HW3: Texture Mapping

Introduction to Computer Graphics

Yu-Ting Wu

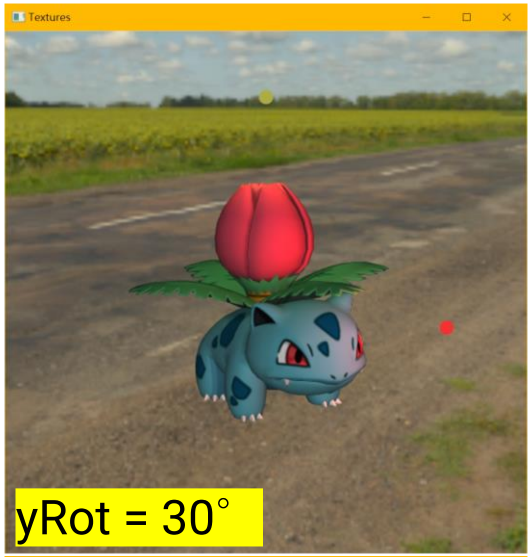
HW Description

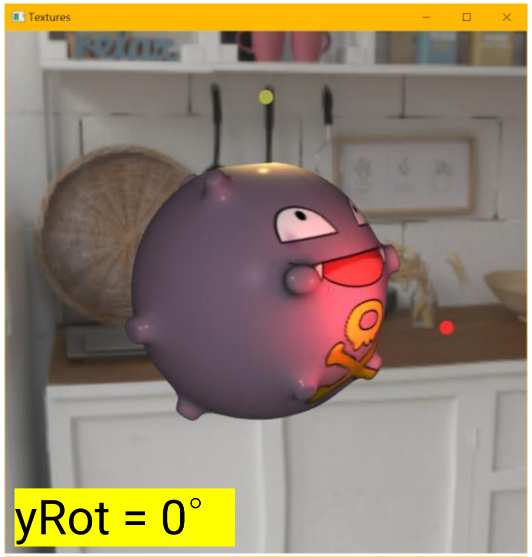
- **Web Link:**
  - https://kevincosner.github.io/courses/ICG2022/hw3.html
- **Major Task**
  - Add texture support to your HW2
    - Load image textures for spatially-varying diffuse coefficients
- **Minor Task**
  - Understand the implementation of skybox and add some functions

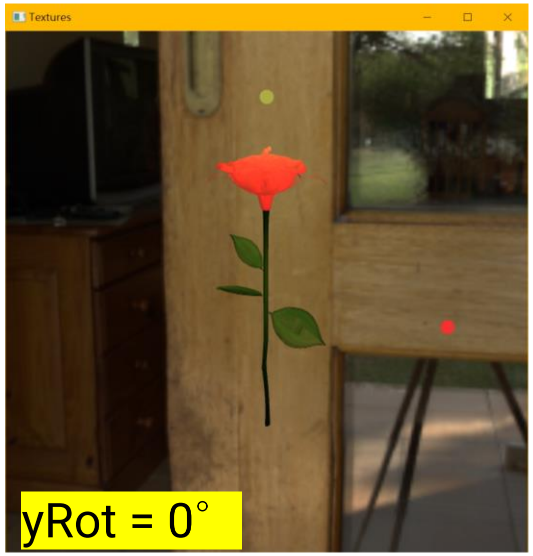
Grading Policy

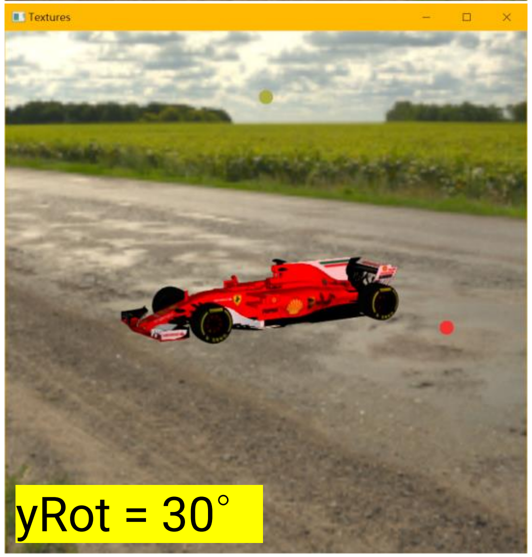
- Load and store the texture data correctly (60%) [Test Models]
- Add texture support to your Phong shading implementation (HW2)
  - At each fragment, lookup the image texture to determine the diffuse coefficient ***Kd*** (20%)
- Code organization (10%)
- Report (5%)
  - Introduce your implementation and put some screenshots
- Bonus (5%)
  - Dynamically change the skybox textures (load/release)
  - Ability to rotate the skybox (with keyboard or mouse)

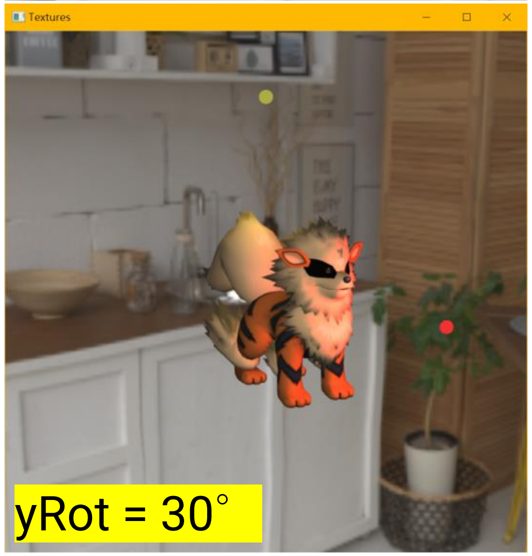
Reference Results

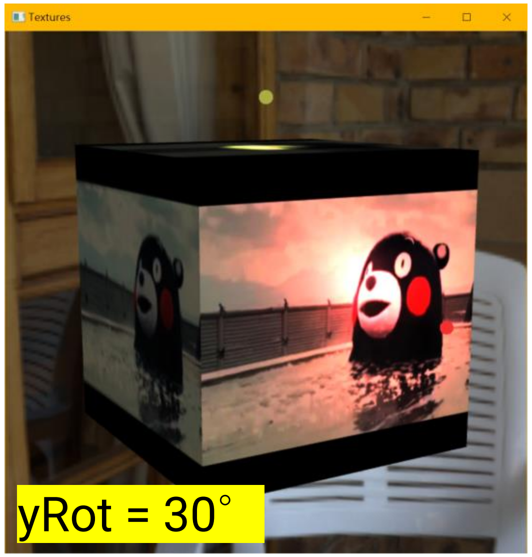












Submission

- **Deadline: Jan. 04, 2023 (PM 11:59)**

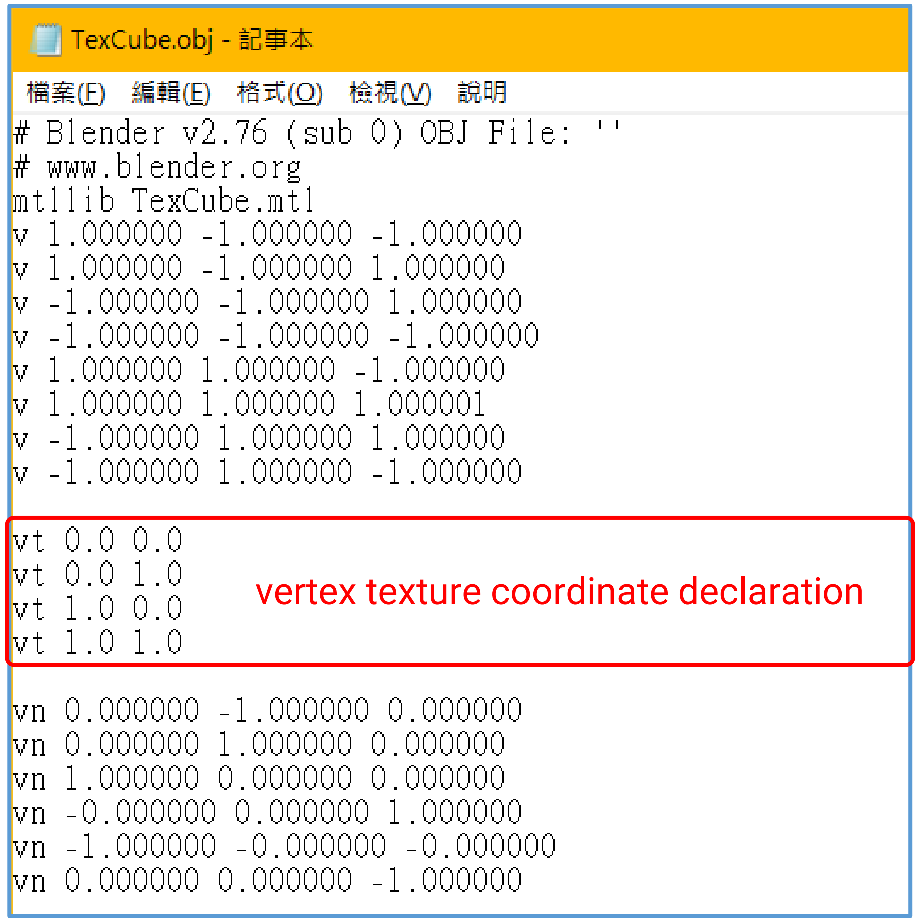
- **Submission rule**
  - The same as HW1 and HW2

- **Late policy**

One day	90%
Two days	80%
Three days	70%
Four days	60%
Five days+	50%

Texture Data in Wavefront OBJ File

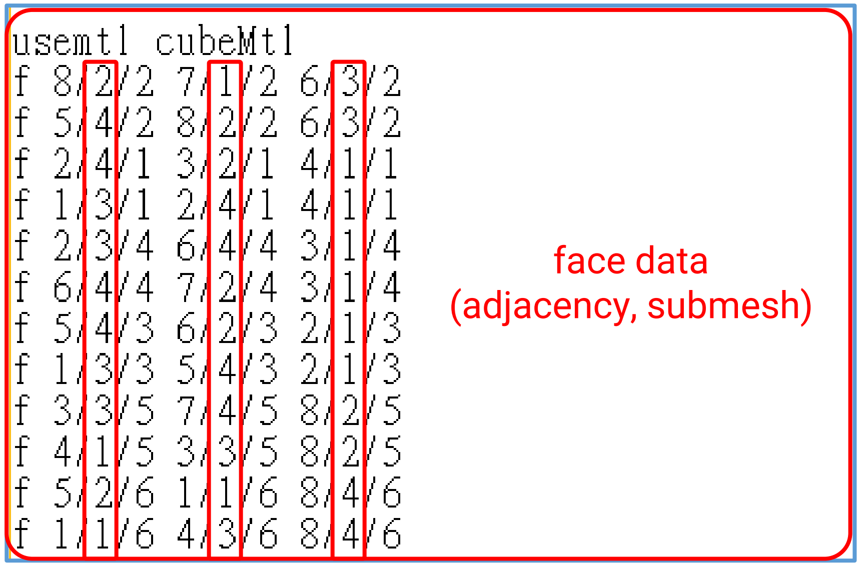
- TexCube.obj



```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
mtllib TexCube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000

vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0

vn 0.000000 -1.000000 0.000000
vn 0.000000 1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -0.000000 0.000000 1.000000
vn -1.000000 -0.000000 -0.000000
vn 0.000000 0.000000 -1.000000
```

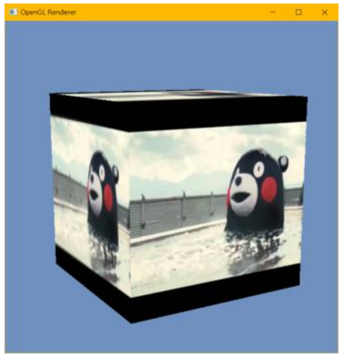
vertex texture coordinate declaration

f P/T/N P/T/N P/T/N

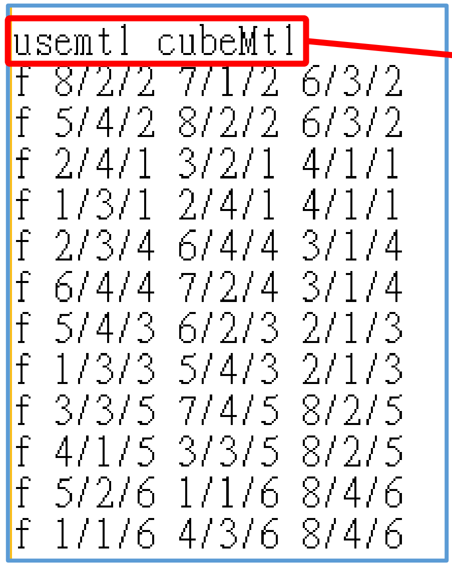


```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

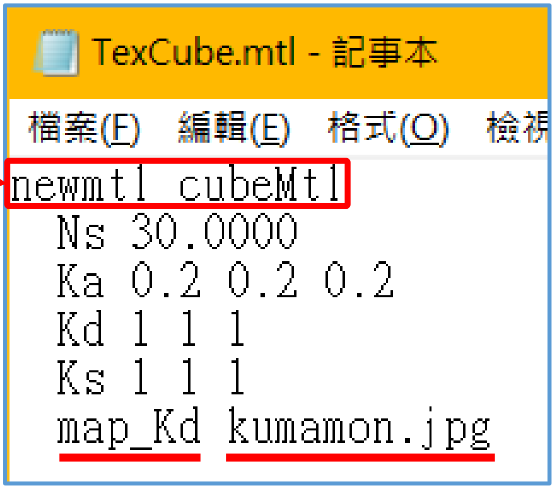
face data
(adjacency, submesh)



Texture Data in Wavefront OBJ File (cont.)

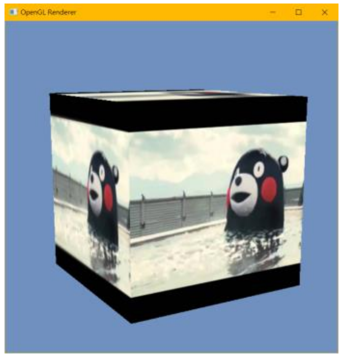


```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```



```
newmtl cubeMtl
  Ns 30.0000
  Ka 0.2 0.2 0.2
  Kd 1 1 1
  Ks 1 1 1
  map_Kd kumamon.jpg
```

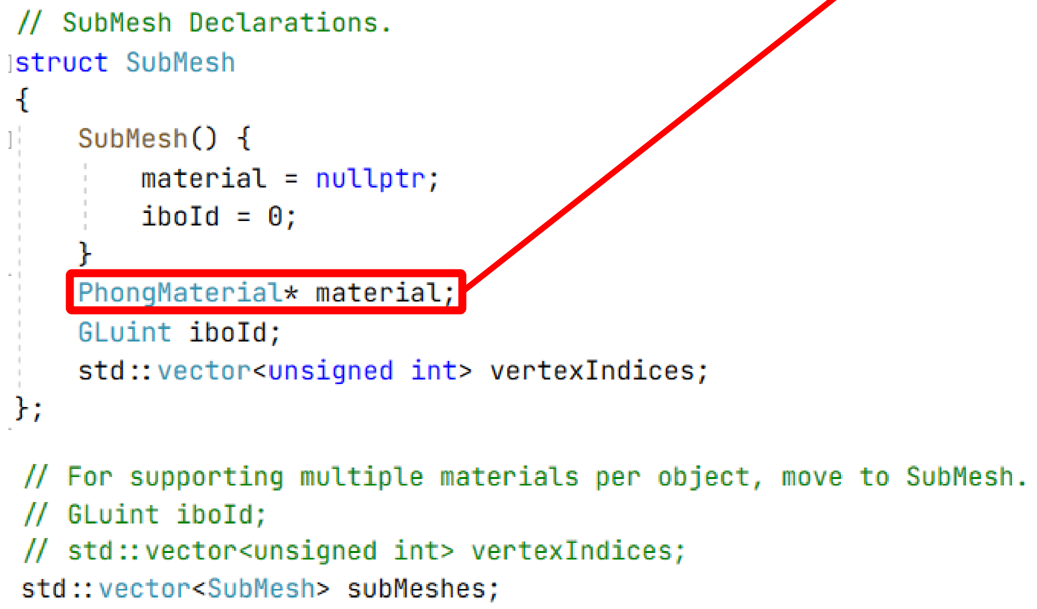
kumamon.jpg



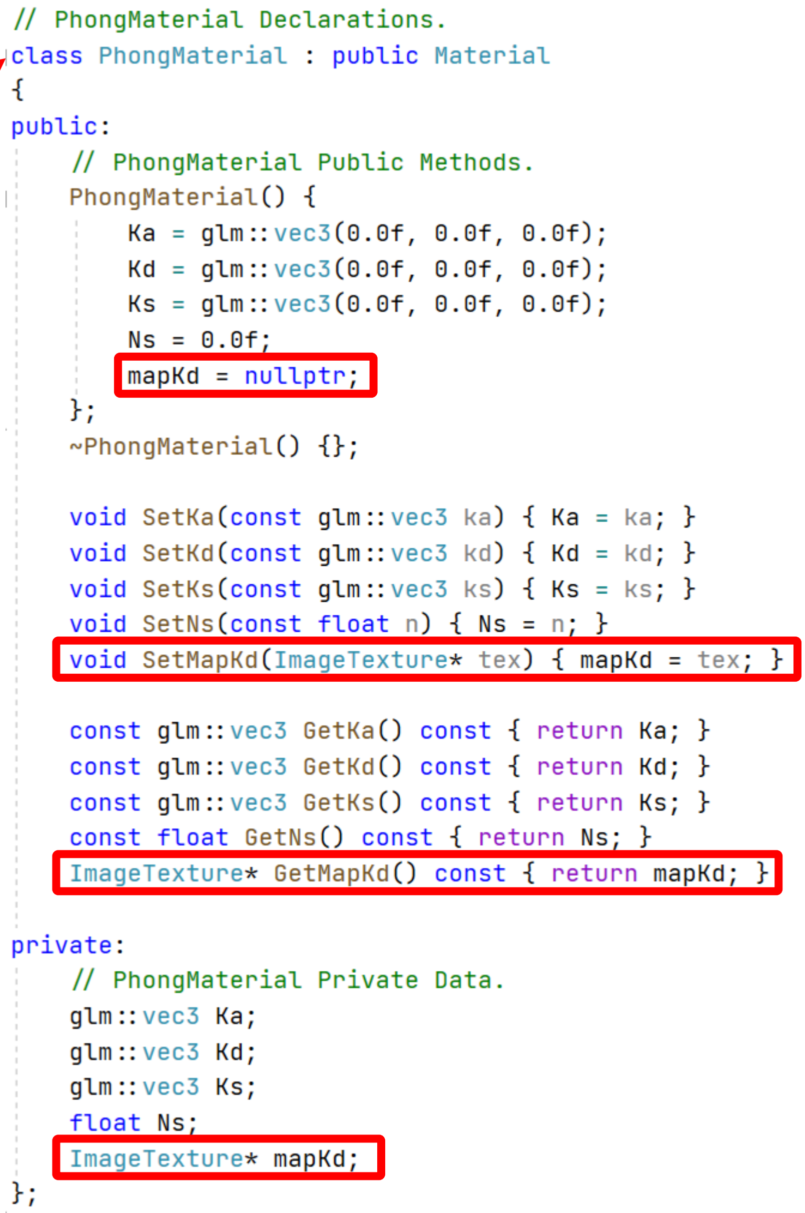
Material

in material.h

in trianglemesh.h



```
// SubMesh Declarations.
struct SubMesh
{
    SubMesh() {
        material = nullptr;
        iboId = 0;
    }
    PhongMaterial* material;
    GLuint iboId;
    std::vector<unsigned int> vertexIndices;
};

// For supporting multiple materials per object, move to SubMesh.
// GLuint iboId;
// std::vector<unsigned int> vertexIndices;
std::vector<SubMesh> subMeshes;
```



```
// PhongMaterial Declarations.
class PhongMaterial : public Material
{
public:
    // PhongMaterial Public Methods.
    PhongMaterial() {
        Ka = glm::vec3(0.0f, 0.0f, 0.0f);
        Kd = glm::vec3(0.0f, 0.0f, 0.0f);
        Ks = glm::vec3(0.0f, 0.0f, 0.0f);
        Ns = 0.0f;
        mapKd = nullptr;
    };
    ~PhongMaterial() {};

    void SetKa(const glm::vec3 ka) { Ka = ka; }
    void SetKd(const glm::vec3 kd) { Kd = kd; }
    void SetKs(const glm::vec3 ks) { Ks = ks; }
    void SetNs(const float n) { Ns = n; }
    void SetMapKd(ImageTexture* tex) { mapKd = tex; }

    const glm::vec3 GetKa() const { return Ka; }
    const glm::vec3 GetKd() const { return Kd; }
    const glm::vec3 GetKs() const { return Ks; }
    const float GetNs() const { return Ns; }
    ImageTexture* GetMapKd() const { return mapKd; }

private:
    // PhongMaterial Private Data.
    glm::vec3 Ka;
    glm::vec3 Kd;
    glm::vec3 Ks;
    float Ns;
    ImageTexture* mapKd;
};
```

ShaderProgram

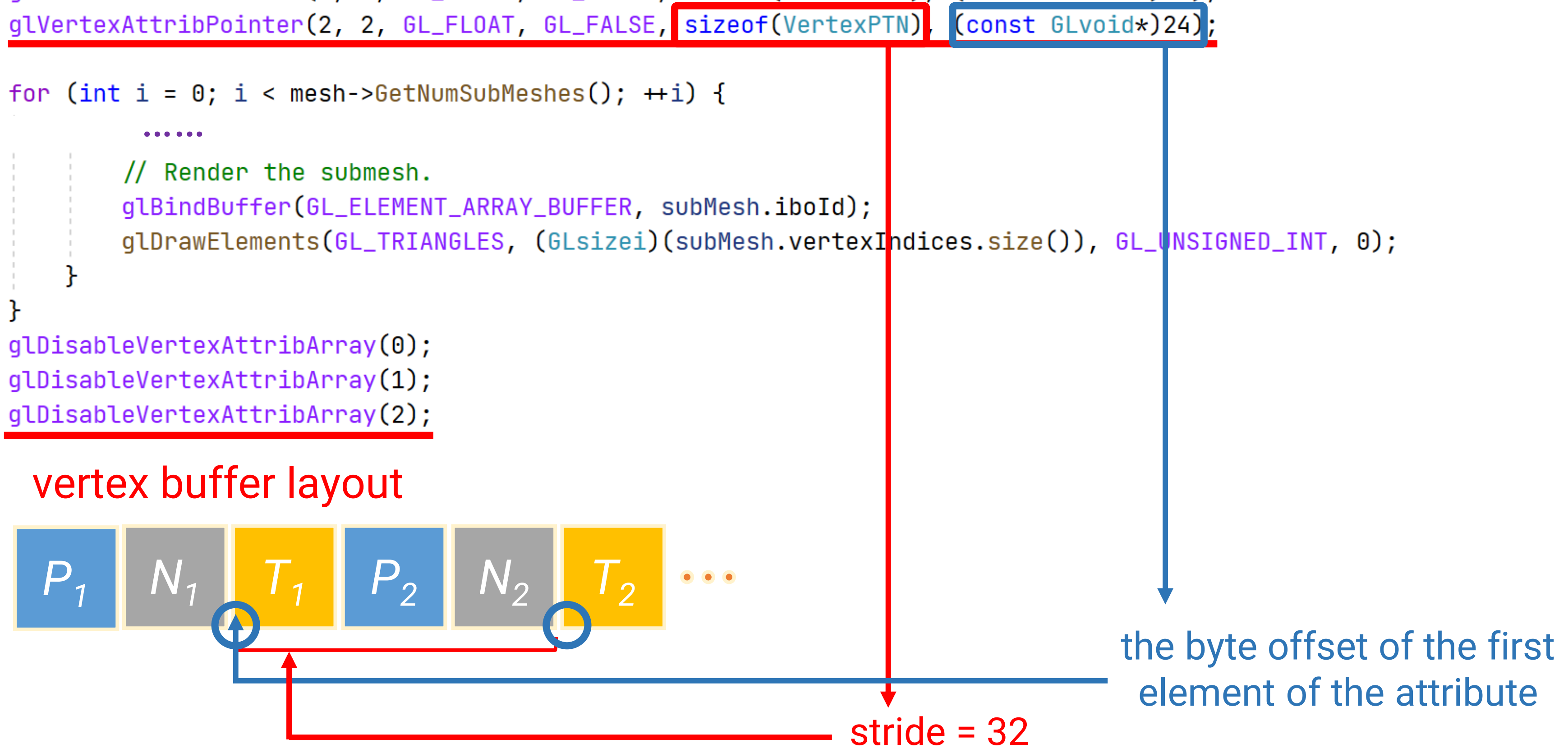
```
// PhongShadingDemoShaderProg Declarations.
class PhongShadingDemoShaderProg : public ShaderProg
{
public:
    // PhongShadingDemoShaderProg Public Methods.
    PhongShadingDemoShaderProg();
    ~PhongShadingDemoShaderProg();

    GLint GetLocM() const { return locM; }
    GLint GetLocNM() const { return locNM; }
    GLint GetLocCameraPos() const { return locCameraPos; }
    GLint GetLocKa() const { return locKa; }
    GLint GetLocKd() const { return locKd; }
    GLint GetLocKs() const { return locKs; }
    GLint GetLocNs() const { return locNs; }
    GLint GetLocAmbientLight() const { return locAmbientLight; }
    GLint GetLocDirLightDir() const { return locDirLightDir; }
    GLint GetLocDirLightRadiance() const { return locDirLightRad
    GLint GetLocPointLightPos() const { return locPointLightPos;
    GLint GetLocPointLightIntensity() const { return locPointLig
    GLint GetLocSpotLightPos() const { return locSpotLightPos; }
    GLint GetLocSpotLightIntensity() const { return locSpotLight
    GLint GetLocSpotLightDir() const { return locSpotLightDir; }
    GLint GetLocSpotLightCosineCutoff() const { return locSpotLi
    GLint GetLocSpotLightCosineTotalWidth() const { return locSp
    // ---------------------------------------------------------
    // Add your methods for supporting textures.
    // ---------------------------------------------------------
```

```
protected:
    // PhongShadingDemoShaderProg Protected Methods.
    void GetUniformVariableLocation();

private:
    // PhongShadingDemoShaderProg Public Data.
    // Transformation matrix.
    GLint locM;
    GLint locNM;
    GLint locCameraPos;
    // Material properties.
    GLint locKa;
    GLint locKd;
    GLint locKs;
    GLint locNs;
    // Light data.
    GLint locAmbientLight;
    GLint locDirLightDir;
    GLint locDirLightRadiance;
    GLint locPointLightPos;
    GLint locPointLightIntensity;
    GLint locSpotLightPos;
    GLint locSpotLightIntensity;
    GLint locSpotLightDir;
    GLint locSpotLightCosCutoff;
    GLint locSpotLightCosTotalWidth;
    // Texture data.
    // ----------------------------------------------------
    // Add your data for supporting textures.
    // ----------------------------------------------------
};
```

Recap: Multiple Vertex Attributes

```
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, mesh->GetVboId());
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPTN), 0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPTN), (const GLvoid*)12);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(VertexPTN), (const GLvoid*)24);

for (int i = 0; i < mesh->GetNumSubMeshes(); ++i) {
        ......
        // Render the submesh.
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, subMesh.iboId);
        glDrawElements(GL_TRIANGLES, (GLsizei)(subMesh.vertexIndices.size()), GL_UNSIGNED_INT, 0);
    }
}
glDisableVertexAttribArray(0);
glDisableVertexAttribArray(1);
glDisableVertexAttribArray(2);
```

vertex buffer layout



Loading and Setting Textures

- Please refer to the slides, “Implementation: Textures”

Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 Position;
layout (location = 1) in vec3 Normal;
layout (location = 2) in vec2 Texcoord;

// Transformation matrices.
uniform mat4 modelMatrix;
uniform mat4 normalMatrix;
uniform mat4 MVP;
…
```

Fragment Shader

```
#version 330 core
in vec3 iPosWorld;
in vec3 iNormalWorld;
```

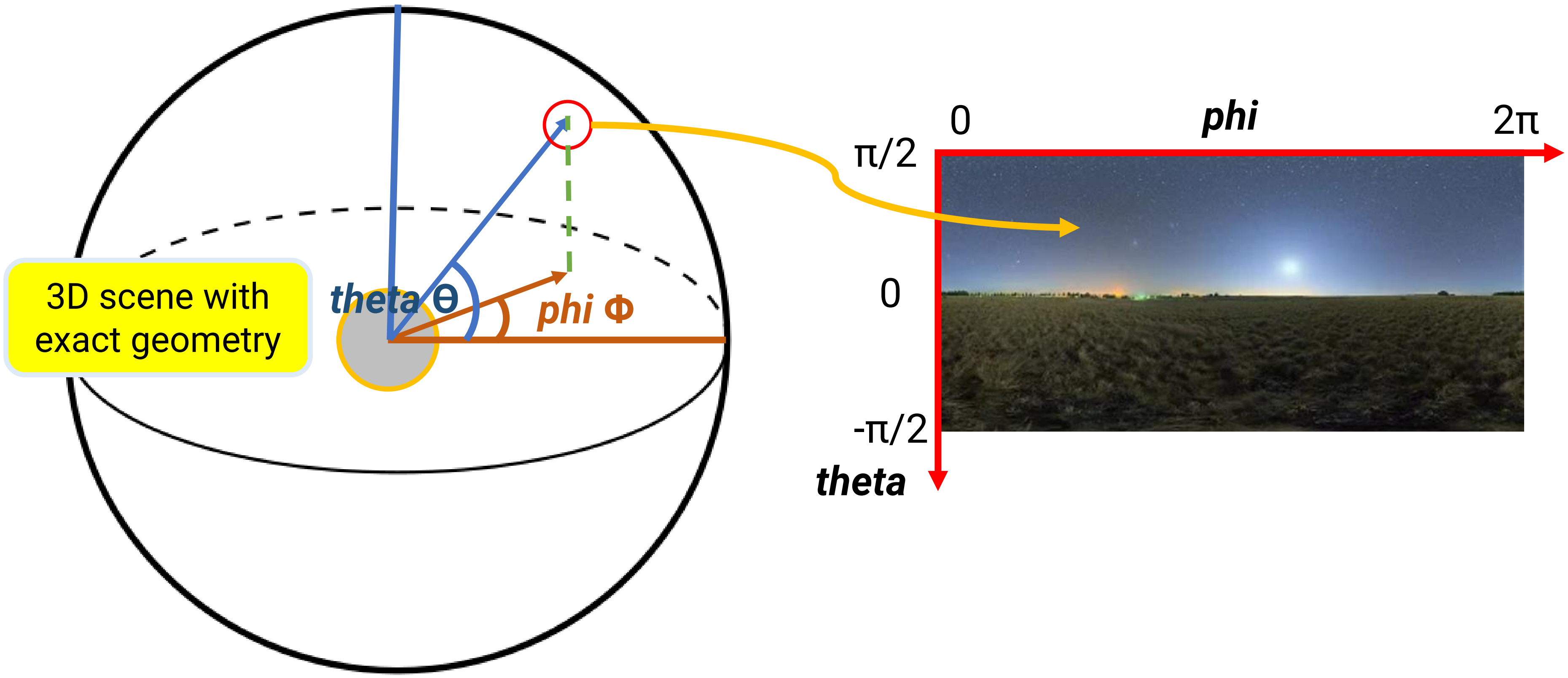
$\boxed{\texttt{in vec2 iTexCoord;}}$ the interpolated texture coordinate

$\boxed{\texttt{uniform sampler2D mapKd;}}$ the texture sampler object

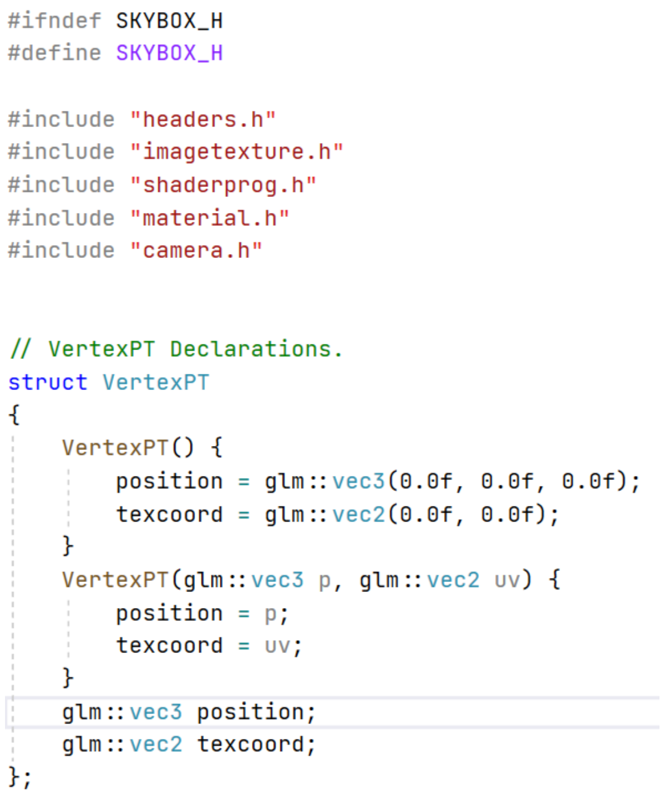
```
…
```

Skybox

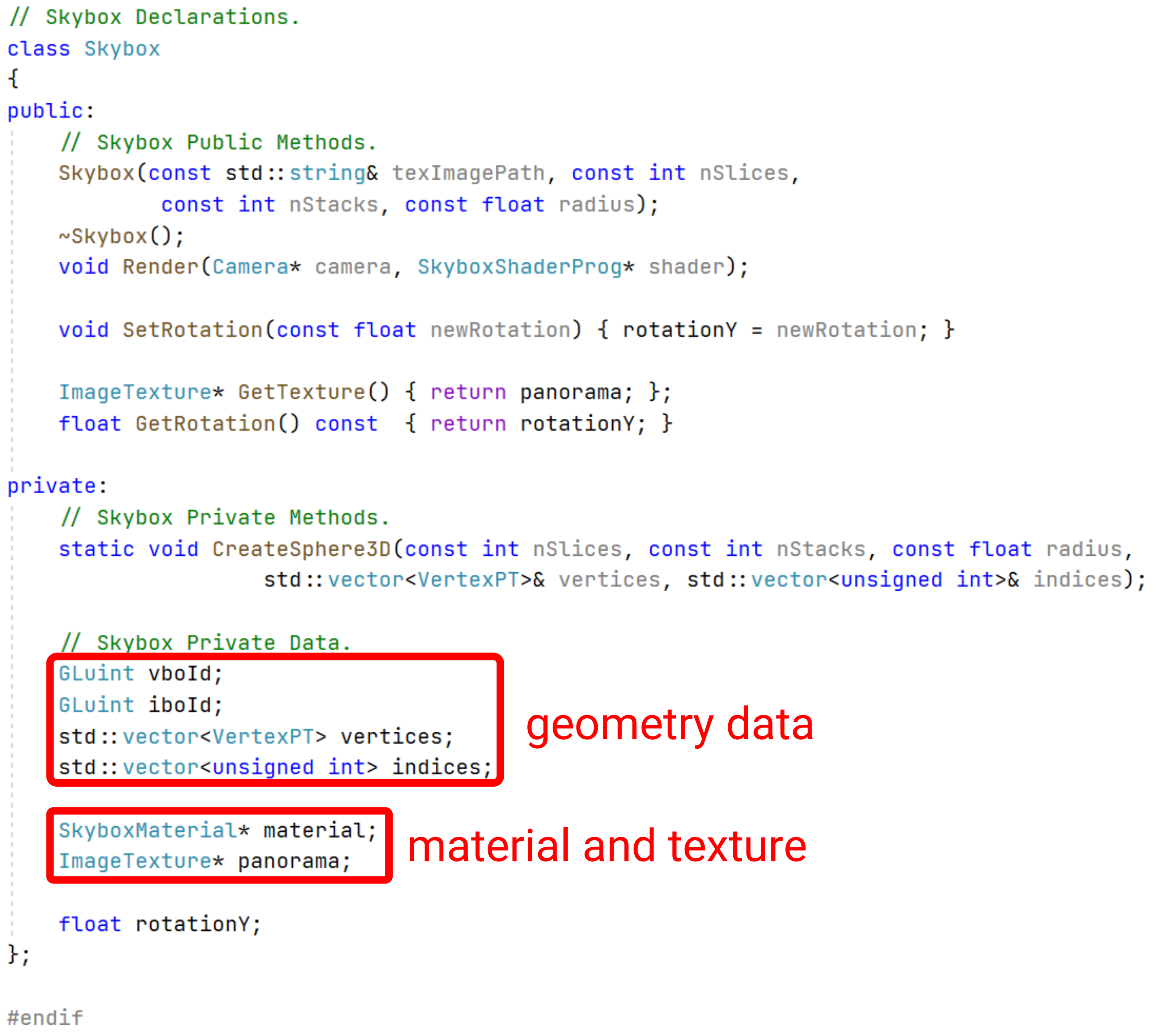
- Sphere geometry + **longitude-latitude** image
  - Centered at world-space origin, with a significant large radius



Skybox Implementation



```cpp
#ifndef SKYBOX_H
#define SKYBOX_H

#include "headers.h"
#include "imagetexture.h"
#include "shaderprog.h"
#include "material.h"
#include "camera.h"


// VertexPT Declarations.
struct VertexPT
{
    VertexPT() {
        position = glm::vec3(0.0f, 0.0f, 0.0f);
        texcoord = glm::vec2(0.0f, 0.0f);
    }
    VertexPT(glm::vec3 p, glm::vec2 uv) {
        position = p;
        texcoord = uv;
    }
    glm::vec3 position;
    glm::vec2 texcoord;
};
```



```cpp
// Skybox Declarations.
class Skybox
{
public:
    // Skybox Public Methods.
    Skybox(const std::string& texImagePath, const int nSlices,
           const int nStacks, const float radius);
    ~Skybox();
    void Render(Camera* camera, SkyboxShaderProg* shader);

    void SetRotation(const float newRotation) { rotationY = newRotation; }

    ImageTexture* GetTexture() { return panorama; };
    float GetRotation() const  { return rotationY; }

private:
    // Skybox Private Methods.
    static void CreateSphere3D(const int nSlices, const int nStacks, const float radius,
                    std::vector<VertexPT>& vertices, std::vector<unsigned int>& indices);

    // Skybox Private Data.
    GLuint vboId;
    GLuint iboId;
    std::vector<VertexPT> vertices;
    std::vector<unsigned int> indices;

    SkyboxMaterial* material;
    ImageTexture* panorama;

    float rotationY;
};

#endif
```

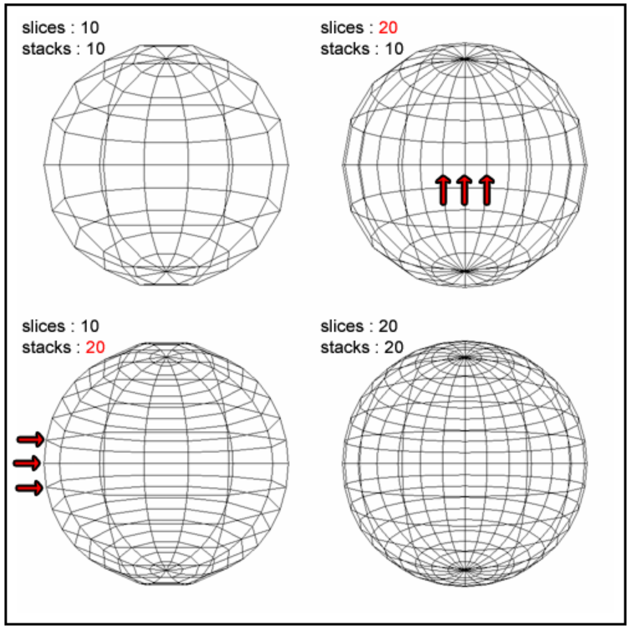
geometry data

material and texture

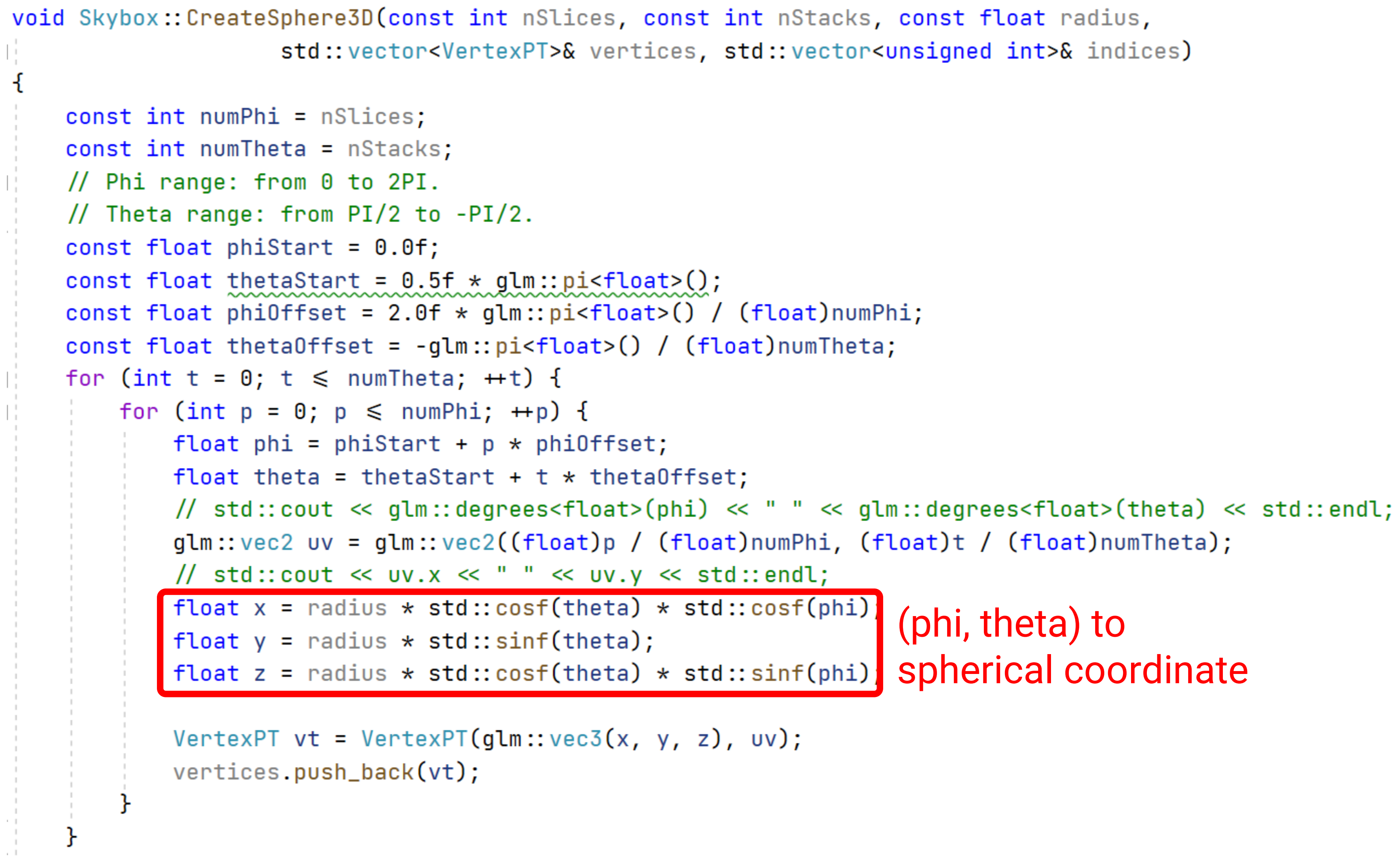
Skybox.h / Skybox.cpp

Skybox Implementation (cont.)

```
Skybox::Skybox(const std::string& texImagePath, const int nSlices, const int nStacks, const float radius)
{
    rotationY = 0.0f;

    // Load panorama.
    panorama = new ImageTexture(texImagePath);
    // panorama->Preview();

    // Create material.
    material = new SkyboxMaterial();
    material->SetMapKd(panorama);

    // Create sphere geometry.
    CreateSphere3D(nSlices, nStacks, radius, vertices, indices);

    // Create vertex buffer.
    glGenBuffers(1, &vboId);
    glBindBuffer(GL_ARRAY_BUFFER, vboId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(VertexPT) * vertices.size(), &vertices[0], GL_STATIC_DRAW);
    // Create index buffer.
    glGenBuffers(1, &iboId);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iboId);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned int) * indices.size(), &(indices[0]), GL_STATIC_DRAW);
}
```



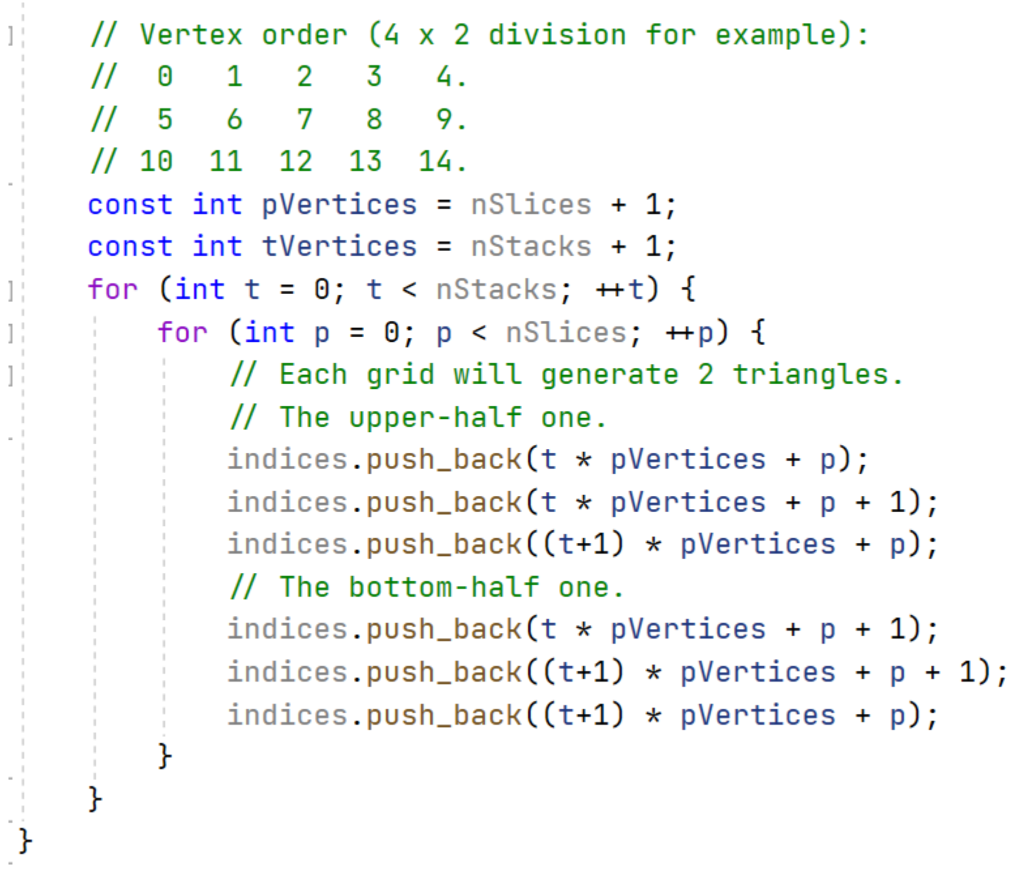
Skybox Implementation (cont.)



```cpp
void Skybox::CreateSphere3D(const int nSlices, const int nStacks, const float radius,
                            std::vector<VertexPT>& vertices, std::vector<unsigned int>& indices)
{
    const int numPhi = nSlices;
    const int numTheta = nStacks;
    // Phi range: from 0 to 2PI.
    // Theta range: from PI/2 to -PI/2.
    const float phiStart = 0.0f;
    const float thetaStart = 0.5f * glm::pi<float>();
    const float phiOffset = 2.0f * glm::pi<float>() / (float)numPhi;
    const float thetaOffset = -glm::pi<float>() / (float)numTheta;
    for (int t = 0; t <= numTheta; ++t) {
        for (int p = 0; p <= numPhi; ++p) {
            float phi = phiStart + p * phiOffset;
            float theta = thetaStart + t * thetaOffset;
            // std::cout << glm::degrees<float>(phi) << " " << glm::degrees<float>(theta) << std::endl;
            glm::vec2 uv = glm::vec2((float)p / (float)numPhi, (float)t / (float)numTheta);
            // std::cout << uv.x << " " << uv.y << std::endl;
            float x = radius * std::cosf(theta) * std::cosf(phi);
            float y = radius * std::sinf(theta);
            float z = radius * std::cosf(theta) * std::sinf(phi);

            VertexPT vt = VertexPT(glm::vec3(x, y, z), uv);
            vertices.push_back(vt);
        }
    }
```

(phi, theta) to spherical coordinate

Skybox Implementation (cont.)



```
    // Vertex order (4 x 2 division for example):
    //  0   1   2   3   4.
    //  5   6   7   8   9.
    // 10  11  12  13  14.
    const int pVertices = nSlices + 1;
    const int tVertices = nStacks + 1;
    for (int t = 0; t < nStacks; ++t) {
        for (int p = 0; p < nSlices; ++p) {
            // Each grid will generate 2 triangles.
            // The upper-half one.
            indices.push_back(t * pVertices + p);
            indices.push_back(t * pVertices + p + 1);
            indices.push_back((t+1) * pVertices + p);
            // The bottom-half one.
            indices.push_back(t * pVertices + p + 1);
            indices.push_back((t+1) * pVertices + p + 1);
            indices.push_back((t+1) * pVertices + p);
        }
    }
}
```

Skybox Implementation (cont.)

```
void Skybox::Render(Camera* camera, SkyboxShaderProg* shader)
{
    glEnableVertexAttribArray(0);
    glEnableVertexAttribArray(1);

    glBindBuffer(GL_ARRAY_BUFFER, vboId);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPT), 0);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(VertexPT), (const GLvoid*)12);

    shader->Bind();

    // Set transform.
    // -------------------------------------------------------
    // Modify code here to rotate the skybox.
    glm::mat4x4 MVP = camera->GetProjMatrix() * camera->GetViewMatrix();
    // -------------------------------------------------------
    glUniformMatrix4fv(shader->GetLocMVP(), 1, GL_FALSE, glm::value_ptr(MVP));
    // Set material properties.
    if (material->GetMapKd() ≠ nullptr) {
        material->GetMapKd()->Bind(GL_TEXTURE0);
        glUniform1i(shader->GetLocMapKd(), 0);
    }
```

Skybox Implementation (cont.)

```
    // Draw.
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iboId);
    glDrawElements(GL_TRIANGLES, (GLsizei)(indices.size()), GL_UNSIGNED_INT, 0);

    shader->UnBind();

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```

In ICG2022_HW3.cpp, RenderSceneCB()

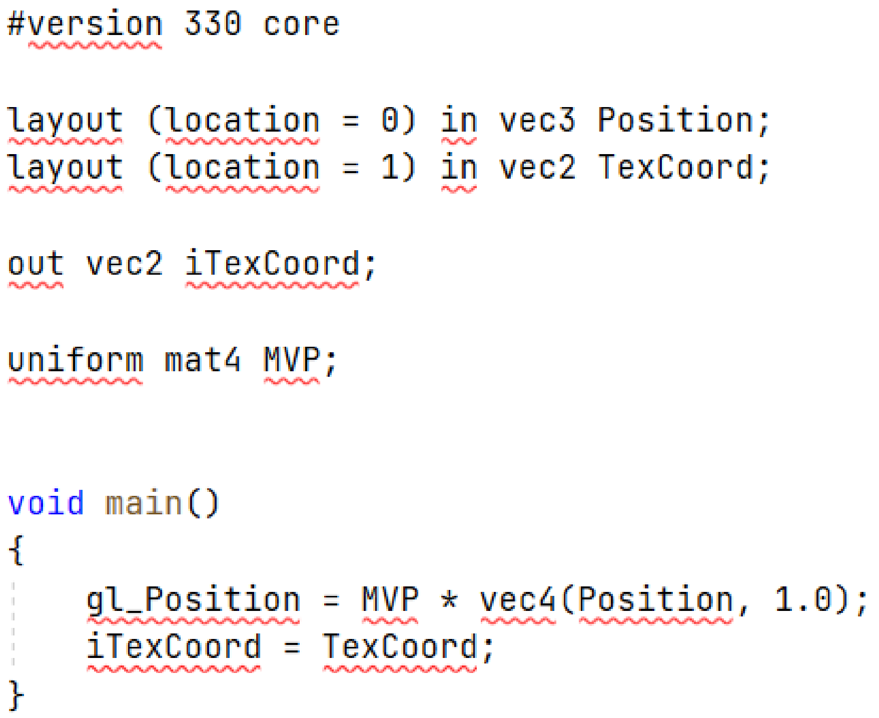
```
// Render skybox. ------------------------------------------------------------
if (skybox ≠ nullptr) {
    // ---------------------------------------------------------
    // TODO: do something to rotate the skybox.
    skybox->Render(camera, skyboxShader);
}
// ------------------------------------------------------------------------------
```

Skybox Implementation (cont.)

```cpp
Skybox::~Skybox()
{
    vertices.clear();
    glDeleteBuffers(1, &vboId);
    indices.clear();
    glDeleteBuffers(1, &iboId);

    if (panorama) {
        delete panorama;
        panorama = nullptr;
    }
    if (material) {
        delete material;
        material = nullptr;
    }
}
```

Skybox Implementation (cont.)

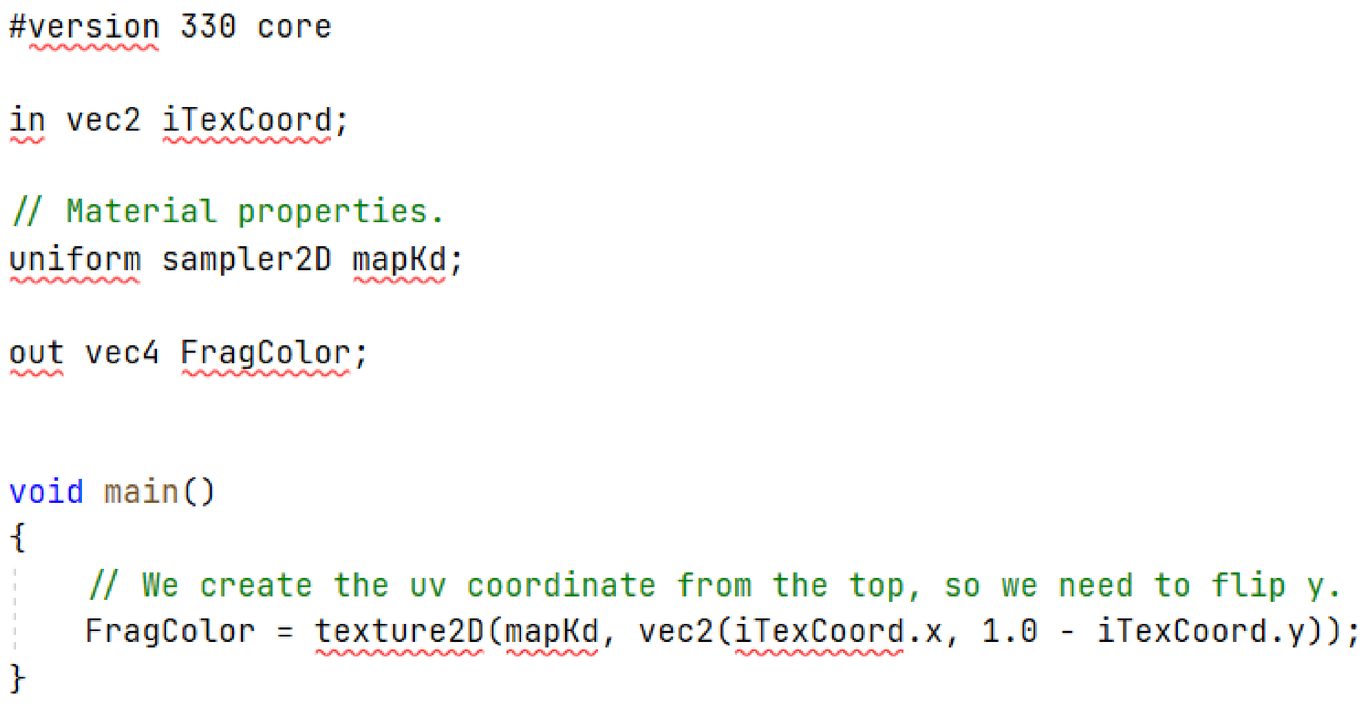
- Vertex shader



```
#version 330 core

layout (location = 0) in vec3 Position;
layout (location = 1) in vec2 TexCoord;

out vec2 iTexCoord;

uniform mat4 MVP;


void main()
{
    gl_Position = MVP * vec4(Position, 1.0);
    iTexCoord = TexCoord;
}
```

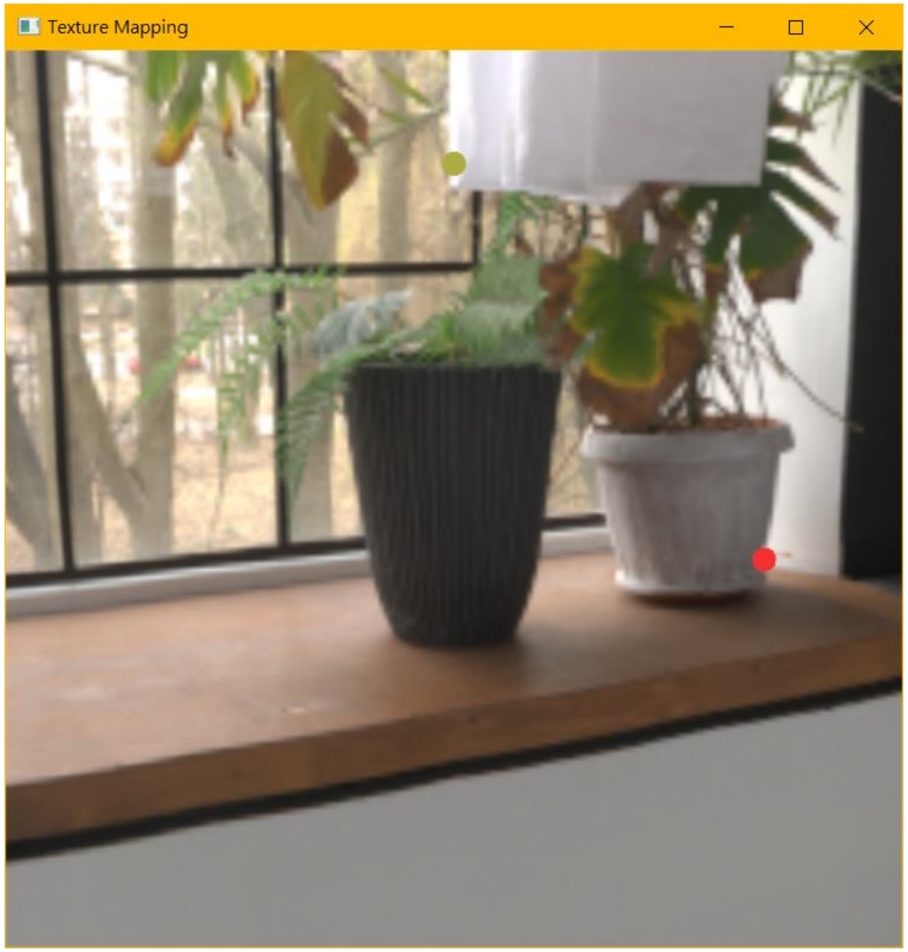
Skybox Implementation (cont.)

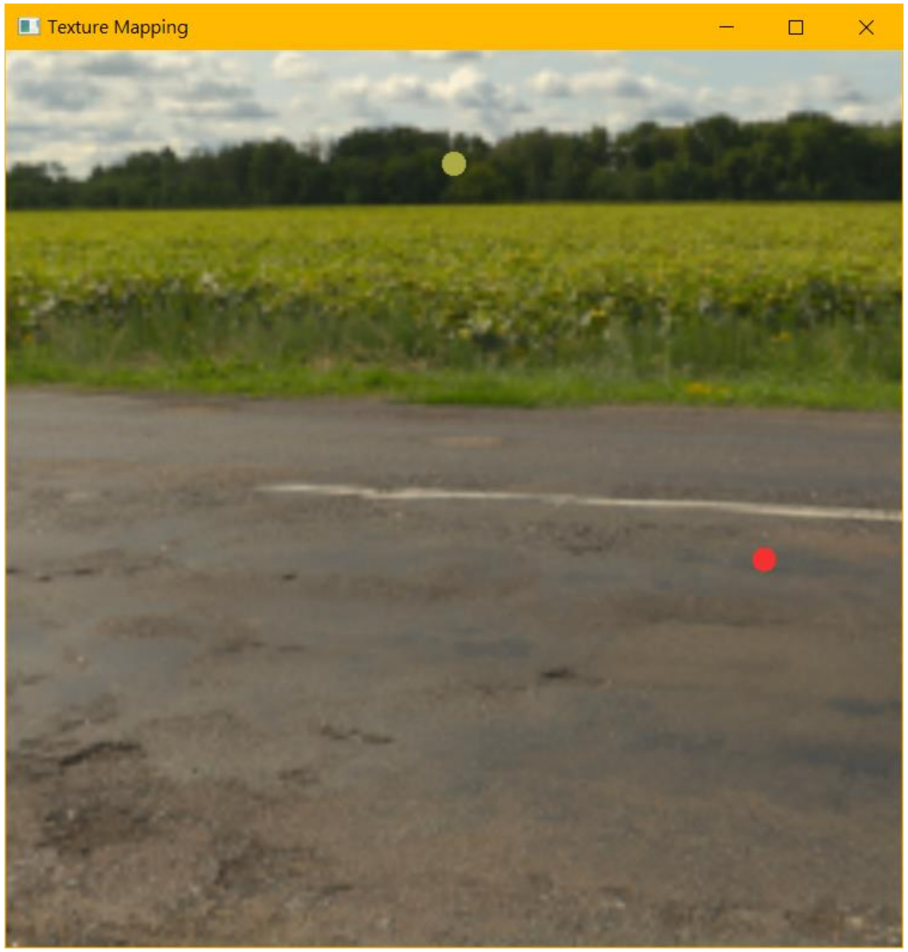
- Fragment shader



```glsl
#version 330 core

in vec2 iTexCoord;

// Material properties.
uniform sampler2D mapKd;

out vec4 FragColor;


void main()
{
    // We create the uv coordinate from the top, so we need to flip y.
    FragColor = texture2D(mapKd, vec2(iTexCoord.x, 1.0 - iTexCoord.y));
}
```

Skybox Implementation (cont.)





Task List

- Revise your OBJ loader if you ignore the texture coordinates in previous homeworks
- Load the texture specified in the **material file (using my *ImageTexture* class)** and **store it into the material data**
- Modify ***PhongMaterial*** and ***PhongShadingDemoShaderProg*** classes for supporting diffuse textures
- Revise the rendering function for **supporting texture coordinate vertex attribute** and **setting textures to shader**
- Revise your **vertex shader** to support **texture coordinate attribute** (and pass to fragment shader)
- Revise your **fragment shader** to lookup the diffuse texture
- Implement the bonus if you want

Task List (cont.)

- Please download the skeleton code from 數位學苑 3.0
- You might need to retouch (but not limited to):
  - ICG2022_HW3.cpp

    ➔ Set textures to shader in **RenderSceneCB()**

    ➔ Add functions for changing/rotating skybox (bonus)
  - trianglemesh.h / trianglemesh.cpp

    ➔ Add your HW2 code & **textures loading**
  - shaderprog.h / shaderprog.cpp

    ➔ Add texture data and shader variable
  - phong_shading_demo.vs / phong_shading_demo.fs

    ➔ Look up the texture for ***Kd***

    (if the submesh has no texture, use its original *Kd !!!*)

Any Questions?