Textures (Part II)

Computer Graphics

Yu-Ting Wu

Outline

- Overview
- Texture data (Part I)
- Texture filtering
- Applications

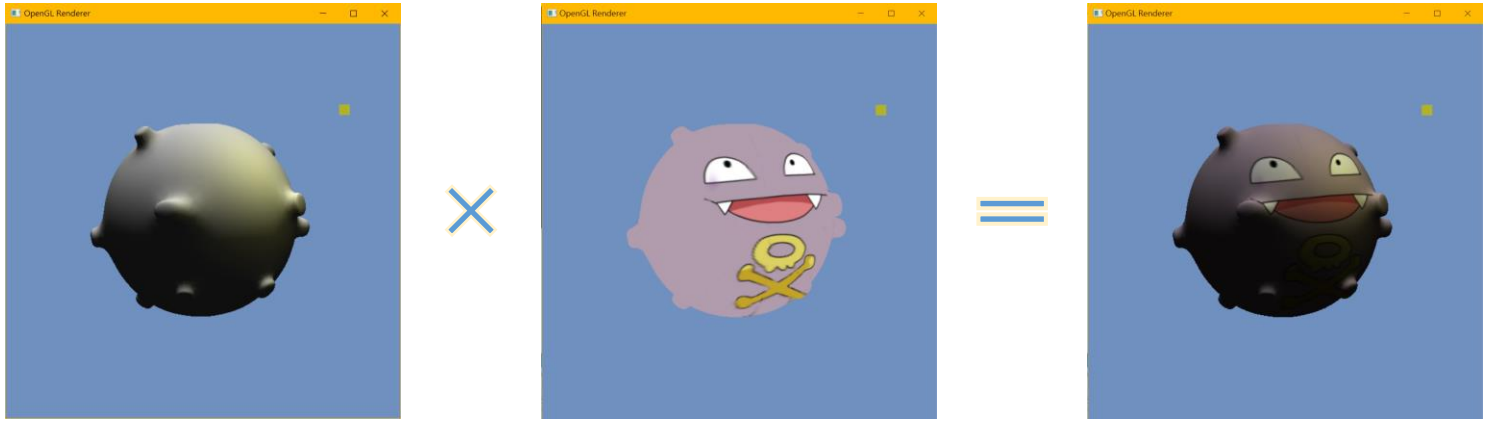
- OpenGL implementation (Part II)

Outline

- Overview
- Texture data
- Texture filtering
- Applications
- **OpenGL implementation**

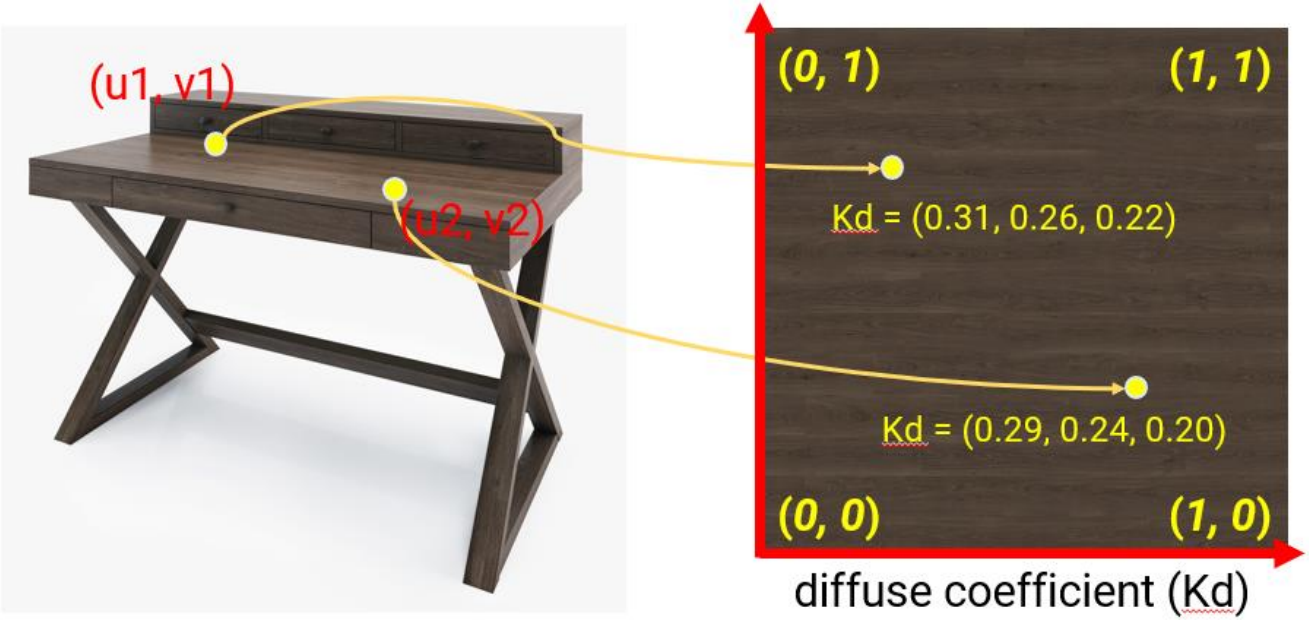
Overview

- The sample program ***Texture*** demonstrates how to create an OpenGL texture and bind it to shader
- The program, ***Texture***, is very similar to the previous sample program, ***Shading***
- In the shader, the output color is determined by **per-vertex lighting multiplied by per-fragment texture color**
  - The way OpenGL 1.1 combines textures and lighting



Overview (cont.)

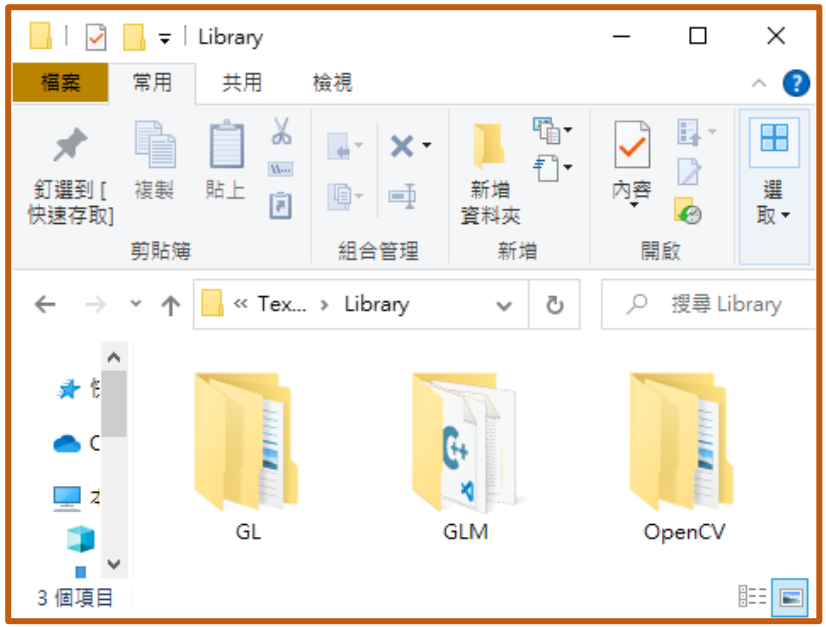
- In OpenGL 2.0 and after, the correct way to handle texture is to use the texture color as diffuse coefficients (Kd)

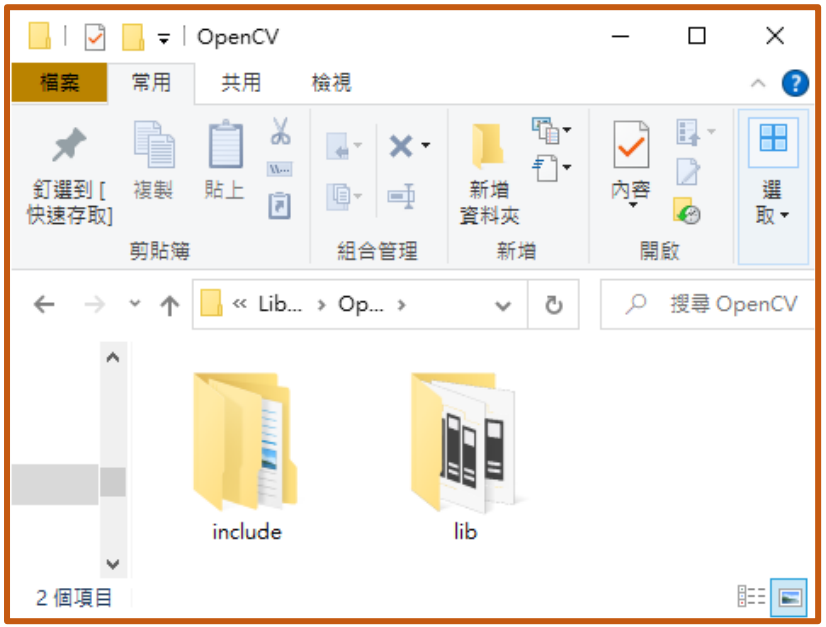


- This needs **per-fragment lighting**, which is part of your HW2/HW3

Additional Library for Loading Images

- **OpenCV: Open Source Computer Vision Library (link)**
  - A cross-platform open-source C/C++ library for computer vision and image processing applications
  - We use it for loading image textures





Data Structure: ImageTexture

- Defined in imagetexture.h / imagetexture.cpp

```
#ifndef IMAGE_TEXTURE_H
#define IMAGE_TEXTURE_H

#include "headers.h"

// Texture Declarations.
class ImageTexture
{
public:
    // Texture Public Methods.
    ImageTexture(const std::string filePath);
    ~ImageTexture();

    void Bind(GLenum textureUnit);
    void Preview();
```

OpenGL texture object (ID)

```
private:
    // Texture Private Data.
    std::string texFileName;
    GLuint textureObj;
    int imageWidth;
    int imageHeight;
    int numChannels;
    cv::Mat texImage;
};

#endif
```

pixel data (2D array)

Data Structure: ImageTexture (cont.)

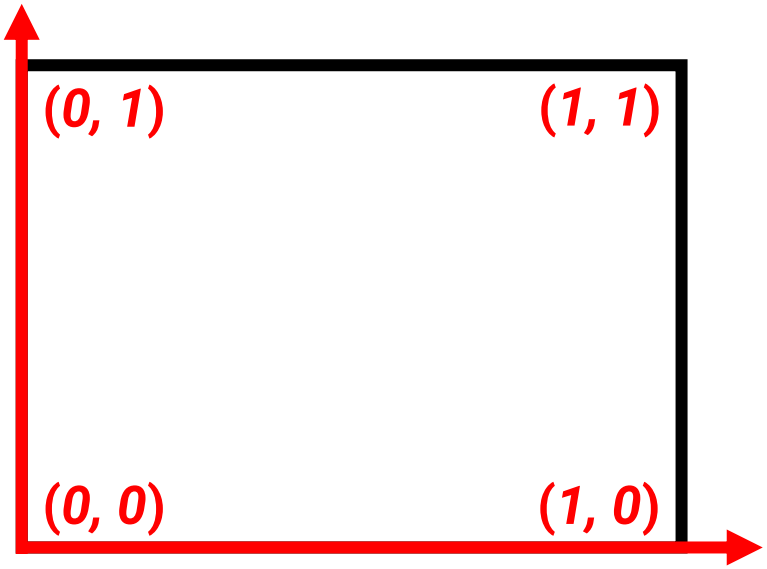
```
ImageTexture::ImageTexture(const std::string filePath)
    : texFileName(filePath)
{
    imageWidth = 0;
    imageHeight = 0;
    numChannels = 0;
    textureObj = 0;

    // Try to load texture image.
    texImage = cv::imread(texFileName);
    if (texImage.rows == 0 || texImage.cols == 0) {
        std::cerr << "[ERROR] Failed to load image texture: " << filePath << std::endl;
        return;
    }
    imageWidth = texImage.cols;
    imageHeight = texImage.rows;
    numChannels = texImage.channels();

    // Flip texture in vertical direction.
    // OpenCV has smaller y coordinate on top; while OpenGL has larger.
    cv::flip(texImage, texImage, 0);
```

load an image and store data in a cv::Mat (OpenCV's API)

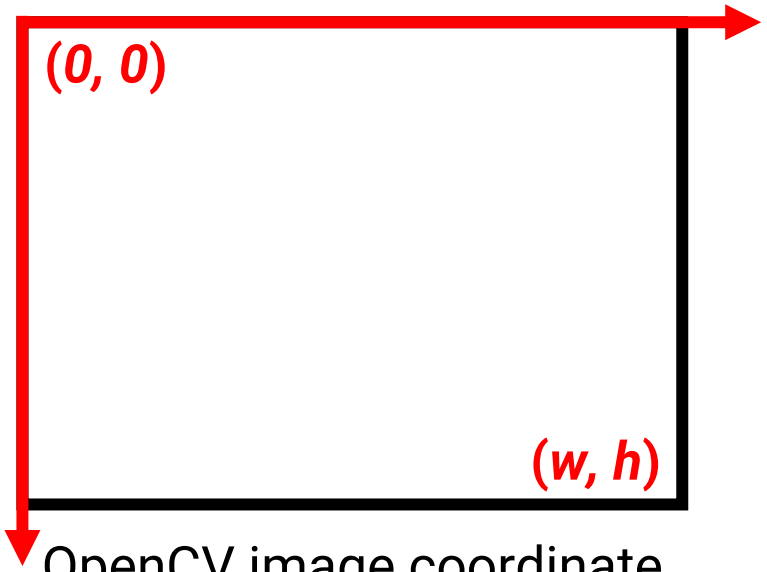
3 for RGB images
4 for RGBA images

flip image vertically (OpenCV's API)

OpenCV Image Format



OpenGL texture coordinate



OpenCV image coordinate

Data Structure: ImageTexture (cont.)

```
glGenTextures(1, &textureObj);
glBindTexture(GL_TEXTURE_2D, textureObj);
switch (numChannels) {
case 1:
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RED, imageWidth, imageHeight,
                 0, GL_RED, GL_UNSIGNED_BYTE, texImage.ptr());
    break;
case 3:
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight,
                 0, GL_BGR, GL_UNSIGNED_BYTE, texImage.ptr());
    break;
case 4:
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imageWidth, imageHeight,
                 0, GL_BGRA, GL_UNSIGNED_BYTE, texImage.ptr());
    break;
default:
    std::cerr << "[ERROR] Unsupport texture format" << std::endl;
    break;
}
```

generate an OpenGL texture object (ID)

bind the texture object for follow-up operations

set image data to texture

OpenCV stores images in BGR/BGRA format

Data Structure: ImageTexture (cont.)

setup texture sampling and filtering mode

```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    glGenerateMipmap(GL_TEXTURE_2D);

    glBindTexture(GL_TEXTURE_2D, 0);
}
```

generate mipmap

unbind texture

Texture Related APIs

- Set image data to texture (ref: https://reurl.cc/NGG805)

```
void glTexImage2D (
    GLenum target,
    GLint level,
    GLint internalformat,
    GLsizei width,
    GLsizei height,
    GLint border,
    GLenum format,
    GLenum type,
    const void * data
);
```

GL_TEXTURE_2D, GL_TEXTURE_CUBE_MAP_POSITIVE_X, … etc.

level of details, usually set to 0

the internal format of the **texture**
GL_RED, GL_RG, GL_RGB, GL_RGBA, GL_DEPTH_COMPONENT … etc.

must be 0

the format of the **image data**
GL_RED, GL_RG, GL_RGB, GL_RGBA … etc.

the data type of the **pixel data**
GL_UNSIGNED_BYTE, GL_FLOAT … etc.

a pointer to the image data in memory

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imageWidth, imageHeight,
                0, GL_BGRA, GL_UNSIGNED_BYTE, texImage.ptr());
```

Texture Related APIs (cont.)

- Set the sampling and filtering mode of the bound texture (ref: https://reurl.cc/911AMv)

```
void glTexParameteri(f) (
    GLenum target,
    GLenum pname,
    GLint (GLfloat) param
);
```

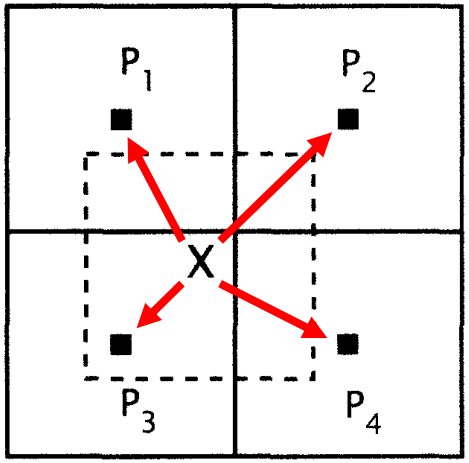
Specifies the symbolic name of a single-valued texture parameter, such as
GL_TEXTURE_MIN_FILTER
GL_TEXTURE_MAG_FILTER
GL_TEXTURE_WRAP_S (T) … etc.

parameter value
GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR
GL_CLAMP_TO_EDGE, GL_REPEAT … etc.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

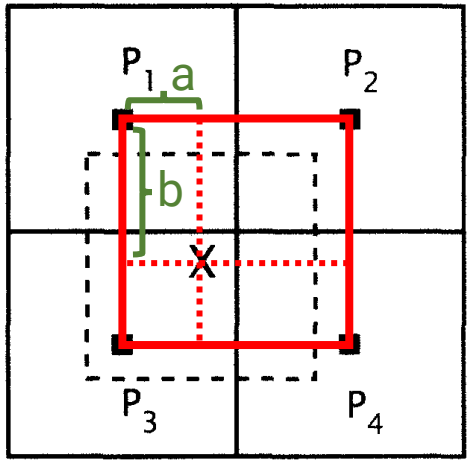
Recap: Texture Filtering

- Strategies
  - **Nearest neighbor**
  - **Bilinear interpolation**



nearest neighbor

P_3 is closest
Use P_3's pixel value

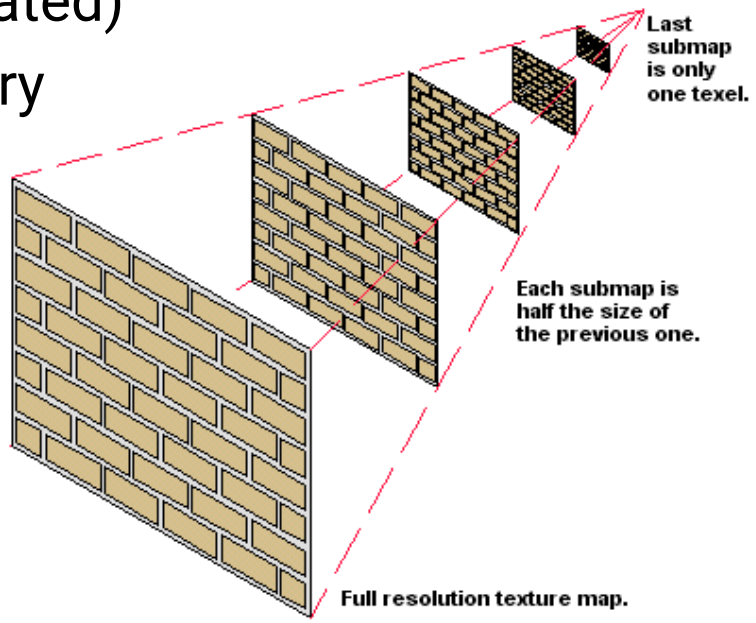


bilinear interpolation

$(1-a)(1-b)P_1 + (a)(1-b)P_2 + (1-a)(b)P_3 + (a)(b)P_4$

Recap: Mipmap

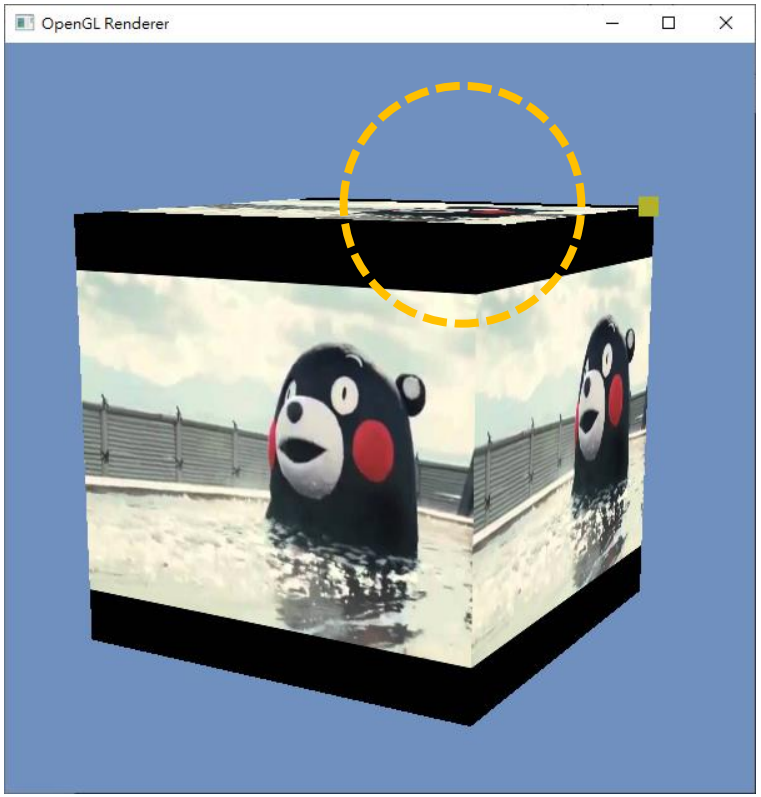
- Mipmap provides a clever way to solve this problem
- **Pre-process**
  - Build a **hierarchical representation** of the texture image
  - Each level has a half resolution of its previous level (generated by linearly interpolated)
  - Take at most **1/3** more memory



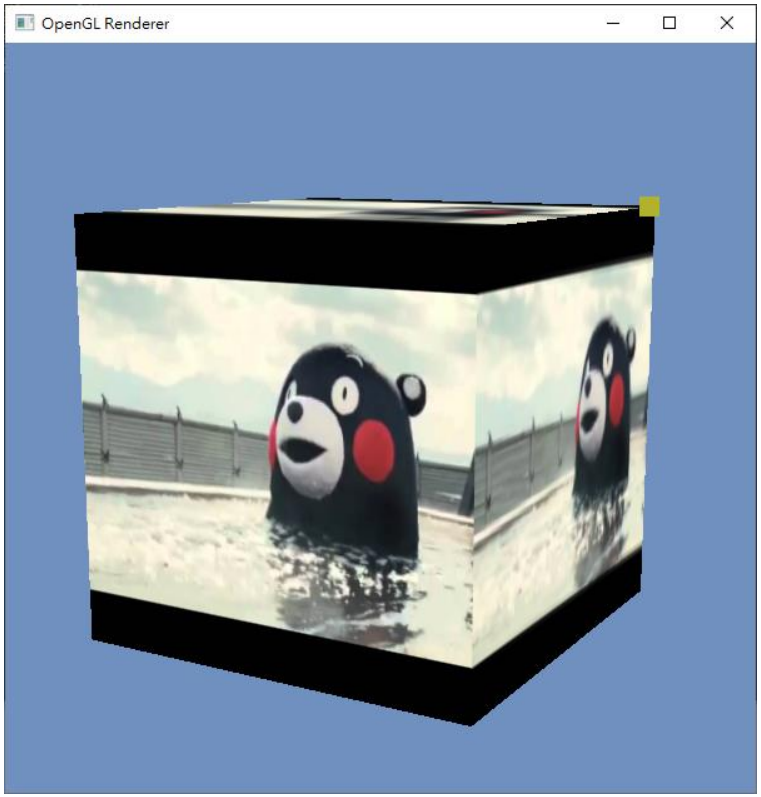

```
glGenerateMipmap(GL_TEXTURE_2D);
```

Texture Related APIs (cont.)

- Mipmap off v.s. on



off



on

Texture Related APIs (cont.)

- **Texture clamping mode**
  - Determine what will happen when the texture coordinates do not locate within [0, 1]

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

Data Structure: ImageTexture (cont.)

```
void ImageTexture::Bind(GLenum textureUnit)
{
    glActiveTexture(textureUnit);
    glBindTexture(GL_TEXTURE_2D, textureObj);
}

void ImageTexture::Preview()
{
    std::string windowText = "[DEBUG] TexturePreview: " + texFileName;
    cv::Mat previewImg = cv::Mat(texImage.rows, texImage.cols, texImage.type());
    cv::cvtColor(texImage, previewImg, cv::COLOR_BGR2RGB);
    cv::imshow(windowText, previewImg);
    cv::waitKey(0);
}
```

the n^{th} texture in the shader

Shader

gouraud_shading_demo.vs - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
#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 worldMatrix;
uniform mat4 viewMatrix;
uniform mat4 normalMatrix;
uniform mat4 MVP;
// Material properties.
uniform vec3 Ka;
uniform vec3 Kd;
uniform vec3 Ks;
uniform float Ns;
// Light data.
uniform vec3 ambientLight;
uniform vec3 dirLightDir;
uniform vec3 dirLightRadiance;
uniform vec3 pointLightPos;
uniform vec3 pointLightIntensity;

// Data pass to fragment shader.
out vec3 iLightingColor;
out vec2 iTexCoord;

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

vertex shader

gouraud_shading_demo.fs - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
#version 330 core

in vec3 iLightingColor;
in vec2 iTexCoord;

uniform sampler2D mapKd;

out vec4 FragColor;


void main()
{
    vec3 texColor = texture2D(mapKd, iTexCoord).rgb;
    // FragColor = vec4(iLightingColor, 1.0);
    // FragColor = vec4(texColor, 1.0);
    FragColor = vec4(iLightingColor * texColor, 1.0);
}
```

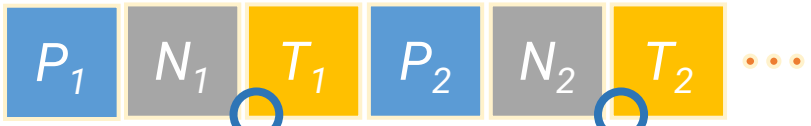
interpolated texture coordinate

sample the texture
using texture coordinate

fragment shader

Adding TexCoord in Vertex Buffer

```
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, vboId);
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);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iboId);
glDrawElements(GL_TRIANGLES, GetNumIndices(), GL_UNSIGNED_INT, 0);
glDisableVertexAttribArray(0);
glDisableVertexAttribArray(1);
glDisableVertexAttribArray(2);
```



the byte offset of the first element of the attribute

stride = 32

Data Structure: ShaderProgram

- Modify the ***GouraudShadingDemoShaderProg*** class in ShaderProg.h / ShaderProgram.cpp

new private data

```
// Texture data.
GLint locMapKd;
```

new public method

```
GLint GetLocMapKd() const { return locMapKd; }
```

get variable location

```
void GouraudShadingDemoShaderProg::GetUniformVariableLocation()
{

    ⋮

    locMapKd = glGetUniformLocation(shaderProgId, "mapKd");
}
```

Main Program

global variable

```
// Texture.
ImageTexture* imageTex = nullptr;
```

modified SceneObject

```
// SceneObject.
struct SceneObject
{
    SceneObject() {
        mesh = nullptr;
        worldMatrix = glm::mat4x4(1.0f);
        Ka = glm::vec3(0.3f, 0.3f, 0.3f);
        Kd = glm::vec3(0.8f, 0.8f, 0.8f);
        Ks = glm::vec3(0.6f, 0.6f, 0.6f);
        Ns = 50.0f;
    }
    TriangleMesh* mesh;
    glm::mat4x4 worldMatrix;
    // Material properties.
    glm::vec3 Ka;
    glm::vec3 Kd;
    glm::vec3 Ks;
    float Ns;
    // Texture.
    ImageTexture* tex = nullptr;
};
```

SetupScene

```
void SetupScene()
{
    // Scene object ------------------------------------------
    mesh = new TriangleMesh();
    // mesh->LoadFromFile("models/Koffing/Koffing.obj", true);
    mesh->LoadFromFile("models/TexCube/TexCube.obj", true);
    mesh->CreateBuffers();
    mesh->ShowInfo();
    sceneObj.mesh = mesh;
    // Load texture.
    // imageTex = new ImageTexture("models/Koffing/tex.png");
    imageTex = new ImageTexture("models/TexCube/kumamon.jpg");
    sceneObj.tex = imageTex;
```

ReleaseResource

```
void ReleaseResources()
{
    // Delete scene objects and lights.
    if (mesh != nullptr) {
        delete mesh;
        mesh = nullptr;
    }
    if (imageTex != nullptr) {
        delete imageTex;
        imageTex = nullptr;
    }
```

Main Program (cont.)

- RenderSceneCB

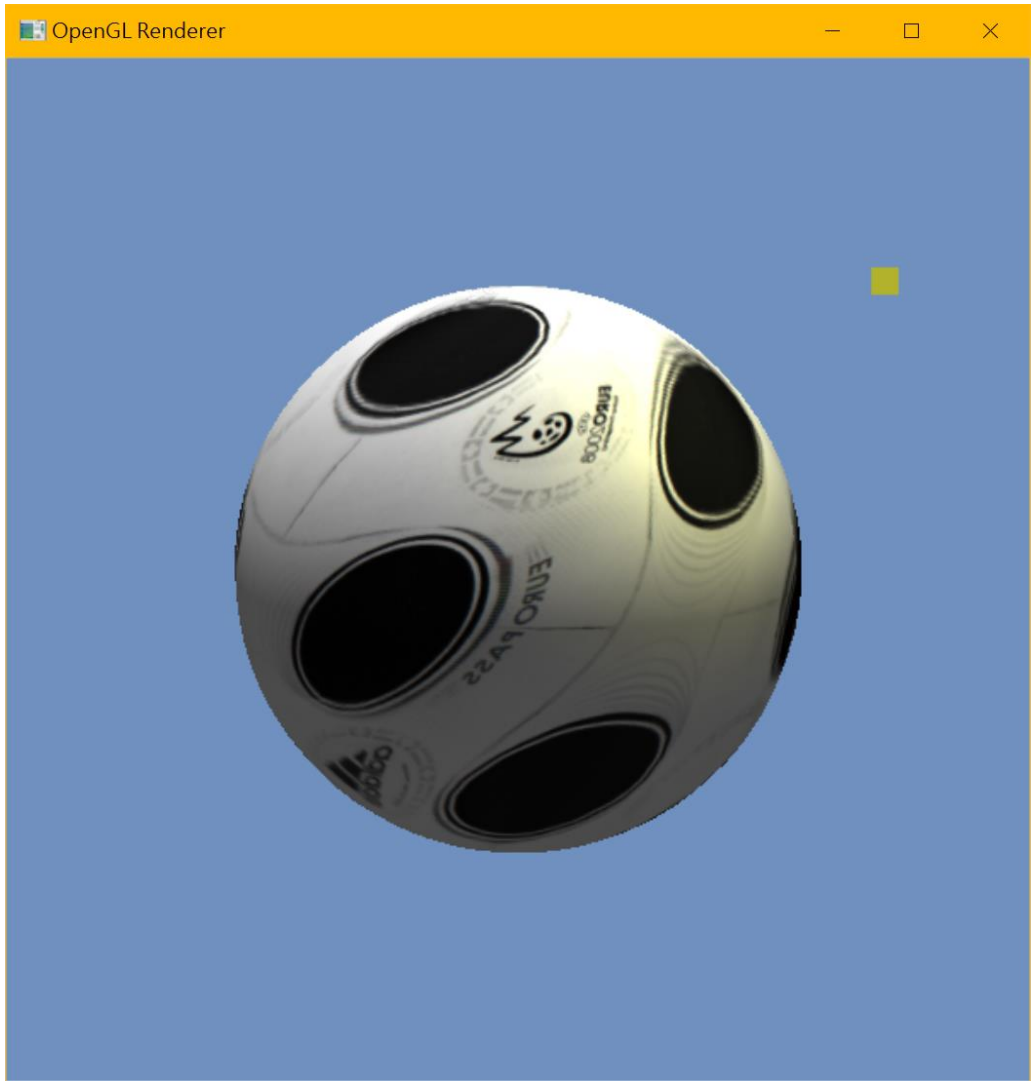
```
void RenderSceneCB()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render a triangle mesh with Gouraud shading. ------------------------------------------
    TriangleMesh* pMesh = sceneObj.mesh;
    if (pMesh ≠ nullptr) {
        // Update transform.
        // curRotationY += rotStep;
        glm::mat4x4 S = glm::scale(glm::mat4x4(1.0f), glm::vec3(1.5f, 1.5f, 1.5f));
        glm::mat4x4 R = glm::rotate(glm::mat4x4(1.0f), glm::radians(curRotationY), glm::vec3(0, 1, 0));
        sceneObj.worldMatrix = S * R;
        glm::mat4x4 normalMatrix = glm::transpose(glm::inverse(camera->GetViewMatrix() * sceneObj.worldMatrix));
        glm::mat4x4 MVP = camera->GetProjMatrix()

        gouraudShadingShader->Bind();

        // Transformation matrix.
        glUniformMatrix4fv(gouraudShadingShader->G
        glUniformMatrix4fv(gouraudShadingShader->G
        glUniformMatrix4fv(gouraudShadingShader->G
        glUniformMatrix4fv(gouraudShadingShader->G
        // Material properties.
        glUniform3fv(gouraudShadingShader->GetLocM
        glUniform3fv(gouraudShadingShader->GetLocM
        glUniform3fv(gouraudShadingShader->GetLocM
        glUniform1f(gouraudShadingShader->GetLocNs
        // Light data.
        if (dirLight ≠ nullptr) {
            glUniform3fv(gouraudShadingShader->Get
            glUniform3fv(gouraudShadingShader->GetLocDirLightRadiance(), 1, glm::value_ptr(dirLight->GetRadiance()));
        }
        if (pointLight ≠ nullptr) {
            glUniform3fv(gouraudShadingShader->GetLocPointLightPos(), 1, glm::value_ptr(pointLight->GetPosition()));
            glUniform3fv(gouraudShadingShader->GetLocPointLightIntensity(), 1, glm::value_ptr(pointLight->GetIntensity()));
        }
        glUniform3fv(gouraudShadingShader->GetLocAmbientLight(), 1, glm::value_ptr(ambientLight));
        // Texture data.
        if (sceneObj.tex ≠ nullptr) {
            imageTex->Bind(GL_TEXTURE0);
            glUniform1i(gouraudShadingShader->GetLocMapKd(), 0);
        }

        // Render the mesh.
        pMesh->Draw();

        gouraudShadingShader->UnBind();
    }
```

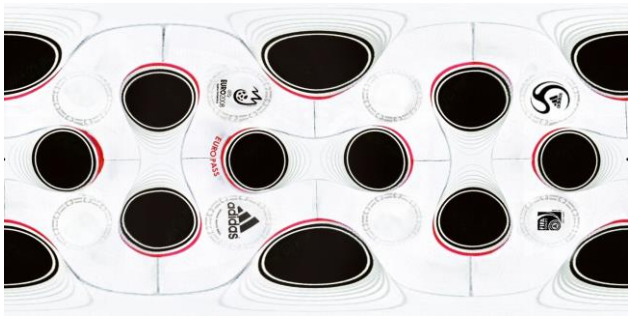
```
void ImageTexture::Bind(GLenum textureUnit)
{
    glActiveTexture(textureUnit);
    glBindTexture(GL_TEXTURE_2D, textureObj);
}
```

the n^{th} texture in the shader

```
// Texture data.
if (sceneObj.tex ≠ nullptr) {
    imageTex->Bind(GL_TEXTURE0);
    glUniform1i(gouraudShadingShader->GetLocMapKd(), 0);
}
```

Result





Practice:
Combine your **TriangleMesh** class in HW2


END OF PRESENTATION
ANY QUESTIONS?