CSCI3260
Principles of Computer Graphics

Tutorial 9

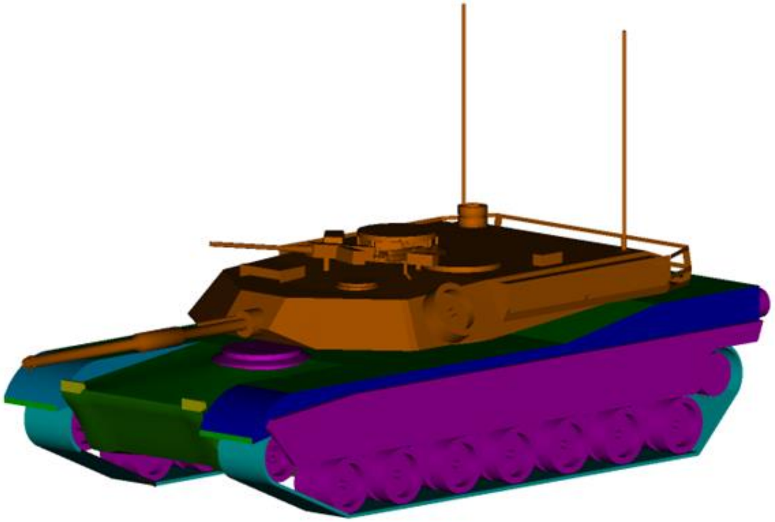
Xin Yang

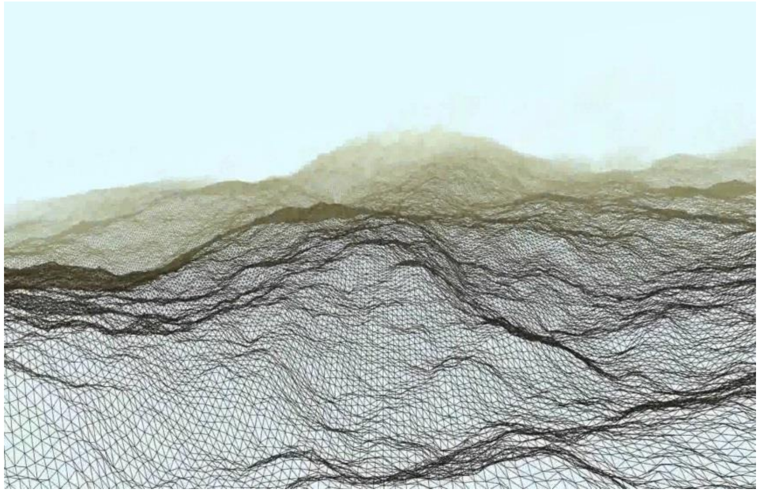
Outline

- Multiple Texture Mapping
- Multiple Shader
- Skybox

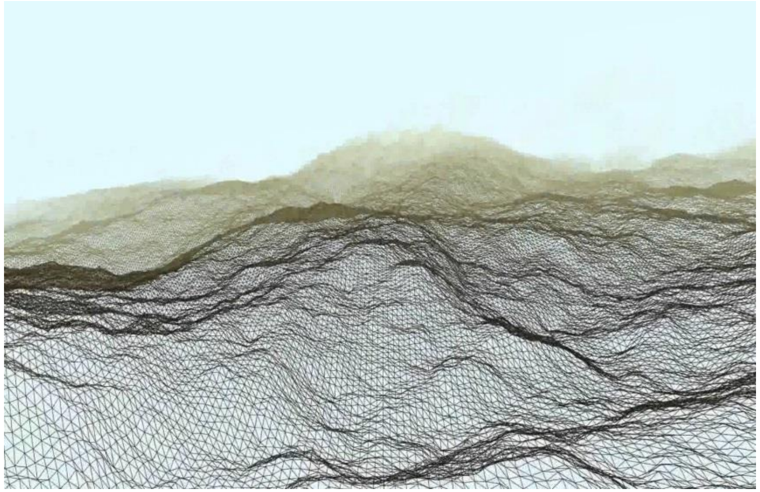
Note: tutorials from today are necessary basic requirements of final project.

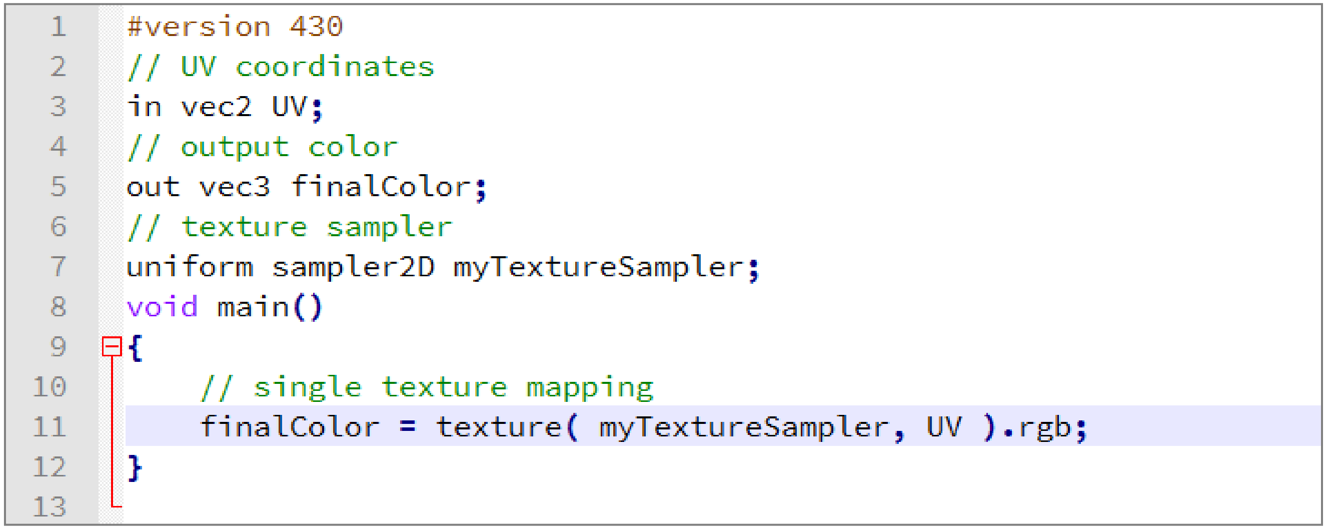
Multiple Texture Mapping





Multiple Texture Mapping





```
#version 430
// UV coordinates
in vec2 UV;
// output color
out vec3 finalColor;
// texture sampler
uniform sampler2D myTextureSampler;
void main()
{
    // single texture mapping
    finalColor = texture( myTextureSampler, UV ).rgb;
}

```

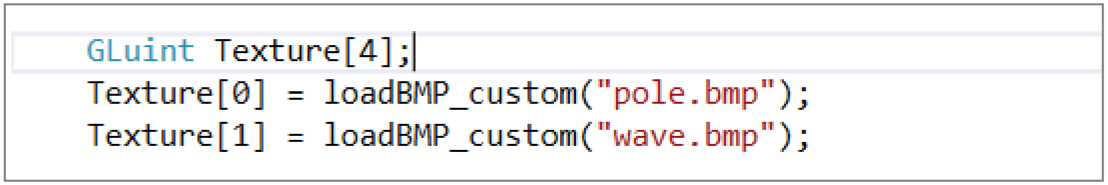
Multiple Texture Mapping

- Allow us to use multiple textures in one fragment shader
- Assign a *location* value (texture unit) to the texture sampler
- OpenGL has at least 16 texture units (GL_TEXTURE0 to GL_TEXTURE15)

1. Generate a texture
2. Active a texture unit
3. Bind texture to the activated texture unit
4. Assign the texture unit to a texture sampler
5. (Shader) Combine the results from multiple texture mapping

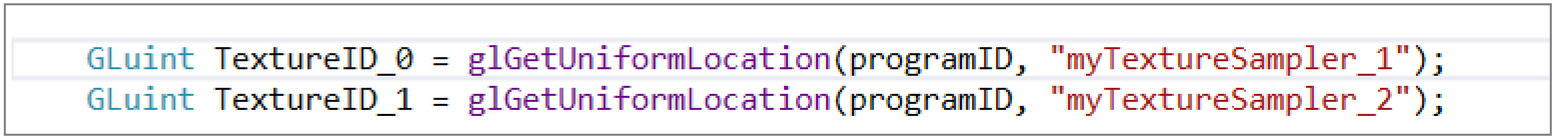
Multiple Texture Mapping

Create texture



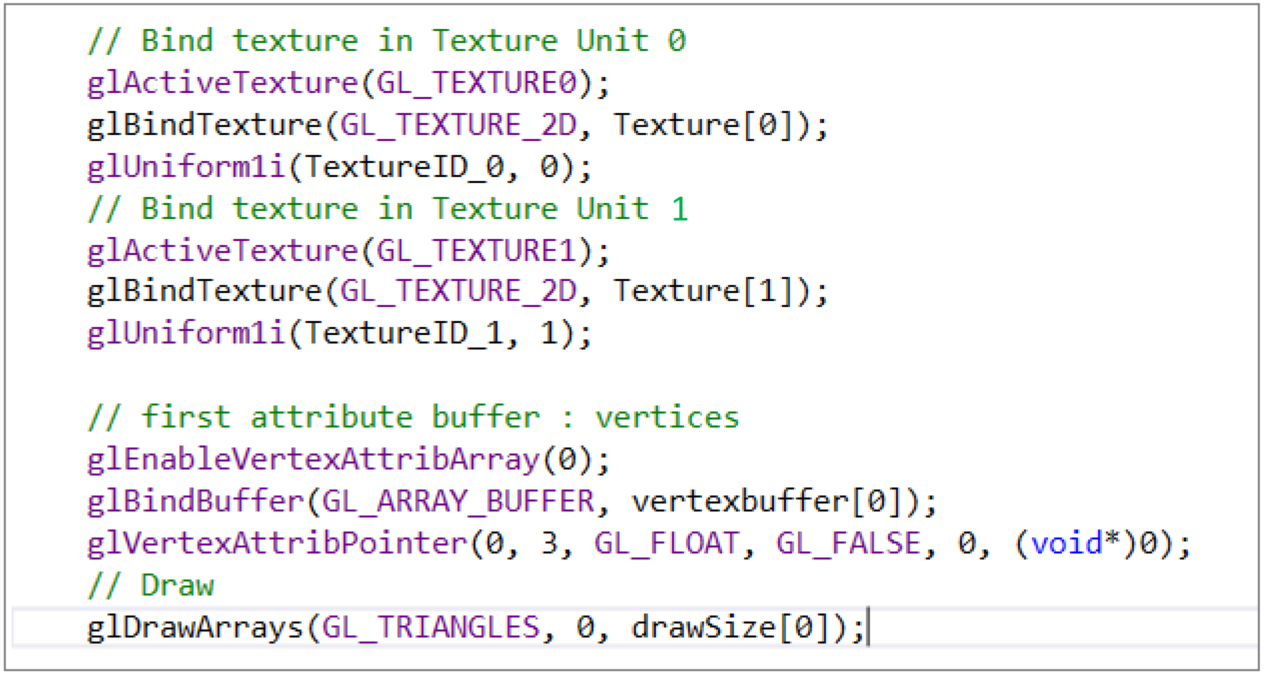
```
    GLuint Texture[4];
    Texture[0] = loadBMP_custom("pole.bmp");
    Texture[1] = loadBMP_custom("wave.bmp");
```

Create texture sampler



```
    GLuint TextureID_0 = glGetUniformLocation(programID, "myTextureSampler_1");
    GLuint TextureID_1 = glGetUniformLocation(programID, "myTextureSampler_2");
```

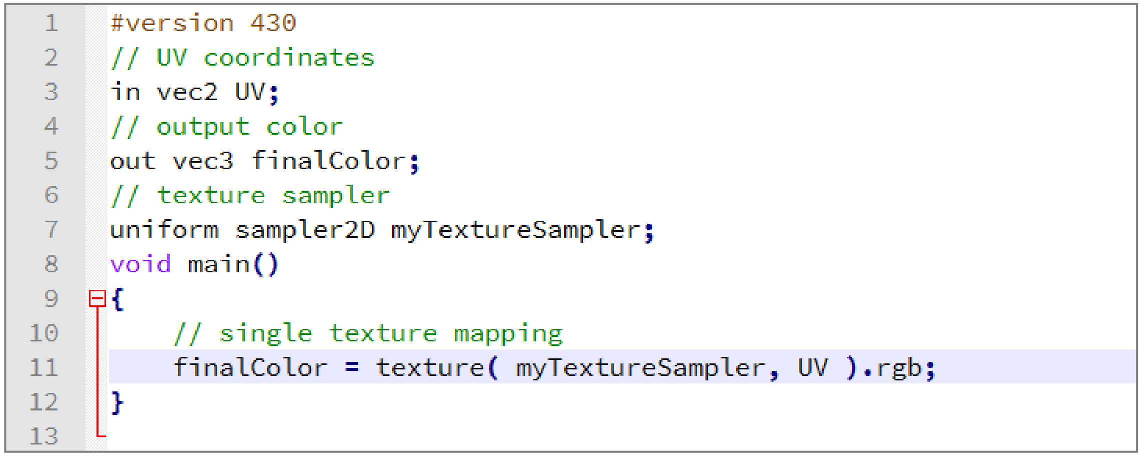
Active texture units and bind several textures



```
    // Bind texture in Texture Unit 0
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, Texture[0]);
    glUniform1i(TextureID_0, 0);
    // Bind texture in Texture Unit 1
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, Texture[1]);
    glUniform1i(TextureID_1, 1);

    // first attribute buffer : vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer[0]);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
    // Draw
    glDrawArrays(GL_TRIANGLES, 0, drawSize[0]);
```

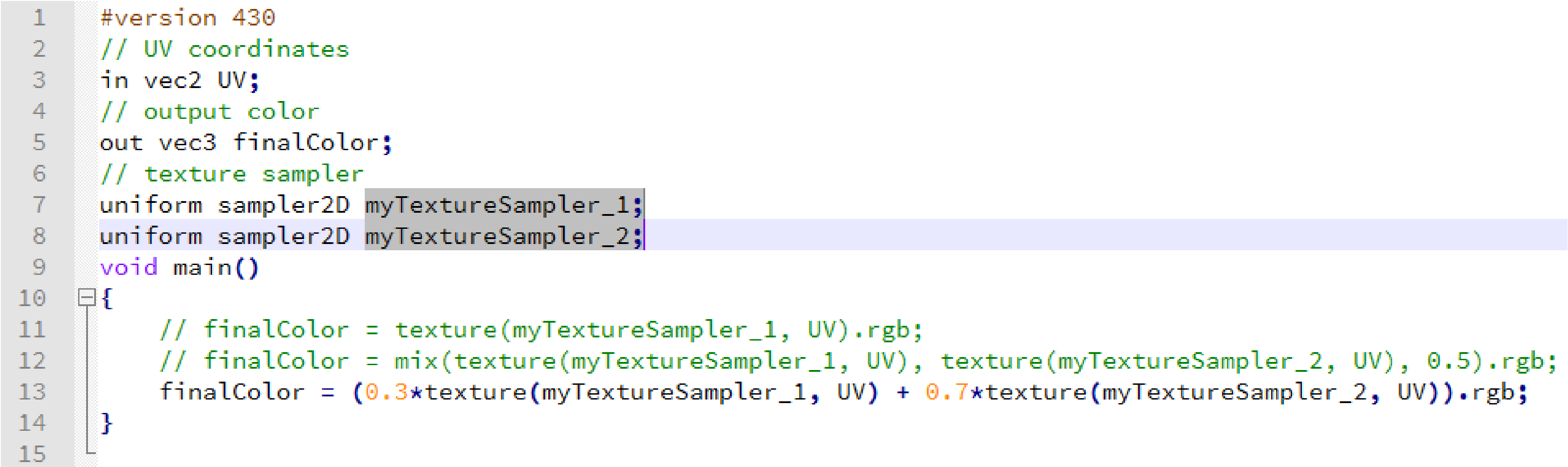
Multiple Texture Mapping

Fragment shader for single texture mapping



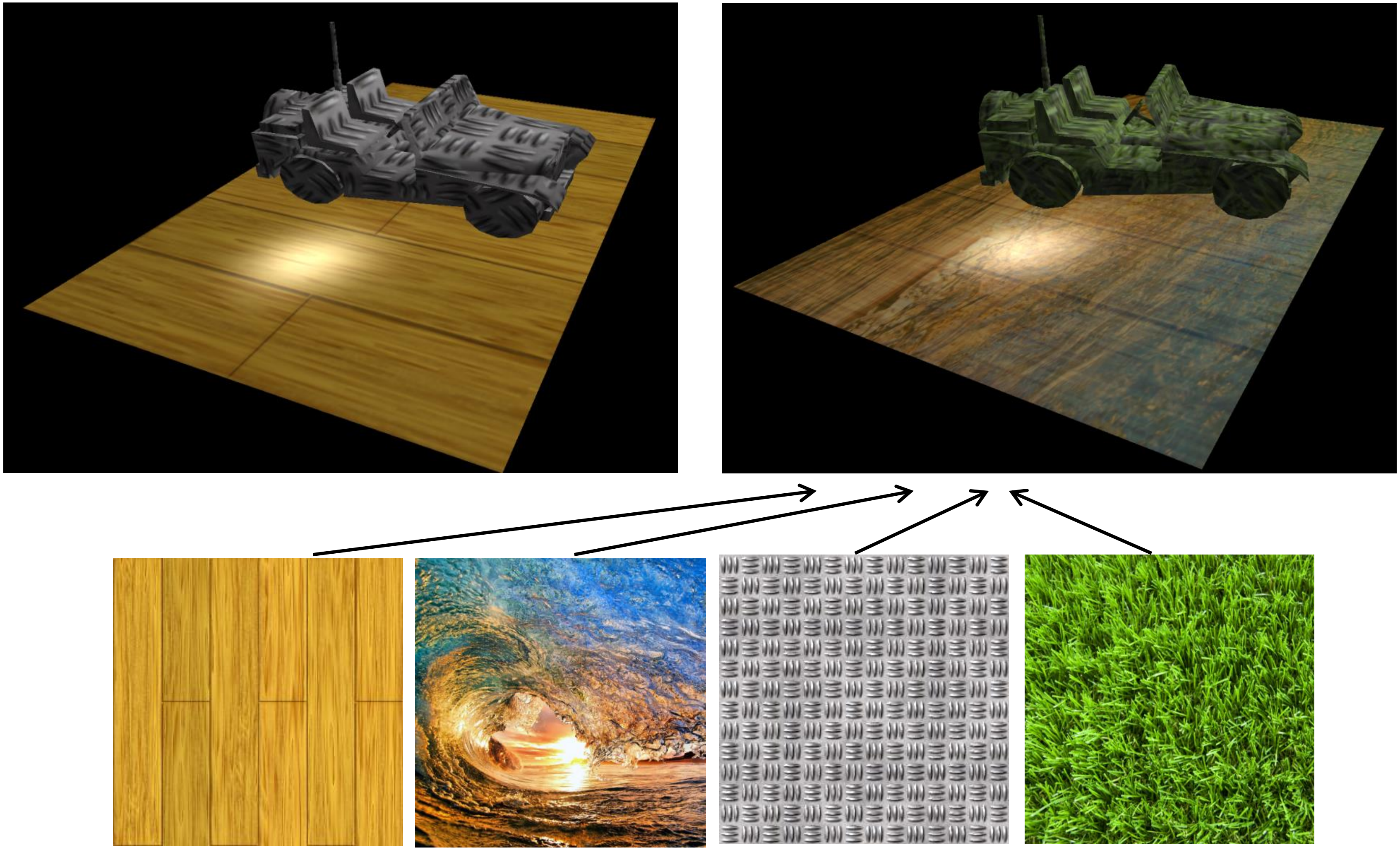
```
#version 430
// UV coordinates
in vec2 UV;
// output color
out vec3 finalColor;
// texture sampler
uniform sampler2D myTextureSampler;
void main()
{
    // single texture mapping
    finalColor = texture( myTextureSampler, UV ).rgb;
}

```

Fragment shader for multiple texture mapping



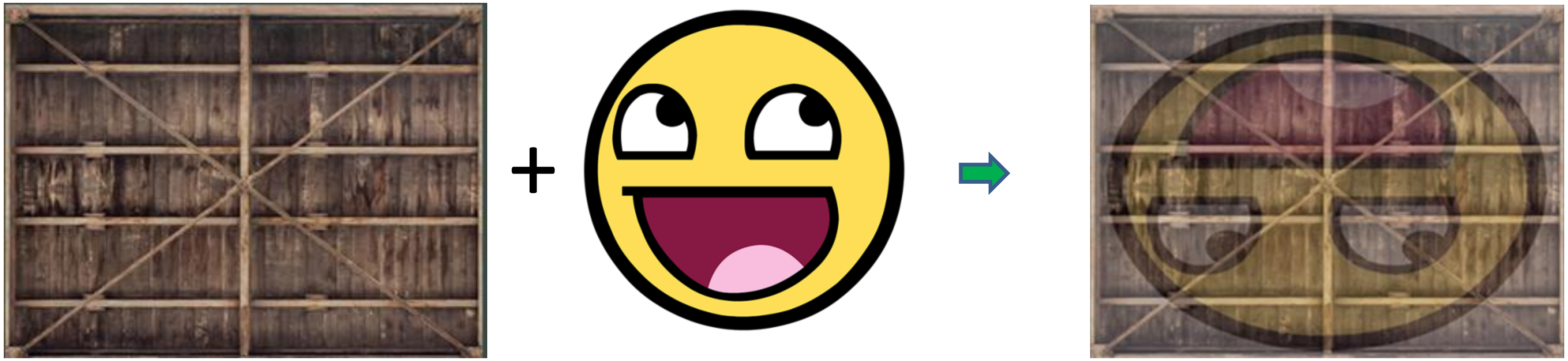
```
#version 430
// UV coordinates
in vec2 UV;
// output color
out vec3 finalColor;
// texture sampler
uniform sampler2D myTextureSampler_1;
uniform sampler2D myTextureSampler_2;
void main()
{
    // finalColor = texture(myTextureSampler_1, UV).rgb;
    // finalColor = mix(texture(myTextureSampler_1, UV), texture(myTextureSampler_2, UV), 0.5).rgb;
    finalColor = (0.3*texture(myTextureSampler_1, UV) + 0.7*texture(myTextureSampler_2, UV)).rgb;
}

```

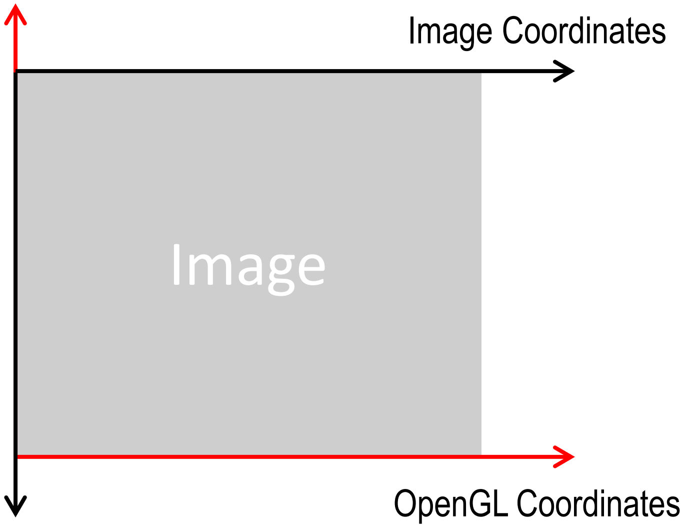
Multiple Texture Mapping



Multiple Texture Mapping

Note: direction problem





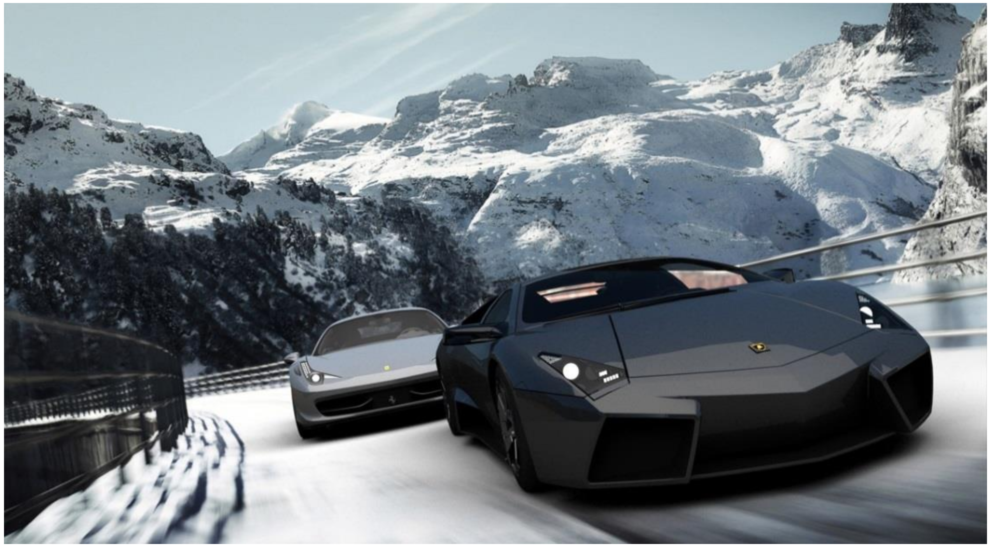
Solution:

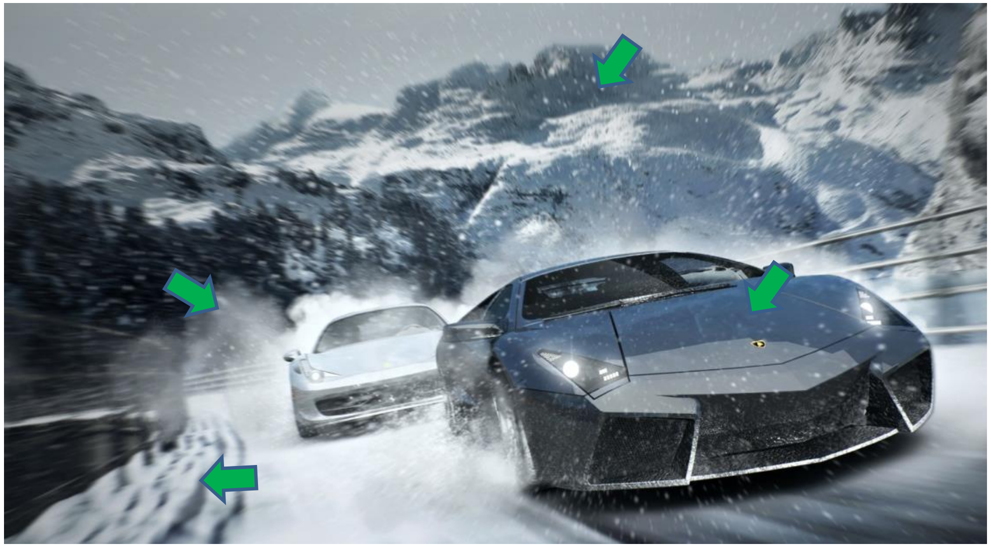
1. alter the texture coordinates data.
2. edit the vertex shader to swap the y-coordinate automatically.

Multiple Texture Mapping

Play with demo

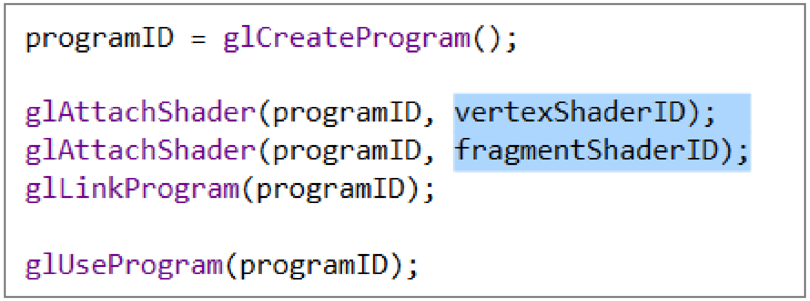
Multiple Shader



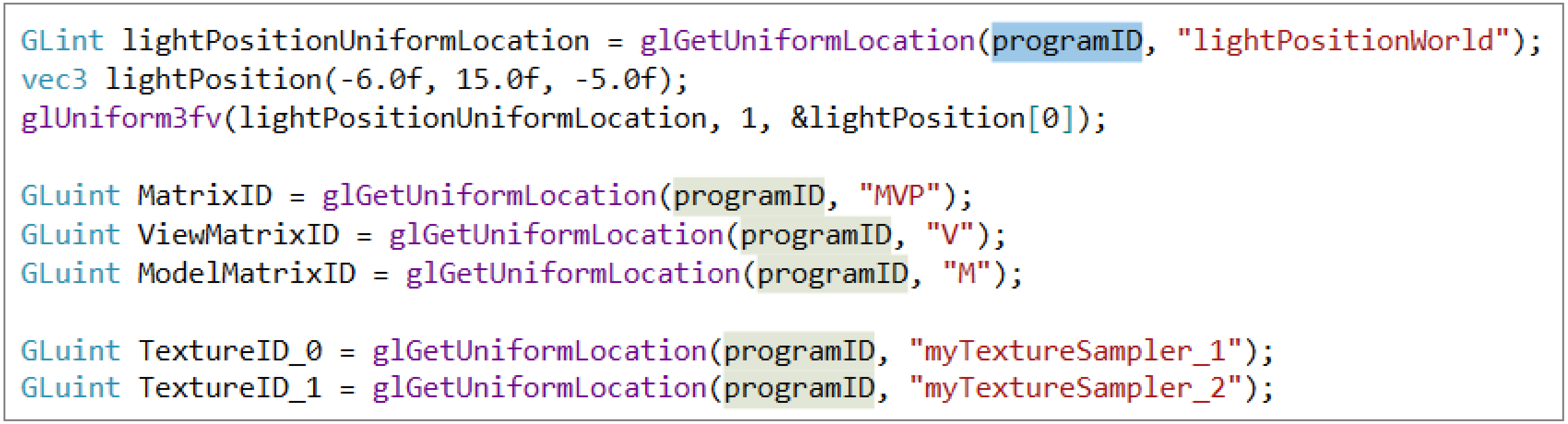


Multiple Shader

- ➢ Many different rendering requirements exist
- ➢ One common vertex/fragment shader cannot satisfy all needs
- ➢ Shaders are associated with shader program object



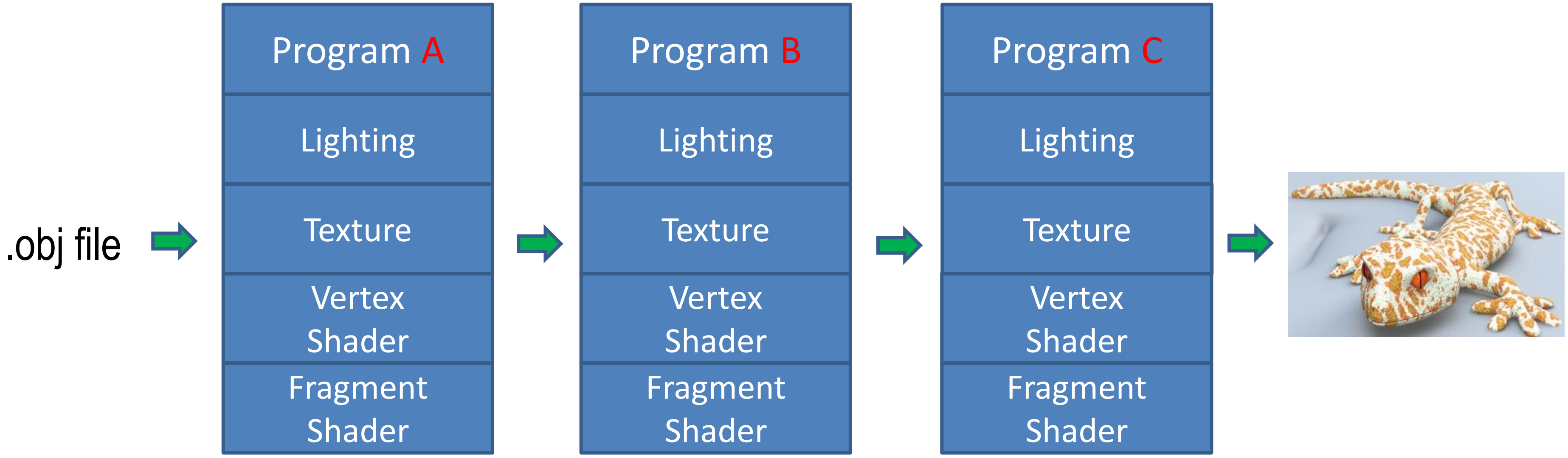
```
programID = glCreateProgram();

glAttachShader(programID, vertexShaderID);
glAttachShader(programID, fragmentShaderID);
glLinkProgram(programID);

glUseProgram(programID);
```



```
GLint lightPositionUniformLocation = glGetUniformLocation(programID, "lightPositionWorld");
vec3 lightPosition(-6.0f, 15.0f, -5.0f);
glUniform3fv(lightPositionUniformLocation, 1, &lightPosition[0]);

GLuint MatrixID = glGetUniformLocation(programID, "MVP");
GLuint ViewMatrixID = glGetUniformLocation(programID, "V");
GLuint ModelMatrixID = glGetUniformLocation(programID, "M");

GLuint TextureID_0 = glGetUniformLocation(programID, "myTextureSampler_1");
GLuint TextureID_1 = glGetUniformLocation(programID, "myTextureSampler_2");
```

Multiple Shader

- Lighting, texture, MVP Matrix and shader modules are optional
- Share the same module or not depends on yourself
- Enable your Program with "glUseProgram(Program_ID)" before use the program.

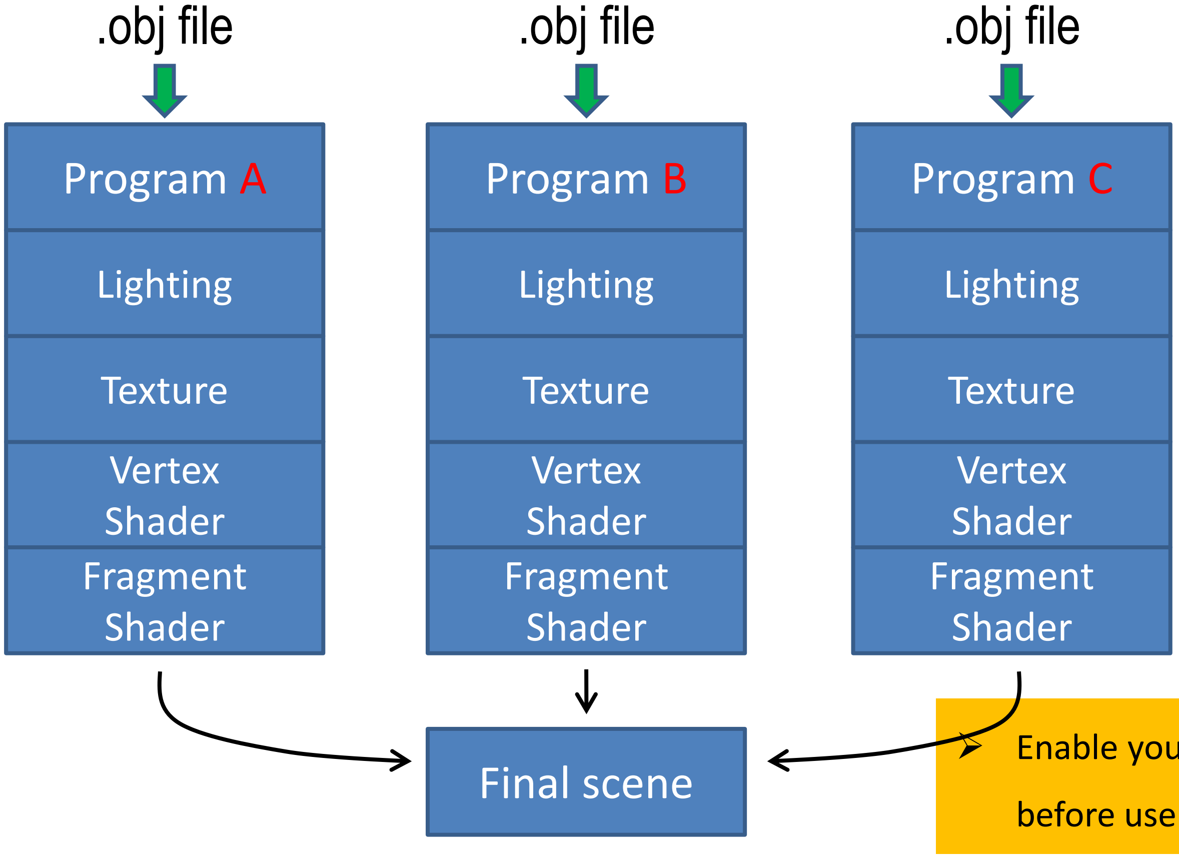


.obj file →	Program A →	Program B →	Program C →
	Lighting	Lighting	Lighting
	Texture	Texture	Texture
	Vertex Shader	Vertex Shader	Vertex Shader
	Fragment Shader	Fragment Shader	Fragment Shader
example:	Texture	Displacement	Shadow

Render one object with different shaders (Sequential)

Multiple Shader

- Lighting, texture, MVP Matrix and shader modules are optional
- Share the same module or not depends on needs



.obj file

Program A

Lighting

Texture

Vertex Shader

Fragment Shader

.obj file

Program B

Lighting

Texture

Vertex Shader

Fragment Shader

.obj file

Program C

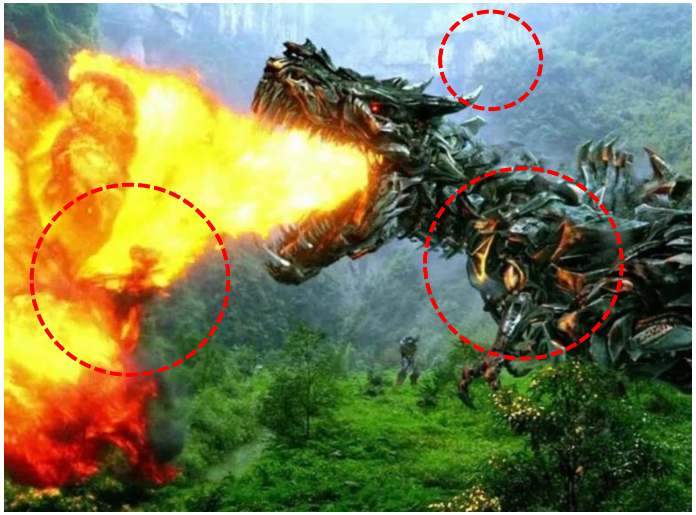
Lighting

Texture

Vertex Shader

Fragment Shader

Final scene



- Enable your Program with "glUseProgram (Program_ID)" before use the program.

Render different objects with different shaders

Multiple Shader

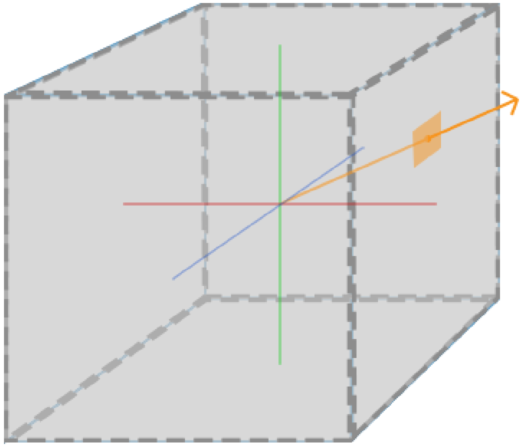
Demo code in the last

Skybox

- What is skybox?
- The realistic scene around that you can never arrive

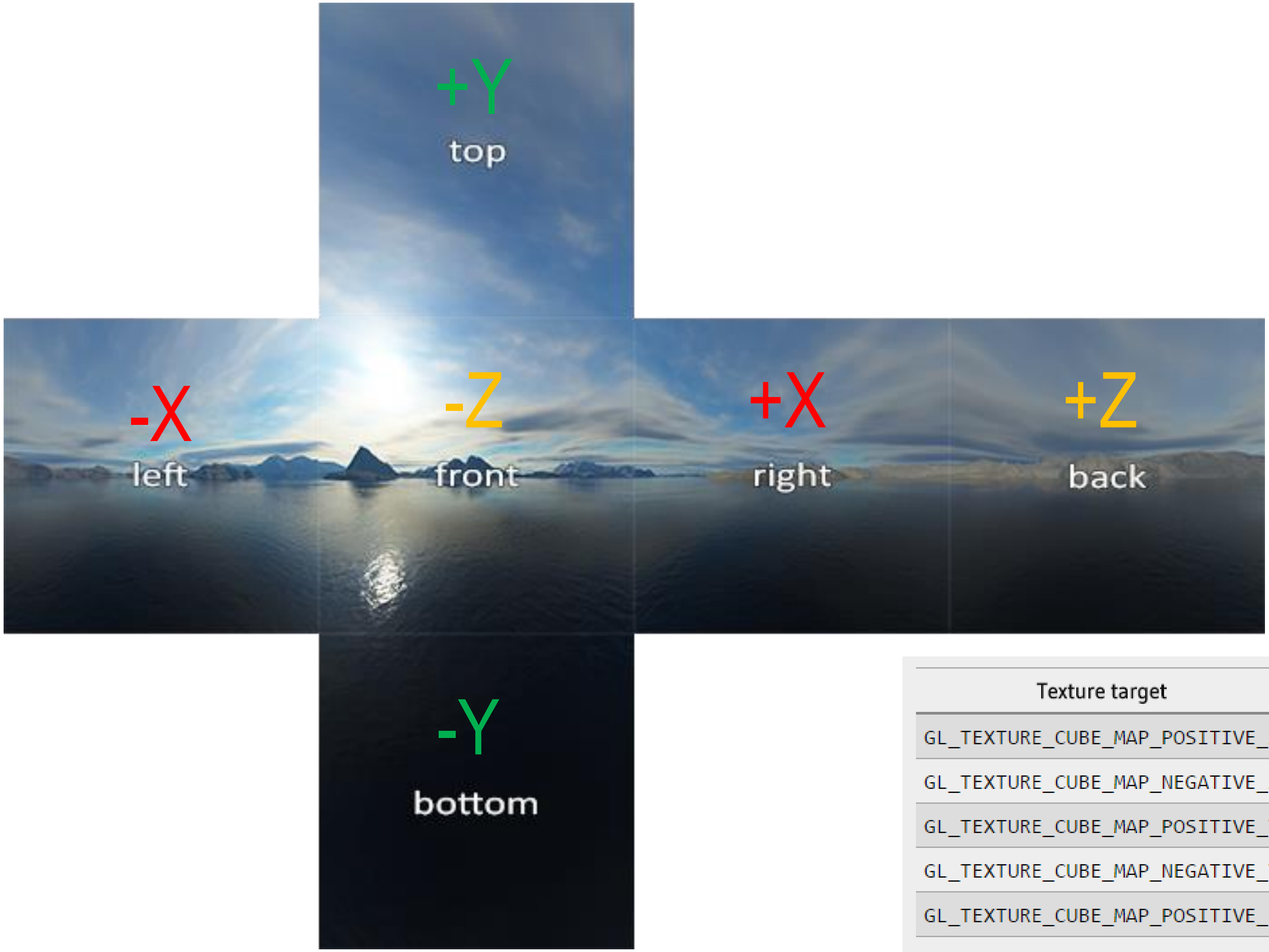
Skybox

- Map multiple textures into a single texture: Cubemap
- 6 individual 2D textures that each form one side of a cube
- Retrieve the texture coordinates of all vertices as the vertex positions of the cube (no extra UV coordinates)



- Skybox image sources here:
- http://www.custommapmakers.org/skyboxes.php
- http://www.humus.name/index.php?page=Textures&start=0

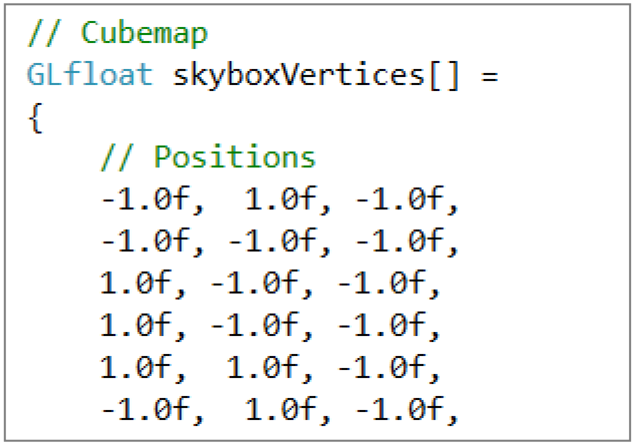
Skybox



Texture target	Orientation
GL_TEXTURE_CUBE_MAP_POSITIVE_X	Right
GL_TEXTURE_CUBE_MAP_NEGATIVE_X	Left
GL_TEXTURE_CUBE_MAP_POSITIVE_Y	Top
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y	Bottom
GL_TEXTURE_CUBE_MAP_POSITIVE_Z	Back
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z	Front

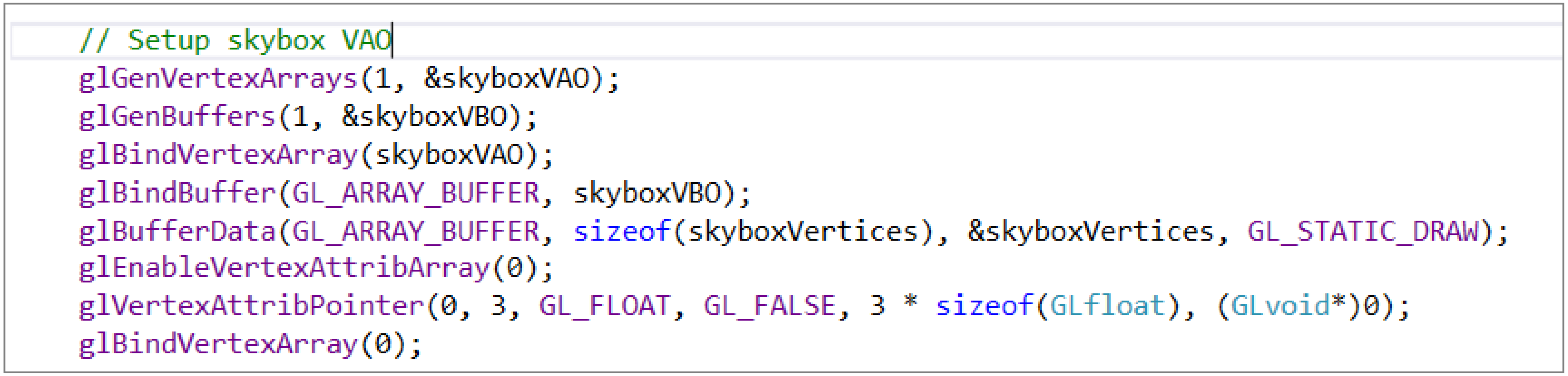
Skybox

Create cube vertex data



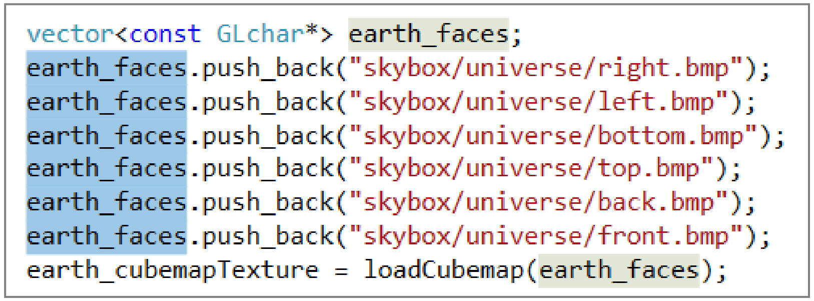
```
// Cubemap
GLfloat skyboxVertices[] =
{
    // Positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
```

Send cube vertex data to OpenGL

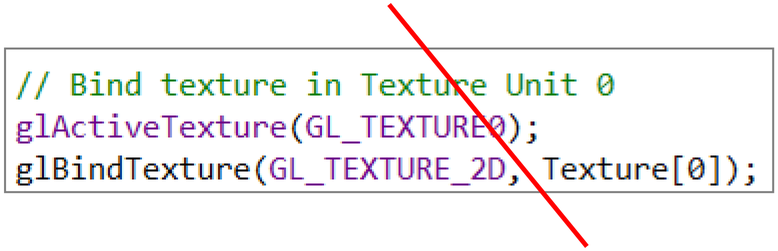


```
    // Setup skybox VAO
    glGenVertexArrays(1, &skyboxVAO);
    glGenBuffers(1, &skyboxVBO);
    glBindVertexArray(skyboxVAO);
    glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
    glBindVertexArray(0);
```

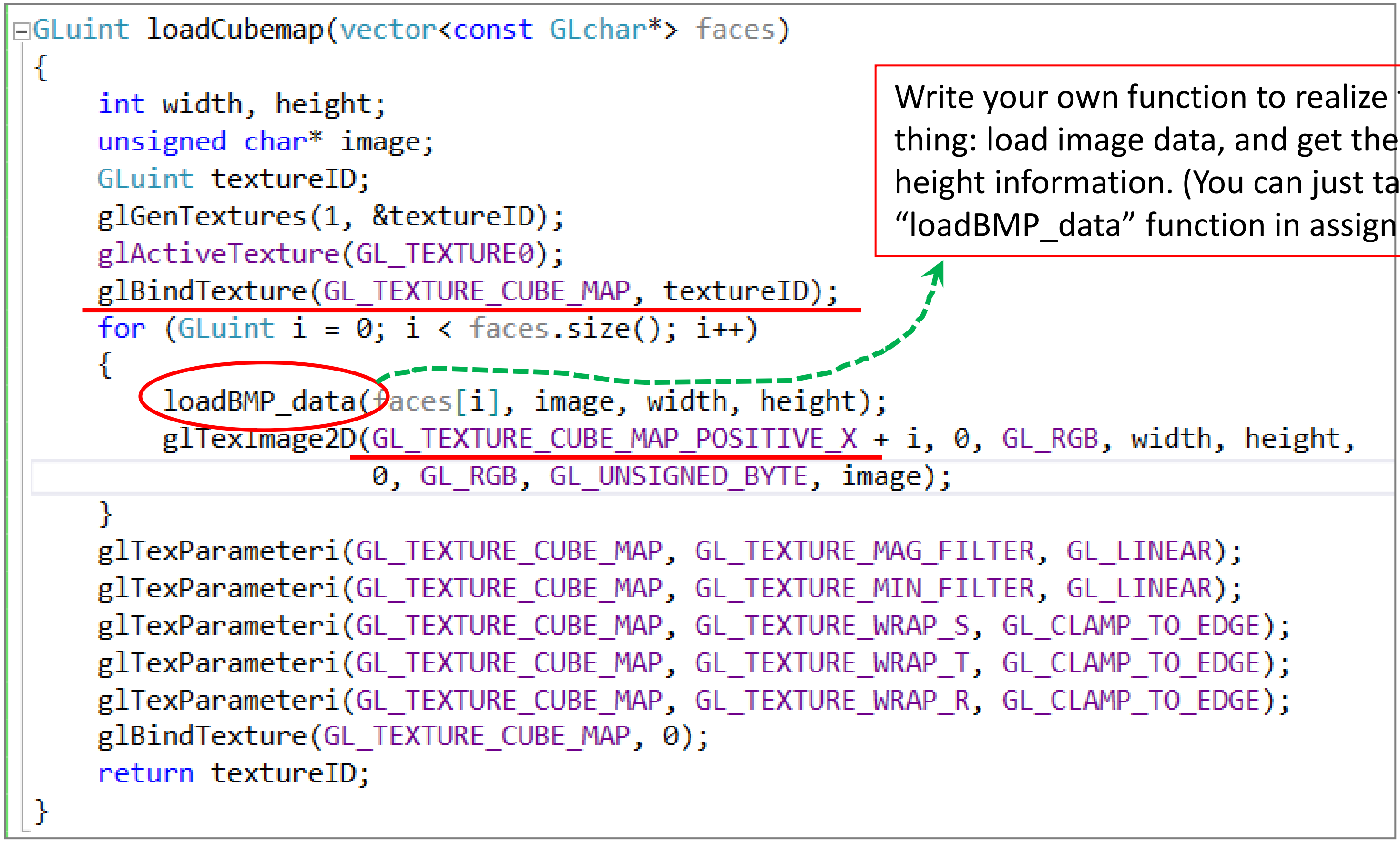
Skybox



```
vector<const GLchar*> earth_faces;
earth_faces.push_back("skybox/universe/right.bmp");
earth_faces.push_back("skybox/universe/left.bmp");
earth_faces.push_back("skybox/universe/bottom.bmp");
earth_faces.push_back("skybox/universe/top.bmp");
earth_faces.push_back("skybox/universe/back.bmp");
earth_faces.push_back("skybox/universe/front.bmp");
earth_cubemapTexture = loadCubemap(earth_faces);
```



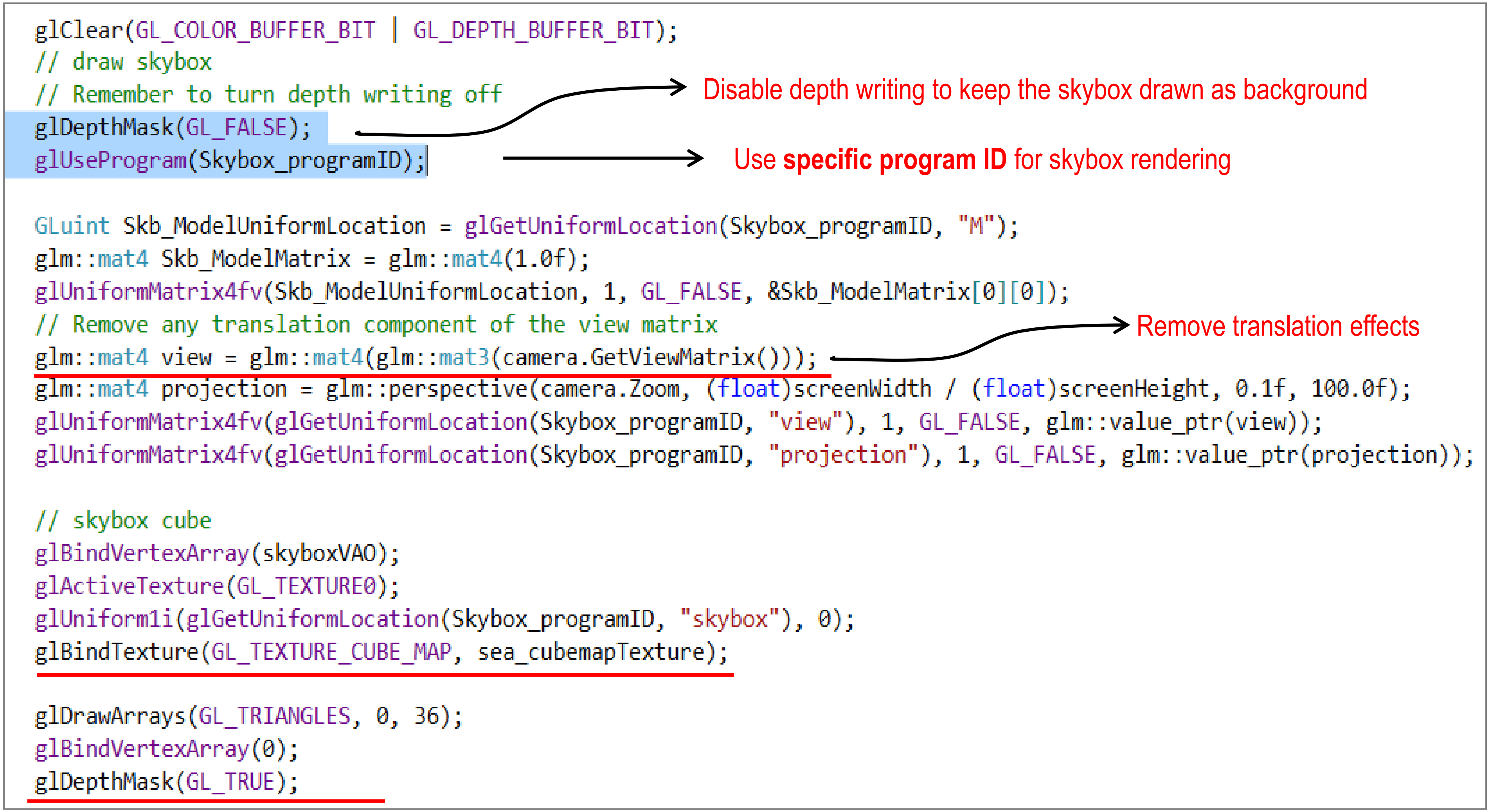
```
// Bind texture in Texture Unit 0
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, Texture[0]);
```



```
GLuint loadCubemap(vector<const GLchar*> faces)
{
    int width, height;
    unsigned char* image;
    GLuint textureID;
    glGenTextures(1, &textureID);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
    for (GLuint i = 0; i < faces.size(); i++)
    {
        loadBMP_data(faces[i], image, width, height);
        glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGB, width, height,
                     0, GL_RGB, GL_UNSIGNED_BYTE, image);
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
    glBindTexture(GL_TEXTURE_CUBE_MAP, 0);
    return textureID;
}
```

Write your own function to realize the same thing: load image data, and get the width and height information. (You can just tailor the “loadBMP_data” function in assignment 2)

Skybox



```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// draw skybox
// Remember to turn depth writing off
glDepthMask(GL_FALSE);
glUseProgram(Skybox_programID);

GLuint Skb_ModelUniformLocation = glGetUniformLocation(Skybox_programID, "M");
glm::mat4 Skb_ModelMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(Skb_ModelUniformLocation, 1, GL_FALSE, &Skb_ModelMatrix[0][0]);
// Remove any translation component of the view matrix
glm::mat4 view = glm::mat4(glm::mat3(camera.GetViewMatrix()));
glm::mat4 projection = glm::perspective(camera.Zoom, (float)screenWidth / (float)screenHeight, 0.1f, 100.0f);
glUniformMatrix4fv(glGetUniformLocation(Skybox_programID, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(Skybox_programID, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

// skybox cube
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE0);
glUniform1i(glGetUniformLocation(Skybox_programID, "skybox"), 0);
glBindTexture(GL_TEXTURE_CUBE_MAP, sea_cubemapTexture);

glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthMask(GL_TRUE);
```

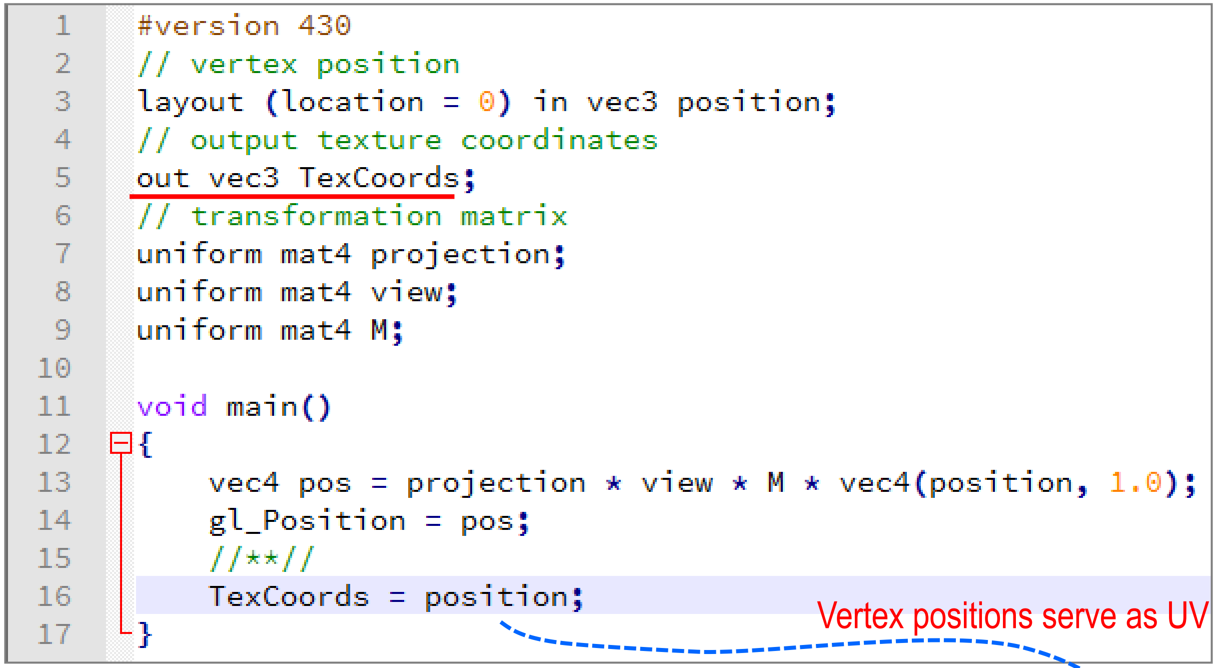
Disable depth writing to keep the skybox drawn as background

Use **specific program ID** for skybox rendering

Remove translation effects

Skybox

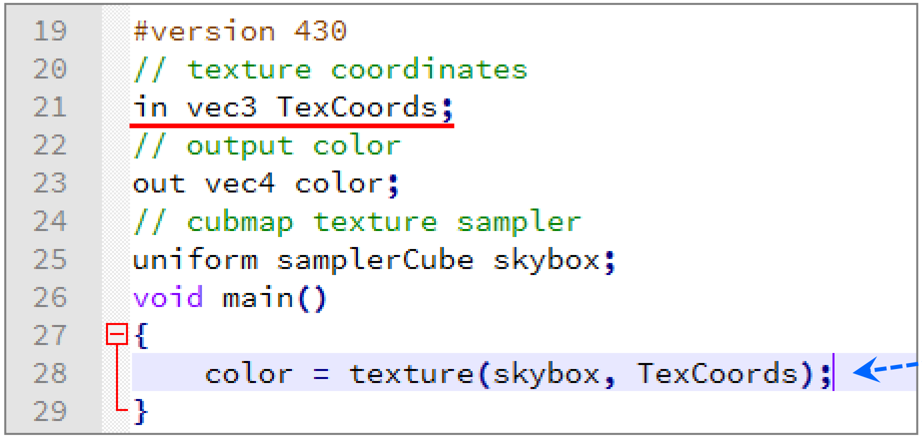
Vertex shader



```
#version 430
// vertex position
layout (location = 0) in vec3 position;
// output texture coordinates
out vec3 TexCoords;
// transformation matrix
uniform mat4 projection;
uniform mat4 view;
uniform mat4 M;

void main()
{
    vec4 pos = projection * view * M * vec4(position, 1.0);
    gl_Position = pos;
    //**//
    TexCoords = position;
}
```

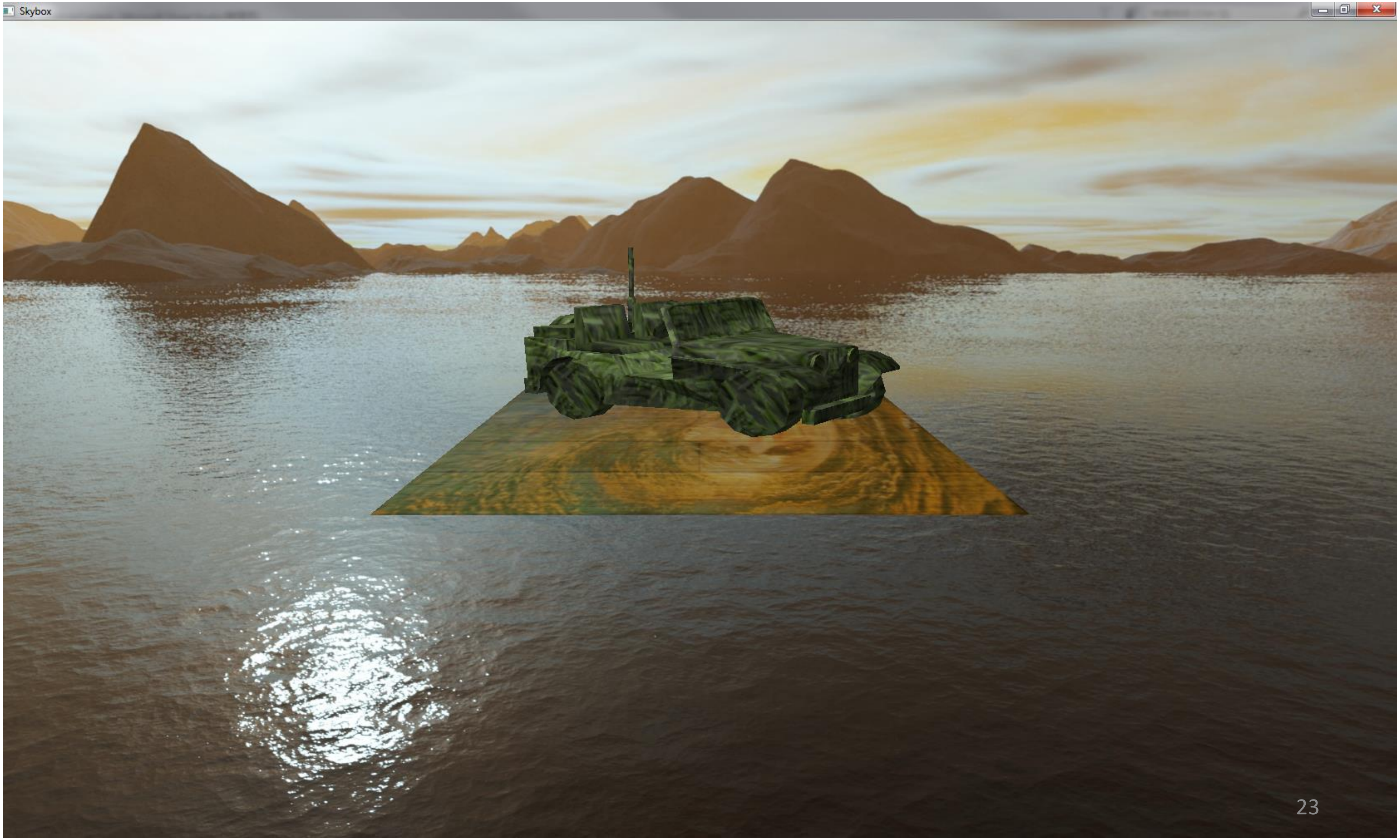
Vertex positions serve as UV coordinates

Fragment shader



```
#version 430
// texture coordinates
in vec3 TexCoords;
// output color
out vec4 color;
// cubmap texture sampler
uniform samplerCube skybox;
void main()
{
    color = texture(skybox, TexCoords);
}
```

Skybox



Skybox

Play with demo

More details can be found: http://learnopengl.com/#!Advanced-OpenGL/Cubemaps
Especially the **github** code: https://github.com/JoeyDeVries/LearnOpenGL