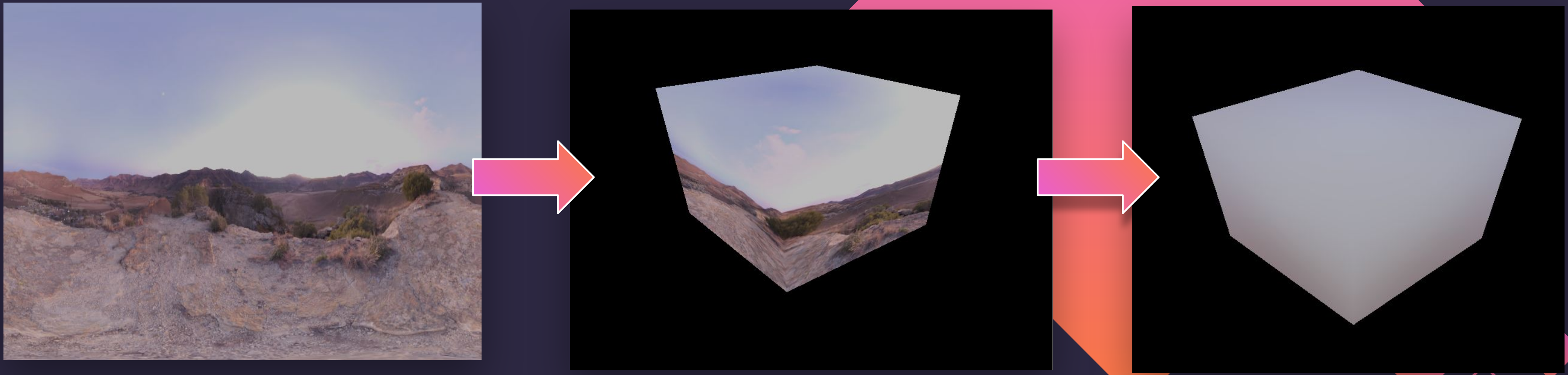
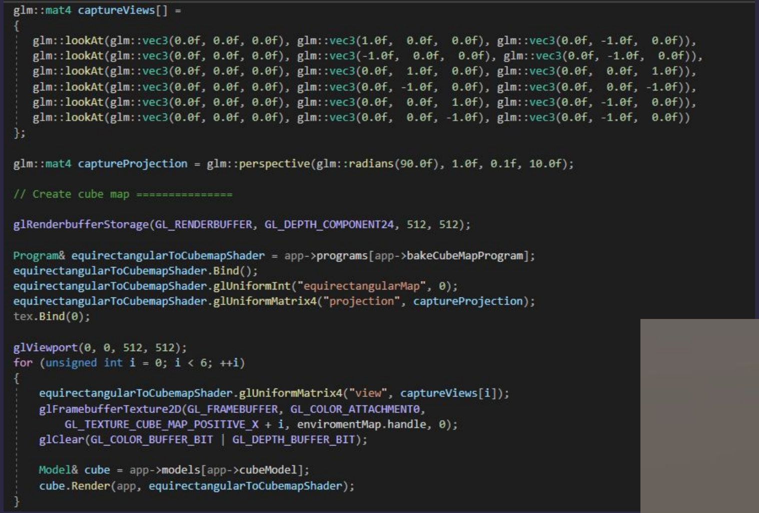
AGP:
WATER EFFECT & ENVIRONMENT MAPPING

ENVIRONMENT MAPPING



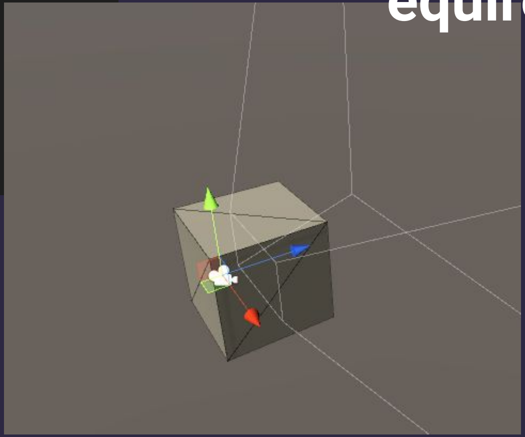
Bake texture to cubemap



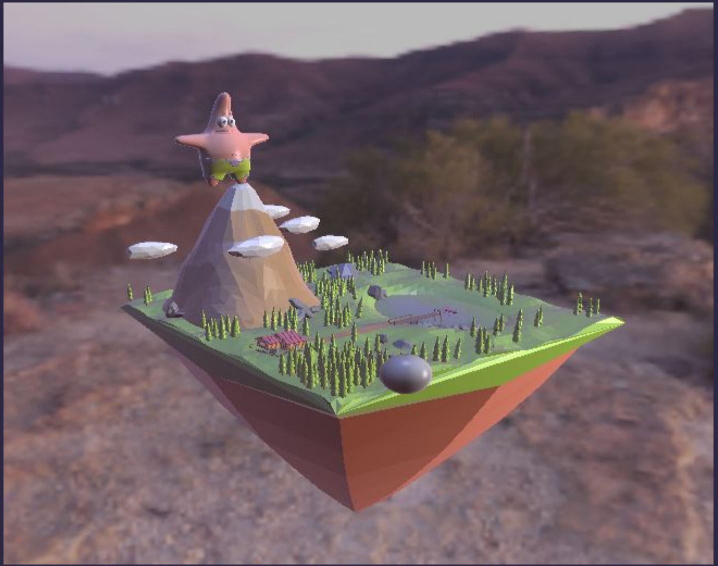
```
glm::mat4 captureViews[] =
{
    glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(1.0f,  0.0f,  0.0f), glm::vec3(0.0f, -1.0f,  0.0f)),
    glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(-1.0f,  0.0f,  0.0f), glm::vec3(0.0f, -1.0f,  0.0f)),
    glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f,  1.0f,  0.0f), glm::vec3(0.0f,  0.0f,  1.0f)),
    glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, -1.0f,  0.0f), glm::vec3(0.0f,  0.0f, -1.0f)),
    glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f,  0.0f,  1.0f), glm::vec3(0.0f, -1.0f,  0.0f)),
    glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f,  0.0f, -1.0f), glm::vec3(0.0f, -1.0f,  0.0f))
};

glm::mat4 captureProjection = glm::perspective(glm::radians(90.0f), 1.0f, 0.1f, 10.0f);

// Create cube map ===============

glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 512, 512);

Program& equirectangularToCubemapShader = app->programs[app->bakeCubeMapProgram];
equirectangularToCubemapShader.Bind();
equirectangularToCubemapShader.glUniformInt("equirectangularMap", 0);
equirectangularToCubemapShader.glUniformMatrix4("projection", captureProjection);
tex.Bind(0);

glViewport(0, 0, 512, 512);
for (unsigned int i = 0; i < 6; ++i)
{
    equirectangularToCubemapShader.glUniformMatrix4("view", captureViews[i]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
        GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, enviromentMap.handle, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    Model& cube = app->models[app->cubeModel];
    cube.Render(app, equirectangularToCubemapShader);
}
```

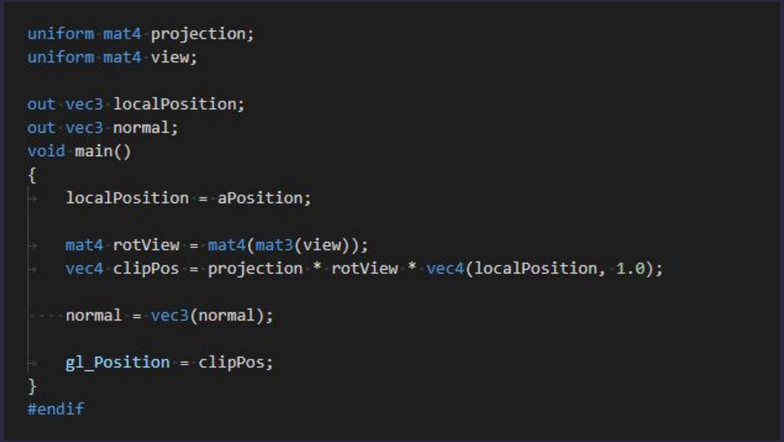
Shader:

- **HDR to cubemap which takes the local position of the cube to sample the equirectangular map**

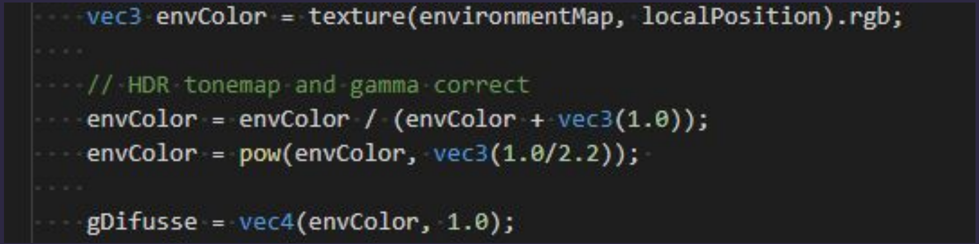


Render Skybox



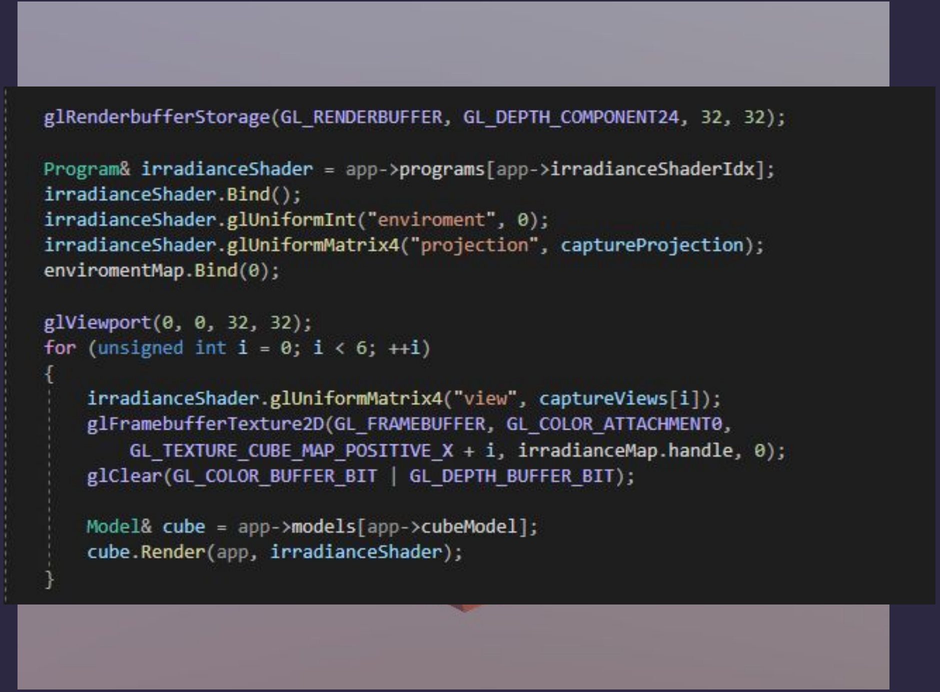


```
uniform mat4 projection;
uniform mat4 view;

out vec3 localPosition;
out vec3 normal;
void main()
{
    localPosition = aPosition;

    mat4 rotView = mat4(mat3(view));
    vec4 clipPos = projection * rotView * vec4(localPosition, 1.0);

    normal = vec3(normal);

    gl_Position = clipPos;
}
#endif
```



```
    vec3 envColor = texture(environmentMap, localPosition).rgb;

    // HDR tonemap and gamma correct
    envColor = envColor / (envColor + vec3(1.0));
    envColor = pow(envColor, vec3(1.0/2.2));

    gDifusse = vec4(envColor, 1.0);
```

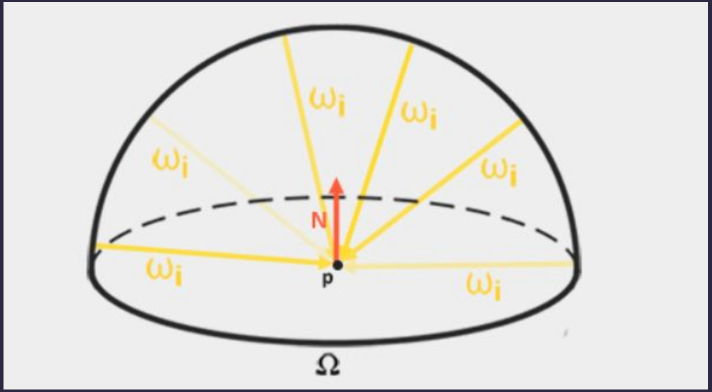
Irradiance map



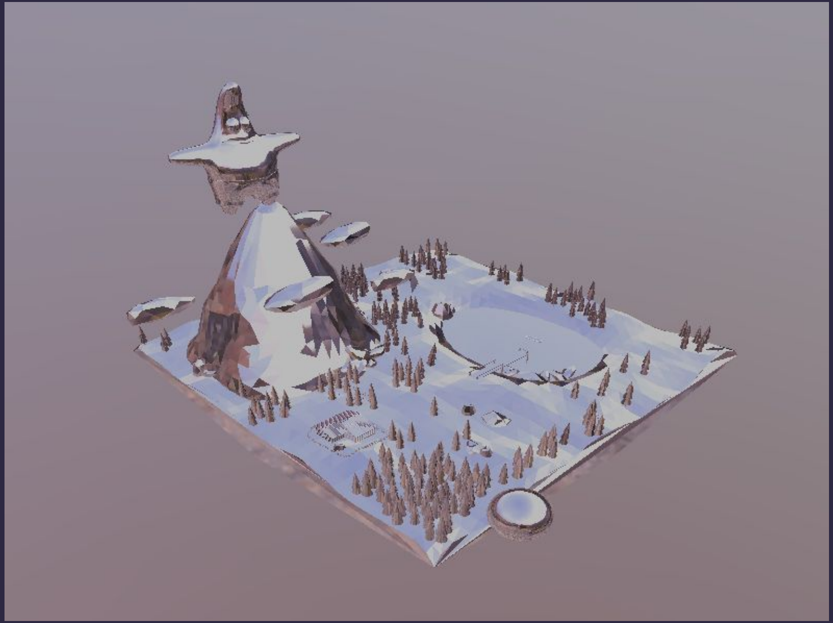
```
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 32, 32);

Program& irradianceShader = app->programs[app->irradianceShaderIdx];
irradianceShader.Bind();
irradianceShader.glUniformInt("enviroment", 0);
irradianceShader.glUniformMatrix4("projection", captureProjection);
enviromentMap.Bind(0);

glViewport(0, 0, 32, 32);
for (unsigned int i = 0; i < 6; ++i)
{
    irradianceShader.glUniformMatrix4("view", captureViews[i]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
        GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, irradianceMap.handle, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    Model& cube = app->models[app->cubeModel];
    cube.Render(app, irradianceShader);
}
```

Shaders:

- **For every fragment calculate the convolution**

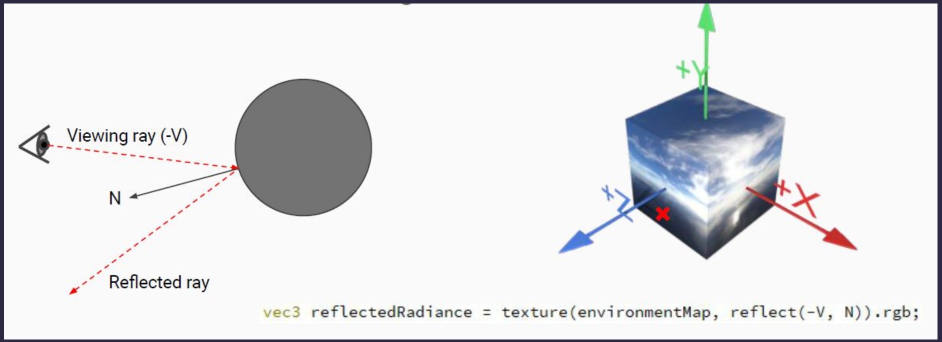


Reflectance



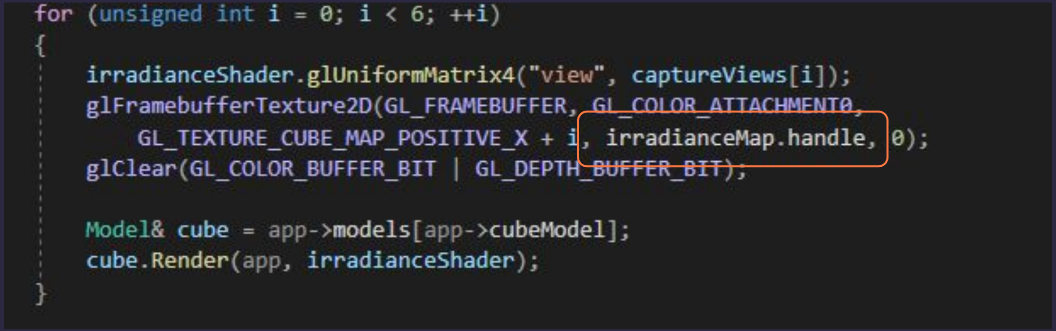
<u>Shaders:</u>

- **Calculates the reflected ray**

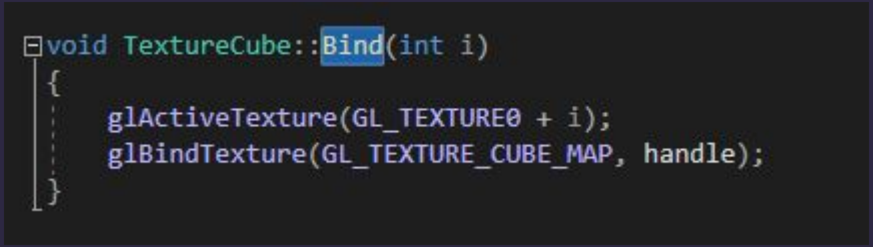


```
vec3 reflectedRadiance = texture(environmentMap, reflect(-V, N)).rgb;
```

Problems

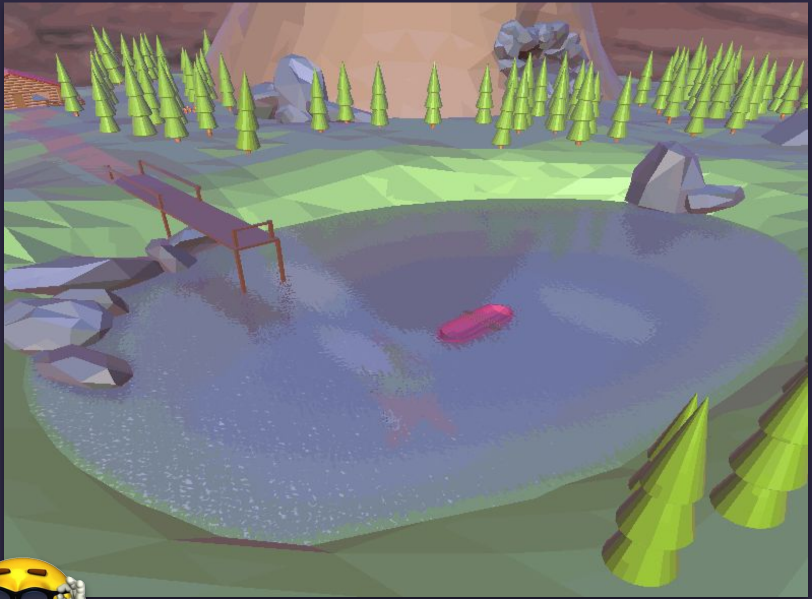


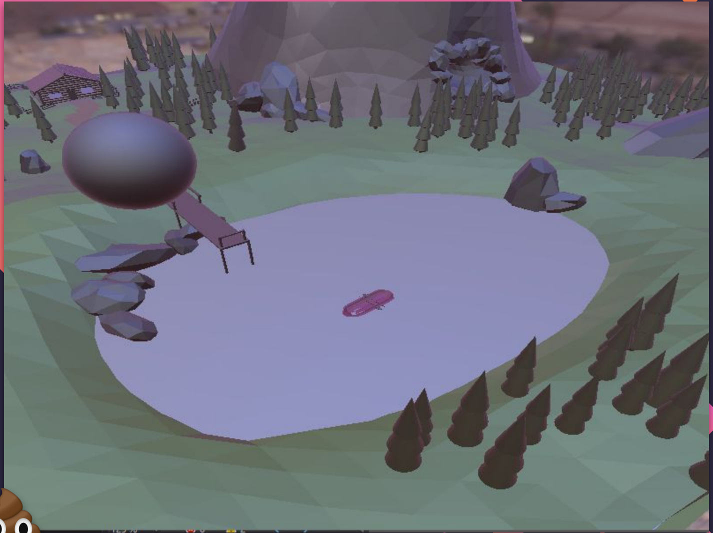
```
for (unsigned int i = 0; i < 6; ++i)
{
    irradianceShader.glUniformMatrix4("view", captureViews[i]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
        GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, irradianceMap.handle, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    Model& cube = app->models[app->cubeModel];
    cube.Render(app, irradianceShader);
}
```



```
void TextureCube::Bind(int i)
{
    glActiveTexture(GL_TEXTURE0 + i);
    glBindTexture(GL_TEXTURE_CUBE_MAP, handle);
}
```

WATER EFFECT





IMPLEMENTATION

2 Shaders:

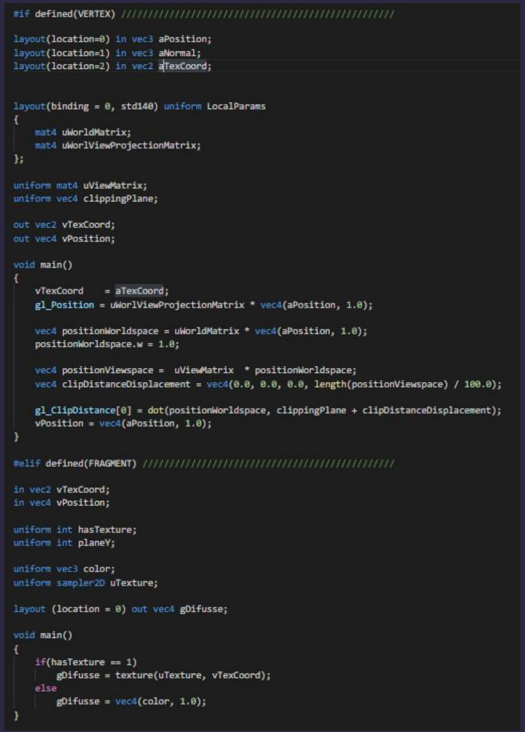
- Clipping Shader
- Water Effect Shader

Pipeline:

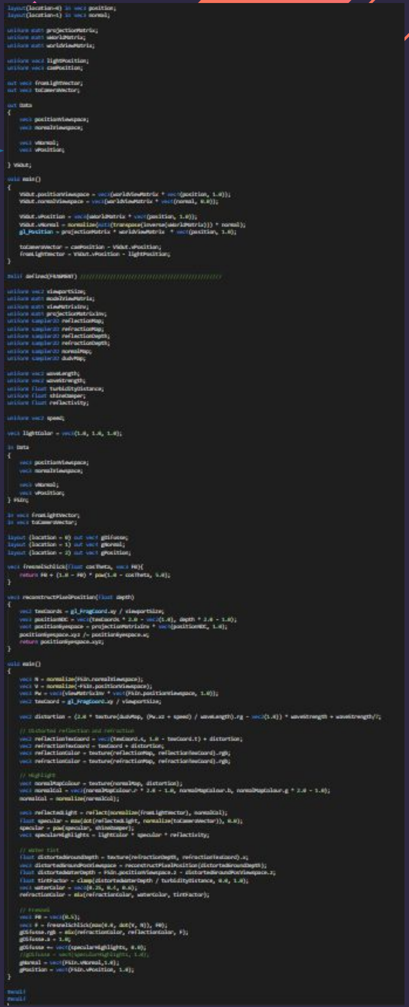
1. Create buffers & textures
2. Generate textures
3. Render water plane

WATER EFFECT SHADER

CLIPPING SHADER



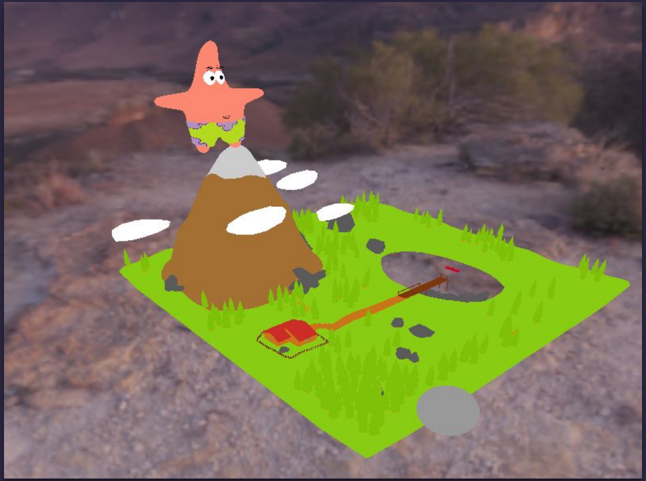
```
#if defined(VERTEX) ///////////////////////////////////////////////////

layout(location=0) in vec3 aPosition;
layout(location=1) in vec3 aNormal;
layout(location=2) in vec2 aTexCoord;


layout(binding = 0, std140) uniform LocalParams
{
    mat4 uWorldMatrix;
    mat4 uWorlViewProjectionMatrix;
};

uniform mat4 uViewMatrix;
uniform vec4 clippingPlane;

out vec2 vTexCoord;
out vec4 vPosition;

void main()
{
    vTexCoord    = aTexCoord;
    gl_Position = uWorlViewProjectionMatrix * vec4(aPosition, 1.0);

    vec4 positionWorldspace = uWorldMatrix * vec4(aPosition, 1.0);
    positionWorldspace.w = 1.0;

    vec4 positionViewspace =  uViewMatrix  * positionWorldspace;
    vec4 clipDistanceDisplacement = vec4(0.0, 0.0, 0.0, length(positionViewspace) / 100.0);

    gl_ClipDistance[0] = dot(positionWorldspace, clippingPlane + clipDistanceDisplacement);
    vPosition = vec4(aPosition, 1.0);
}

#elif defined(FRAGMENT) ///////////////////////////////////////////////

in vec2 vTexCoord;
in vec4 vPosition;

uniform int hasTexture;
uniform int planeY;

uniform vec3 color;
uniform sampler2D uTexture;

layout (location = 0) out vec4 gDifusse;

void main()
{
    if(hasTexture == 1)
        gDifusse = texture(uTexture, vTexCoord);
    else
        gDifusse = vec4(color, 1.0);
}
```



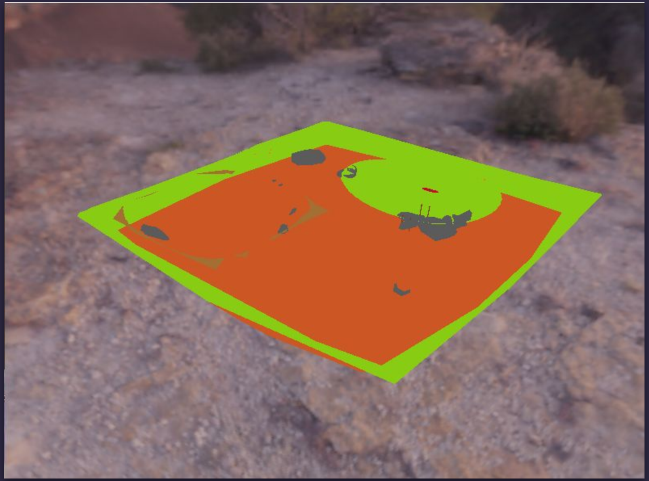
```
layout(location=0) in vec3 position;
layout(location=1) in vec3 normal;

uniform mat4 projectionMatrix;
uniform mat4 worldMatrix;
uniform mat4 worldViewMatrix;

uniform vec3 lightPosition;
uniform vec3 camPosition;

out vec3 fromLightVector;
out vec3 toCameraVector;

out Data
{
    vec3 positionViewspace;
    vec3 normalViewspace;

    vec3 vNormal;
    vec3 vPosition;

} VsOut;

void main()
{
    VsOut.positionViewspace = vec3(worldViewMatrix * vec4(position, 1.0));
    VsOut.normalViewspace = vec3(worldViewMatrix * vec4(normal, 0.0));

    VsOut.vPosition = vec3(worldMatrix * vec4(position, 1.0));
    VsOut.vNormal = normalize(mat3(transpose(inverse(worldMatrix))) * normal);
    gl_Position = projectionMatrix * worldViewMatrix  * vec4(position, 1.0);

    toCameraVector = camPosition - VsOut.vPosition;
    fromLightVector = VsOut.vPosition - lightPosition;
}

#elif defined(FRAGMENT) ///////////////////////////////////////////////

uniform vec2 viewportSize;
uniform mat4 modelViewMatrix;
uniform mat4 viewMatrixInv;
uniform mat4 projectionMatrixInv;
uniform sampler2D reflectionMap;
uniform sampler2D refractionMap;
uniform sampler2D reflectionDepth;
uniform sampler2D refractionDepth;
uniform sampler2D normalMap;
uniform sampler2D dudvMap;

uniform vec2 waveLength;
uniform vec2 waveStrength;
uniform float turbidityDistance;
uniform float shineDamper;
uniform float reflectivity;

uniform vec2 speed;

vec3 lightColor = vec3(1.0, 1.0, 1.0);

in Data
{
    vec3 positionViewspace;
    vec3 normalViewspace;

    vec3 vNormal;
    vec3 vPosition;
} FsIn;

in vec3 fromLightVector;
in vec3 toCameraVector;

layout (location = 0) out vec4 gDifusse;
layout (location = 1) out vec4 gNormal;
layout (location = 2) out vec4 gPosition;

vec3 fresnelSchlick(float cosTheta, vec3 F0){
    return F0 + (1.0 - F0) * pow(1.0 - cosTheta, 5.0);
}

vec3 reconstructPixelPosition(float depth)
{
    vec2 texCoords = gl_FragCoord.xy / viewportSize;
    vec3 positionNDC = vec3(texCoords * 2.0 - vec2(1.0), depth * 2.0 - 1.0);
    vec4 positionEyespace = projectionMatrixInv * vec4(positionNDC, 1.0);
    positionEyespace.xyz /= positionEyespace.w;
    return positionEyespace.xyz;
}

void main()
{
    vec3 N = normalize(FsIn.normalViewspace);
    vec3 V = normalize(-FsIn.positionViewspace);
    vec3 Pw = vec3(viewMatrixInv * vec4(FsIn.positionViewspace, 1.0));
    vec2 texCoord = gl_FragCoord.xy / viewportSize;

    vec2 distortion = (2.0 * texture(dudvMap, (Pw.xz + speed) / waveLength).rg - vec2(1.0)) * waveStrength + waveStrength/7;

    // Distorted reflection and refraction
    vec2 reflectionTexCoord = vec2(texCoord.s, 1.0 - texCoord.t) + distortion;
    vec2 refractionTexCoord = texCoord + distortion;
    vec3 reflectionColor = texture(reflectionMap, reflectionTexCoord).rgb;
    vec3 refractionColor = texture(refractionMap, refractionTexCoord).rgb;

    // Highlight
    vec4 normalMapColour = texture(normalMap, distortion);
    vec3 normalCol = vec3(normalMapColour.r * 2.0 - 1.0, normalMapColour.b, normalMapColour.g * 2.0 - 1.0);
    normalCol = normalize(normalCol);

    vec3 reflectedLight = reflect(normalize(fromLightVector), normalCol);
    float specular = max(dot(reflectedLight, normalize(toCameraVector)), 0.0);
    specular = pow(specular, shineDamper);
    vec3 specularHighlights = lightColor * specular * reflectivity;

    // Water tint
    float distortedGroundDepth = texture(refractionDepth, refractionTexCoord).x;
    vec3 distortedGroundPosViewspace = reconstructPixelPosition(distortedGroundDepth);
    float distortedWaterDepth = FsIn.positionViewspace.z - distortedGroundPosViewspace.z;
    float tintFactor = clamp(distortedWaterDepth / turbidityDistance, 0.0, 1.0);
    vec3 waterColor = vec3(0.25, 0.4, 0.6);
    refractionColor = mix(refractionColor, waterColor, tintFactor);

    // Fresnel
    vec3 F0 = vec3(0.5);
    vec3 F = fresnelSchlick(max(0.0, dot(V, N)), F0);
    gDifusse.rgb = mix(refractionColor, reflectionColor, F);
    gDifusse.a = 1.0;
    gDifusse += vec4(specularHighlights, 0.0);
    //gDifusse = vec4(specularHighlights, 1.0);
    gNormal = vec4(FsIn.vNormal,1.0);
    gPosition = vec4(FsIn.vPosition, 1.0);
}

#endif
#endif
```

CLIPPING SHADER

Generates color and depth texture. With glClipDistance discards the pixels under or above 0.



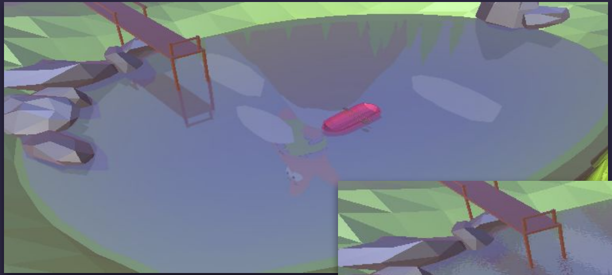
Reflection Texture



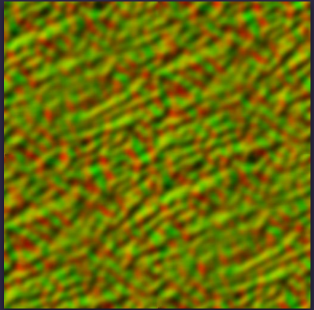
Refraction Texture

WATER SHADER

- Combines textures and project them into planes
- Normal texture used for highlight
- DudV texture used for distortion

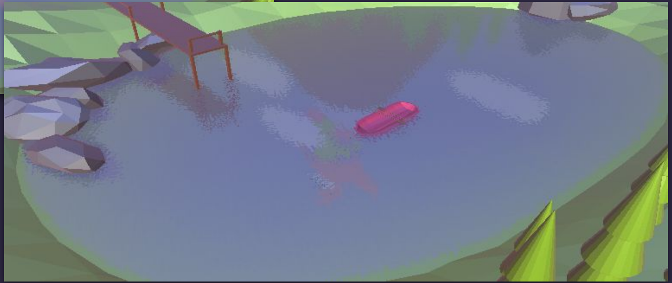


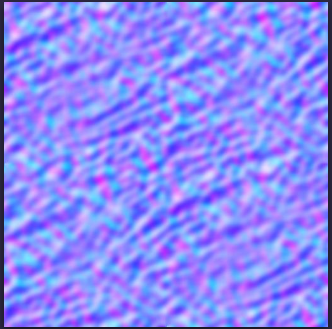
No distortion



Dudv Texture

No highlight



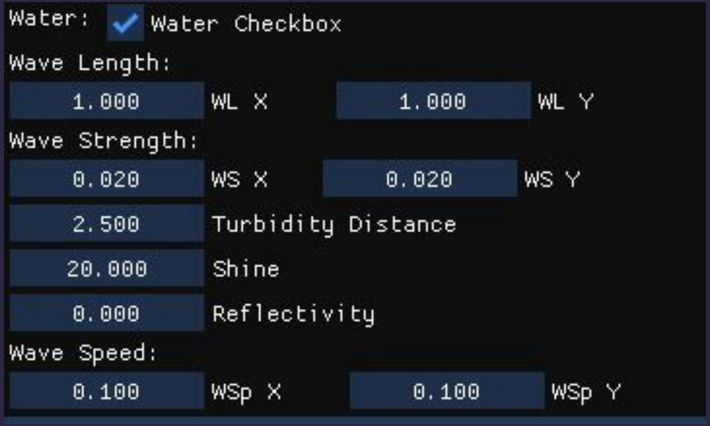


Normal Texture

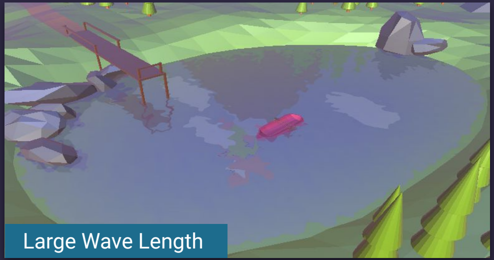
WATER SHADER

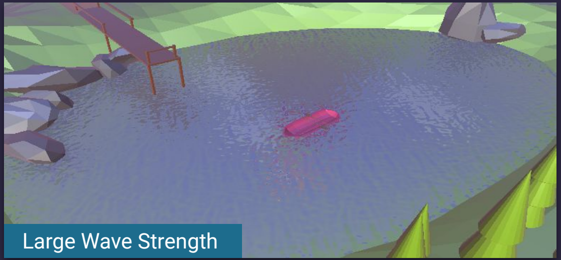
There are some modifiable values:

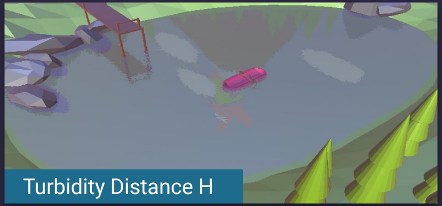
- Wave Length
- Wave Strength
- Turbidity Distance
- Shine Damper
- Reflectivity
- Wave Speed

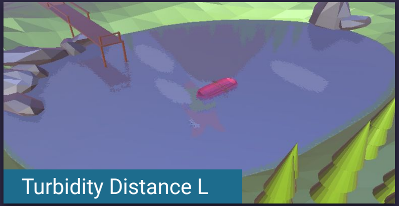


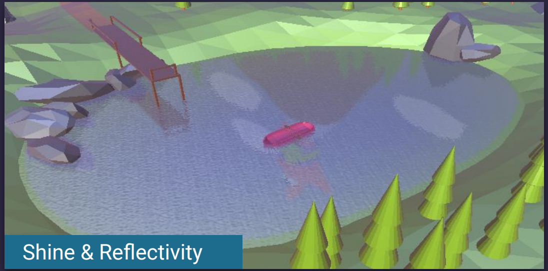
```
Water: ✓ Water Checkbox
Wave Length:
  1.000  WL X       1.000  WL Y
Wave Strength:
  0.020  WS X       0.020  WS Y
  2.500  Turbidity Distance
 20.000  Shine
  0.000  Reflectivity
Wave Speed:
  0.100  WSp X      0.100  WSp Y
```


Large Wave Length


Large Wave Strength


Turbidity Distance H


Turbidity Distance L


Shine & Reflectivity

TROUBLES :$

- I DIDN'T UNDERSTAND NOTHING :_)
  - Yessica saved my life helping me
- WATER WASN'T FLAT & CLIPPING DIDN'T WORK AS EXPECTED
  - Use a flat plane and not the devil water plane
- WATER SHADER NORMAL PROBLEM W LIGHTING
  - Think

FEEDBACK

UNITY

Check how all this can be applied in Unity or similar Engine

2 MUCH SHADER TOY

Lot of time dedicated too shadertoy which we won't see again

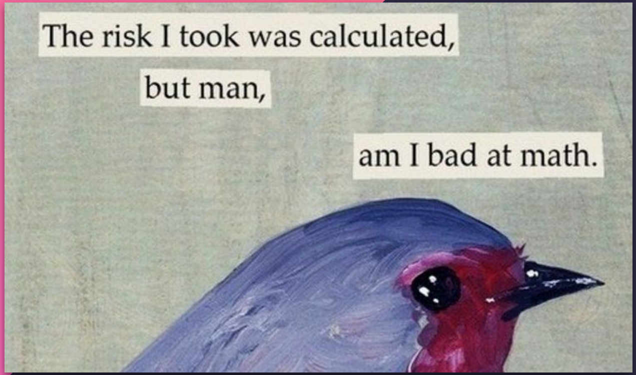
WHY?

“I thought it will be fun and that i could apply it to Unity. But bro, I didn’t understand shit :_)”

“I thought sky would look cool with water”

YESS

ALEX


The risk I took was calculated,
but man,
am I bad at math.

THANKS!

Do you have any questions?