, alignment);

// Apply red light with beam effect
float distanceToVolcano = length(FragPos - volcanoPosition);
float falloff = exp(-distanceToVolcano * redLightFalloff);
vec3 redLight = redLightColor * redLightIntensity * beamEffect * falloff;

// Combine particle color with red light
FragColor = vec4((texColor.rgb * particleColor.rgb + redLight) * alpha, texCo.
```

Scene Setup

The scene is centered with an active volcano that is spreading out smoke from its crater. A group of fishes are swimming around the vivid smoke.



Declaration

This work is all by myself.