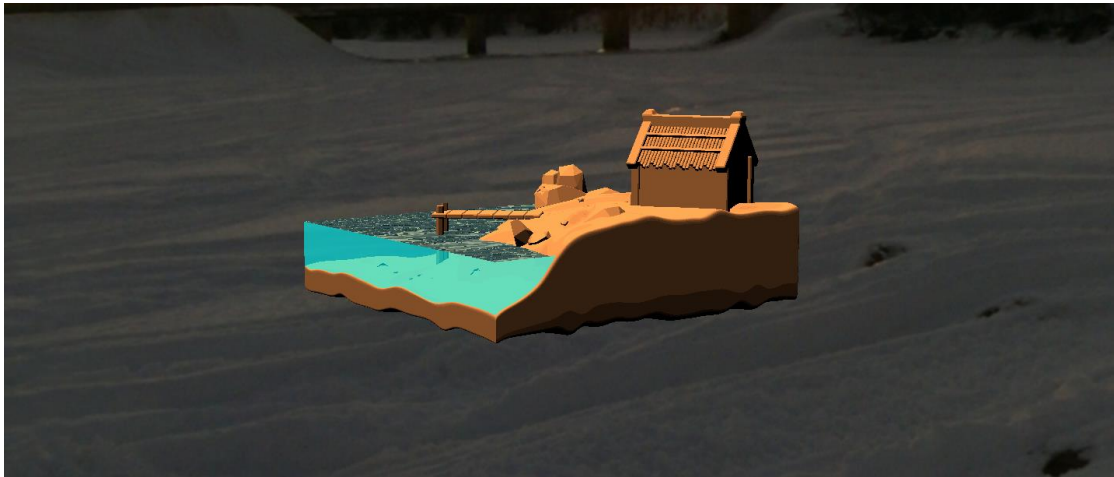


# Computer Graphics for Games 2024

## Stylized island

A stylized island with toon shading lighting and a water simulation.



**Catarina Costa**

ist1112377

MEIC

catarina.g.costa@tecnico.ulisboa.pt

## **Abstract**

In this document I will explain how I've implemented my chosen technical challenges to simulate water and a stylized island.

For the water I wanted to recreate waves, normal maps seemed like the best solution, so that's how I've implemented it.

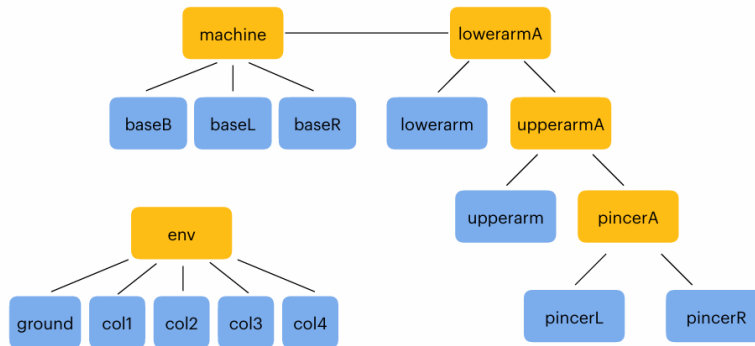
As for the transparencies and reflections on the water, a skybox was used, and the Fresnel effect recreated.

To follow my inspiration material, I've chosen a non-realistic lighting model, in my case a toon shading, so not to include silhouettes.

# 1. Technical Challenges

## 1.1. Generic Scenegraph

A Scenegraph was implemented. The Scenegraph works in a tree structure with parent and child nodes, where the transformations done in the parents are applied in their children, making it easier to perform transformations in groups.



## 1.2. Toon shading Model

In my project the island uses a non-realistic lighting model, more specifically toon shading. To do this, a light position, colour and an ambient colour, was chosen and a 13-level toon shading model was used to light it, the value was chosen to show a mix of a more blended and natural effect without losing to much of the stylized look.

## 1.3. Normal Mapping

To simulate the water's waves, a normal map was used. To make it look better I've used a variable time to change slightly the fragment's texture coordinates, in order to simulate movement in the water.

## 1.4. Transparent Material

I would identify this as the hardest challenge to tackle, proving to be more difficult than expected in the beginning.

First, I'll explain in general how it was implemented:

To start a skybox was created, this was used to implement the reflections and refractions of the water, using the camera's position and the fragment's normal, the fragment's colour was calculated based on the image from the skybox.

To determine if the water was reflecting or refracting, Fresnel reflectance was used. This created a problem, in my project I've decided to use a mesh that has part of the island inside the water mesh, my objective was to have the island's mesh on top and showing in the refraction of the water.

Since, I was using the skybox for the refraction the island never showed inside the water.

There were two possible solutions for this problem: one was using a dynamic skybox that would include the island mesh in it making it show in the refractions and reflection of the water; the other was changing the alpha component of the water

fragments to show the island, the problem with this solution was that it made it impossible to have a distortion of the image refracted in the water.

I will explain in the next section in more depth the challenges, with each possible solution, for now I'll explain the final decision.

The top of the water stayed the same, it uses only the original skybox, the sides of the water mesh were changed to be only transparent by changing the alpha component to show the island.

The dynamic skybox is still used but more as an extra. By clicking "B" the user can change between the normal mode, and one were instead of the island we have a small cube rotating around the water, and by using a dynamic cubemap it's possible to see the cube, and the background reflected and refracted in the water.

## 2. Technical Solutions

### 2.1. Generic Scenegraph

To implement the Scenegraph a new class was created.

This class contains the nodes information, such as its mesh, the parent, a vector of its children, color, scale and both its own transformation and the resulting transformation that takes into account the parent's.

When creating the node, it is possible to send or not, the mesh information in case we only want to create an empty node, that serves only to group other nodes and perform transformations on them.

The draw function calls the mesh draw.

Before drawing, the function update must be called to get the final transformation, which is calculated, here based on if the node has a parent or not and updates the child's final transform information.

```
void SceneNode::Update() {  
    if (parent) { // This node has a parent ...  
        worldTransform = parent->worldTransform * transform;  
    }  
    else { // Root node , world transform is local transform !  
        worldTransform = transform;  
    }  
    for (std::vector<SceneNode* >::iterator i = children.begin(); i != children.end(); ++i) {  
        (*i)->Update();  
    }  
}
```

To create the class SceneNode I followed a tutorial for scenegraph implementation.  
[1]

### 2.2. Toonshading Model

For the toonshading a light direction, a light colour and an ambient colour were chosen.

Another variable necessary is the number of levels for the toonshading which determines the number of different colours that will be shown depending on the value from the intensity level of the light, calculated as the dot product, between the light direction and the fragments normal.

```

in vec4 exColor;
in vec3 exNormal;
out vec4 outColor;
uniform samplerCube textCubeMap;

int toonLevels = 13;
float toonScaleFactor = 1.0/ toonLevels;
vec3 lightDirection = vec3(0.4, 1.0, 0.0);
vec3 lightColor = vec3(1.0, 1.0, 1.0);
vec3 ambientColor = vec3(0.2, 0.2, 0.2);

void main(void){

    vec3 N = normalize(exNormal);

    vec3 lightDir = normalize(lightDirection);
    float intensity = dot(N, lightDir);

    float quantizedDiffuse = floor(intensity / toonScaleFactor) * toonScaleFactor;

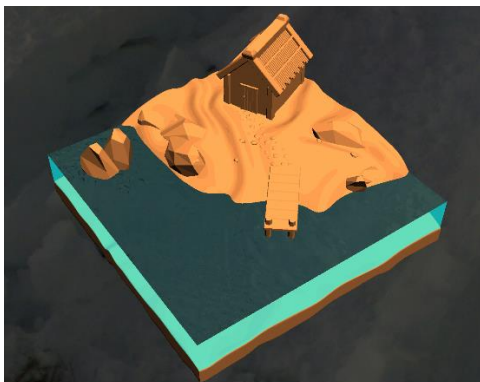
    vec3 toonShadedColor = quantizedDiffuse * lightColor + ambientColor;

    vec3 finalColor = toonShadedColor * exColor.rgb;

    outColor = vec4(finalColor, exColor.a);
}

```

The result looks like this:



To implement the toon shading I used the following resource [5]

## 2.3. Normal Mapping

To use the normal map correctly it was necessary to have the water normal's in tangent space, for this the tangent coordinates are necessary. The file used for the mesh is an .obj file, this type of file imported from blender does not include the tangent coordinates, so using the normal it was calculated when creating the mesh.

```

mgl::Mesh* MyApp::createMesh(const std::string& mesh_dir, const std::string& mesh_file) {
    auto* mesh = new mgl::Mesh();
    mesh->joinIdenticalVertices();
    if(mesh_file == "water.obj")
        mesh->calculateTangentSpace();
    mesh->create(mesh_dir + mesh_file);
    return mesh;
}

```

Having the tangents it is now possible to calculate the tangent space.

From here there were two ways of working, either in world space, or in tangent space.

For me it was easier to just work in world space, so that meant creating the TBN matrix to transform from tangent to world space by using the ModelMatrix.

The TBN will be used in the resulting normal from the normal map to transform it to world space, since it is the only one in tangent space.

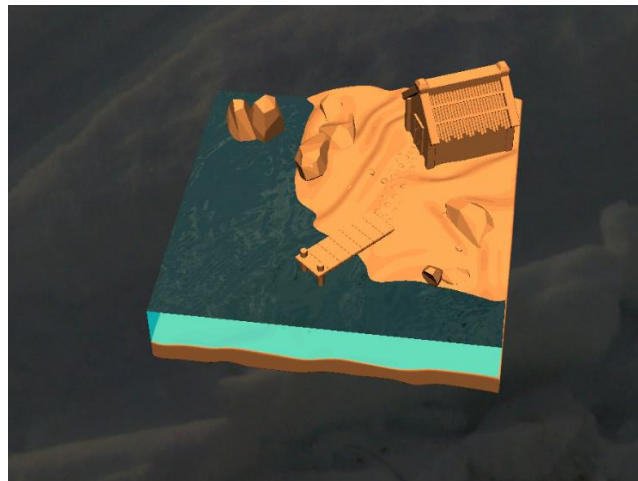
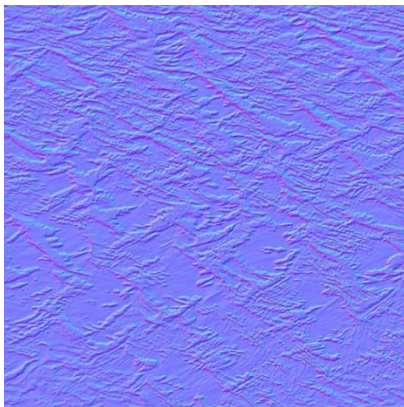
```
void main()
{
    Normal = mat3(transpose(inverse(ModelMatrix))) * inNormal;
    Position = vec3(ModelMatrix * inPosition);
    exColor = ColorVecId;
    exTexCoord = inTexCoord;
    vec3 T = normalize(vec3(ModelMatrix * vec4(inTangent,0.0)));
    vec3 B = cross(Normal, T);
    TBN = mat3(T, B, Normal);
    gl_Position = ProjectionMatrix * ViewMatrix * vec4(Position, 1.0);
}
```

```
25 void main(void) {
26     vec2 movingTexCoord = exTexCoord + vec2(-sin(time) * 0.001, sin(time)*0.003);
27
28     vec3 NormalC = texture(normalMap, movingTexCoord).rgb;
29     NormalC = NormalC * 2.0 - 1.0;
30     NormalC = normalize(TBN * NormalC);
31 }
```

The normal obtained from the normal map is in a range of [0,1], whereas we need it in a range of [-1,1], so we transform it like it's done in line 29.

So, by using the next image as the normal map file, we get the following result.

In my case I've decided to only have the normal map affecting the top of the water mesh.



To implement the normal mapping I've used the [learnopengl.com](http://learnopengl.com) resources. [2]

## 2.4. Transparent Material

To create a transparent material two things were necessary; one was having reflections, and another was having refractions. To create this effect, I used a skybox.

- **Skybox:**

To have a skybox a couple of elements were necessary;

First, I've created a cube mesh, with its own shader, this mesh has the texture from the files chosen for the skybox.

Since the cube represents a background, it needs to be always behind other objects, to do this in line 270 I set the depth function to less or equal ensuring that the cube is always behind the other objects, and on line 271 I ensure that the inside of the cube is seen.

```
265
266     void MyApp::drawSkybox() {
267         createSkyboxShaderProgram();
268         Shaders->bind();
269         cubemap.bind();
270         glDepthFunc(GL_LEQUAL);
271         glCullFace(GL_FRONT);
272         root4->Update();
273         drawNode(root4);
274         glDepthFunc(GL_LESS);
275         glCullFace(GL_BACK);
276     }
```

For the cubemap texture stb\_image.h was used to load the images and then the textures set for every face of the cube.

Having the skybox ready to use in the shader, we can now start working in the reflections and refractions.

For the skybox implementation I've used the learnopengl.com resources [3]

- **Refraction:**

To calculate the refraction color, I've used the vector -I, which represents the vector from the camera to the fragment position. By using the normal we can determine the color from the refraction.

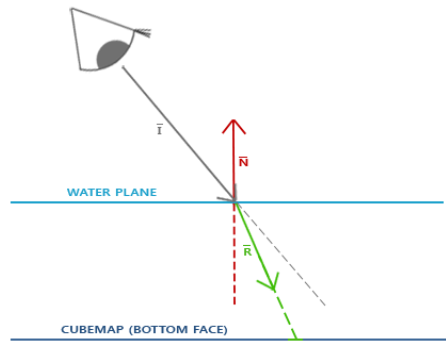
Since water has a displacement effect on the refracted image, a ratio is added, where 1.0 represents no displacement. In my case I've decided to use a ratio of 0.9.

```
vec3 I = normalize(cameraPosition - Position);

R = refract(-I, normalize(newNormal), ratio);

vec4 refractColor = vec4(texture(currentSceneNormalMap, R).rgb, 1.0);
```

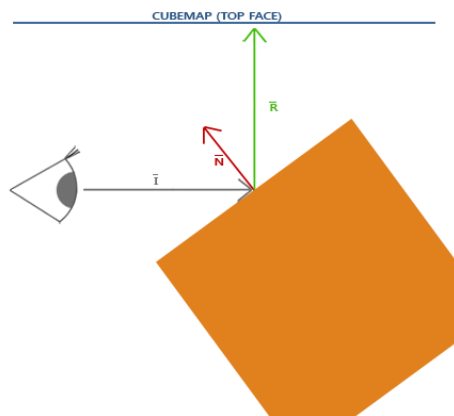




- **Reflection:**

For the reflection I again used the vector  $-I$  and the normal to calculate the color of the reflection from the skybox texture.


```
vec3 R = reflect(-I, normalize(newNormal));
vec4 reflectColor = vec4(texture(currentSceneNormalMap, R).rgb, 1.0);
```



- **Fresnel factor:**

Since the water has both reflections and refractions, it was necessary to figure out a way to know which one of them is to be shown at any moment. To do so I've used the Fresnel factor, more specifically Schlick's approximation.

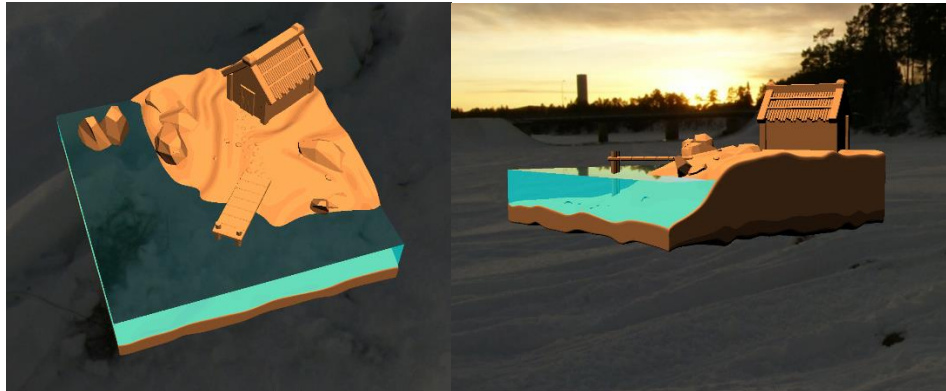
$$F = F_0 + (1 - F_0) * (1 - (\vec{n} \cdot \vec{v}))^5$$

Material	$F_0$ (Linear)	$F_0$ (sRGB)	Color
Water	(0.02, 0.02, 0.02)	(0.15, 0.15, 0.15)	

```
// Fresnel factor using Schlick's approximation
float cosTheta = dot(I, newNormal);
float F0 = 0.02; //value for water
float fresnel = F0 + (1.0 - F0) * pow(1.0 - cosTheta, 5.0);
finalColor = vec4(mix(refractColor.rgb, reflectColor.rgb, fresnel),1.0);
```

For the Fresnel effect I've used the learnopengl.com resources [4]

The end result looks like this:



- **Dynamic cubemap:**

One problem came from the water material, since all the reflections and refractions use the skybox to be calculated, the island mesh when inside or under the water is not shown when looking through it. To try and solve this, a dynamic cubemap was implemented.

In the end the solution proved to be imperfect, so for the result I removed the dynamic cubemap (the situation is better explained in the Post-Mortem). This implementation was put as an extra in the project.

A dynamic cubemap is a cubemap that is created from the objects in the scene.

In every frame, I've used a camera and recorded the image shown from the 6 different directions around it.

The images are used as textures, for the uniform, in the water shader instead of the normal skybox, to determine the reflections and refractions.

To do this, I've created a new class that takes care of the dynamic cubemap based on the position of the camera, given to it.

For this part I've used the help of ChatGPT on helping me figure out how I could take the image from the cameras' view and bind it to the uniform in the water shader.

```
void DynamicTextureCubeMap::setupCaptureViews(const glm::vec3& position) {
    captureViews[0] = glm::lookAt(position, position+ glm::vec3(1.0f, 0.0f, 0.0f), glm::vec3(0.0f, -1.0f, 0.0f)); // +X
    captureViews[1] = glm::lookAt(position, position+glm::vec3(-1.0f, 0.0f, 0.0f), glm::vec3(0.0f, -1.0f, 0.0f)); // -X
    captureViews[2] = glm::lookAt(position, position+glm::vec3(0.0f, 1.0f, 0.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // +Y
    captureViews[3] = glm::lookAt(position, position+ glm::vec3(0.0f, -1.0f, 0.0f), glm::vec3(0.0f, 0.0f, -1.0f)); // -Y
    captureViews[4] = glm::lookAt(position, position+ glm::vec3(0.0f, 0.0f, 1.0f), glm::vec3(0.0f, -1.0f, 0.0f)); // +Z
    captureViews[5] = glm::lookAt(position, position+ glm::vec3(0.0f, 0.0f, -1.0f), glm::vec3(0.0f, -1.0f, 0.0f)); // -Z
}
```

To initialize the dynamic cubemap it is allocated 6 textures with the resolution given and is set the parameters for every face. The frame and depth buffer are also created.

```

void DynamicTextureCubeMap::initializeDynamicCubemap(int resolution) {
    glGenTextures(1, &id);
    glBindTexture(GL_TEXTURE_CUBE_MAP, id);

    for (int i = 0; i < 6; ++i) {
        glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGBA, resolution, resolution, 0, GL_RGBA, GL_UNSIGNED_BYTE, nullptr);
    }

    for (const auto& p : CUBEMAP_PARAMS) {
        glTexParameteri(GL_TEXTURE_CUBE_MAP, p.name, p.value);
    }

    glGenFramebuffers(1, &framebuffer);
    glGenRenderbuffers(1, &depthBuffer);
    glBindRenderbuffer(GL_RENDERBUFFER, depthBuffer);
    glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, resolution, resolution);
    glBindRenderbuffer(GL_RENDERBUFFER, 0);

    glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, depthBuffer);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
}

```

To render the cube, it's set a projection matrix with an FOV of 90° to capture everything scene from the camera and put in the cubemap texture.

```

void DynamicTextureCubeMap::renderDynamicCubemap(const glm::vec3& position, float ratio, int resolution, std::function<void(const glm::mat4&, const glm::mat4&)> renderScene) {
    setupCaptureViews(position);
    glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);

    for (int i = 0; i < 6; ++i) {
        glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, id, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glm::mat4 projection = glm::perspective(glm::radians(90.0f), 1.0f, 0.1f, 100.0f);

        renderScene(projection, captureViews[i]);

        //std::string filename = "cubemap_face_" + std::to_string(i) + ".png";
        //saveFaceToPNG(id, GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, resolution, filename);
    }

    glBindFramebuffer(GL_FRAMEBUFFER, 0);
}

```

To render the scene, I drew the objects that I wanted to be shown in the new cubemap, which were the skybox and the small cube.

```

if (!isIslandActive) {

    glm::mat4 curViewMat = Camera->getViewMatrix();
    glm::mat4 curProjMat = Camera->getProjectionMatrix();

    currentSceneCubeMap.renderDynamicCubemap(curCameEye, ratio, 1024, [this](const glm::mat4& projection, const glm::mat4& view) {
        glViewport(0, 0, 1024, 1024);

        Camera->setProjectionMatrix(projection);
        Camera->setViewMatrix(view);

        Camera->getViewMatrix();
        Camera->getProjectionMatrix();

        drawSkybox();

        drawIslandORCube(isIslandActive);

        Shaders->unbind();
    });

    glViewport(0, 0, width, height);

    Camera->setProjectionMatrix(curProjMat);
    Camera->setViewMatrix(curViewMat);
}

```

After this, the meshes are drawn again like before, and before drawing the water the new cubemap is binded to use it in the shader.

There were a lot of different camera and island positions that were tested, but unfortunately none looked good enough to include in the end result, that is the reason to only have it as an extra in the project.

### **3. Post-Mortem**

#### **3.1.What went well?**

I believe that despite the difficulties, the results came out looking like what I had in mind, so I'm happy with the result as it aligns with my inspirations.

#### **3.2. What did not go so well?**

In my opinion, what did not go so well was the transparent material.

At the beginning, it didn't cross my mind that by using a skybox for the reflections and refractions the island part that was inside, the water would not show.

When making my project proposal document, I've specified that I would want a rock on the bottom of the water to show the refraction effect, this proved way more difficult than I expected.

In the end, I'm happy with my overall solution, even though it is not what I had intended in the beginning, with the challenges that the material posed, I believe I was able to implement a good middle ground solution.

My final solution was to have the top of the water behaving like the transparent material, and the rest of the mesh being only a transparent material, by reducing its alpha component, this way the island is still visible, but the water doesn't have any displacement of the reflected image.

During the project development, I discussed with my TA, the best solution for this problem, that's where I learned about dynamic cubemaps.

This seemed like the perfect solution for my problem but turned out to be more difficult to implement correctly.

The problem came from the fact that if I've tried to implement the displacement of the image, from the refraction the island would be too out of place and would not look good.

Various configurations were tested, but none seemed to work perfectly.

To not completely discard the work done, I've implemented different modes in my project. It starts with the "final solution" chosen, and by clicking "B" the mode changes to show the dynamic cubemap. It includes a small cube rotating around the water since to show the effect, this solves one of the problems which came from having the island inside the water.

#### **3.3.Lessons learned**

If I could go back to the beginning of the project, I would have taken more time while researching for the project's proposal, to try and understand more in depth the limitations of what I was suggesting, more specifically the water's transparent material.

This way I could have learned about dynamic cubemaps earlier and would have had more time to correctly implement it.

## References

- [1] <https://research.ncl.ac.uk/game/mastersdegree/graphicsforgames/scenegraphs/Tutorial%206%20-%20Scene%20Graphs.pdf>
- [2] <https://learnopengl.com/Advanced-Lighting/Normal-Mapping>
- [3] <https://learnopengl.com/Advanced-OpenGL/Cubemaps>
- [4] <https://learnopengl.com/PBR/Theory>
- [5] <https://www.lighthouse3d.com/tutorials/glsl-12-tutorial/toon-shader-version-ii/>

ChatGPT was used in the creation of the class for the Dynamic cubemap.