Textures

Computer Graphics

Yu-Ting Wu

Outline

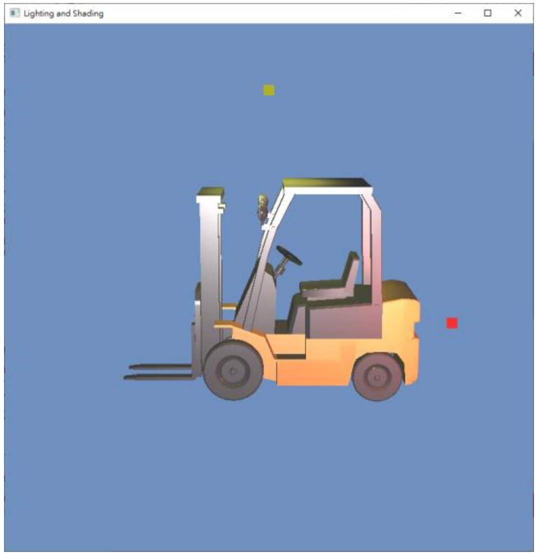
- Overview
- Texture data
- Texture filtering
- Applications
- OpenGL implementation

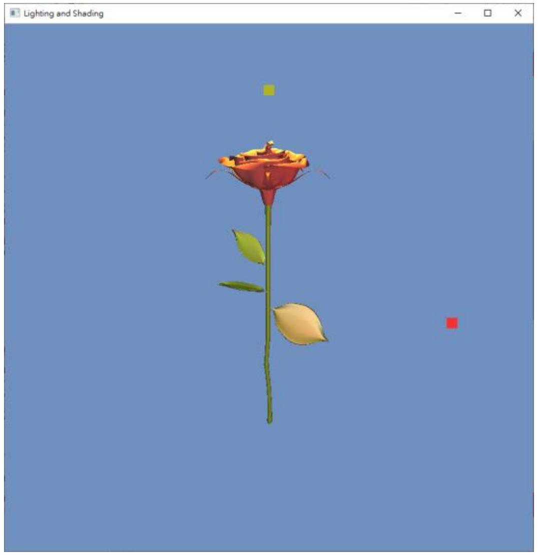
Outline

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

Why Do We Need Textures

- So far, we have described object colors using their reflectance functions
  - Subdivide an object into several parts, each has its reflectance properties (e.g., different diffuse and specular colors)



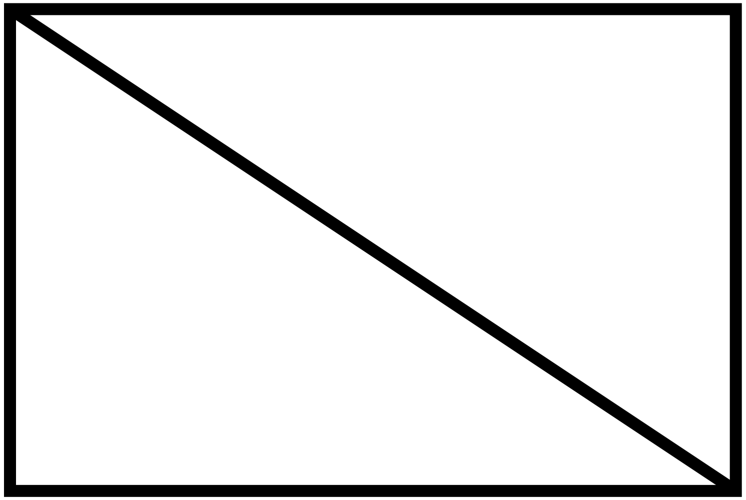


Why Do We Need Textures (cont.)

- Consider the following cases
    - Do we need (or can we) to finely subdivide the object?

Textures

- Can be used to represent **spatially-varying** data
- Can **decouple** materials from the geometry



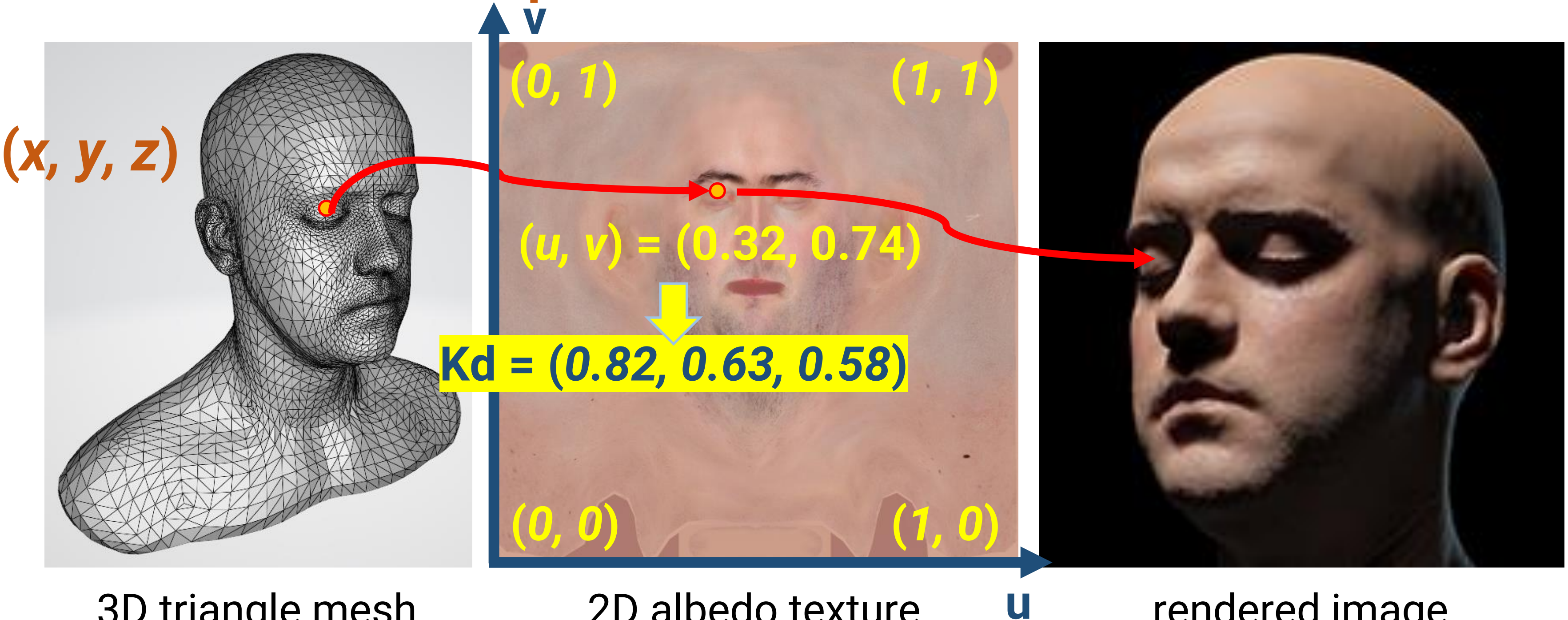
Geometry: two triangles

2D image texture for Kd
(spatially-varying material)

= **complex appearance**

Texture Coordinate

- A coordinate to look up the texture
- The way to map a point on an **arbitrary 3D surface** to a pixel (texel) on an **image** texture
  - Need **surface parameterization**



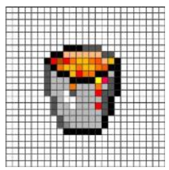
3D triangle mesh 2D albedo texture rendered image

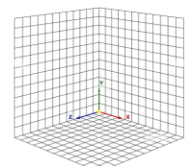
Texture Coordinate (cont.)

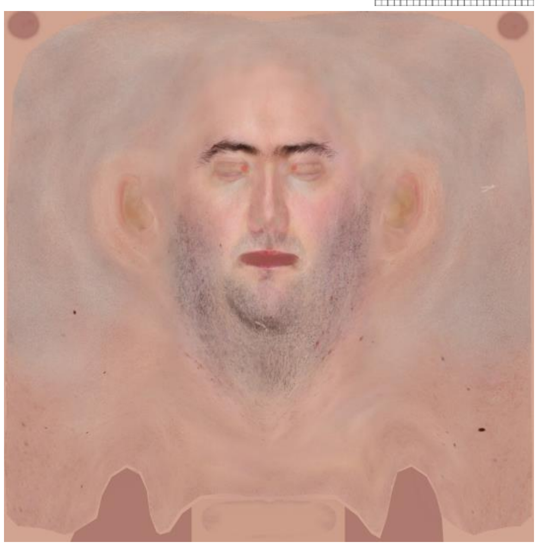
- A coordinate to look up the texture
- The way to map a point on an **arbitrary 3D surface** to a pixel (texel) on an **image** texture
  - Need **surface parameterization**
  - Usually produced by 3D artists

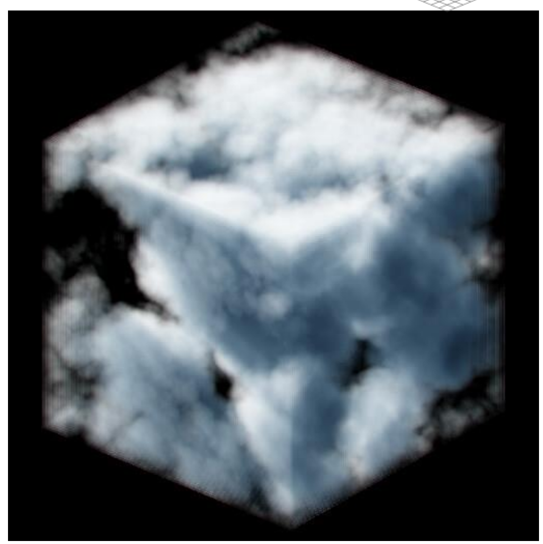
Types of Textures

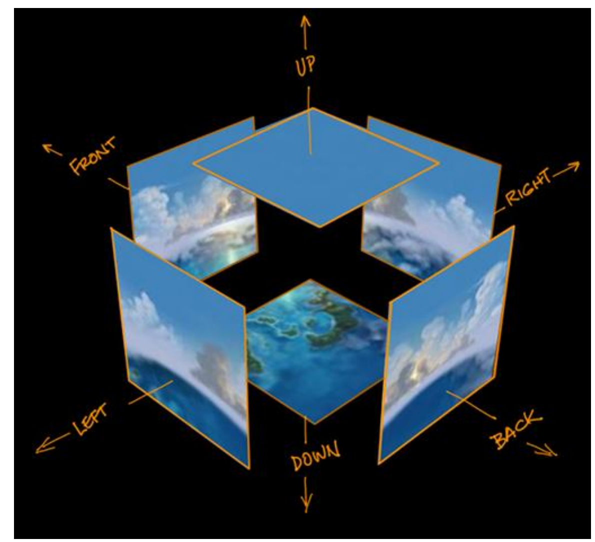
- **2D image texture (most common)**
- 3D volume texture
- Cubemap





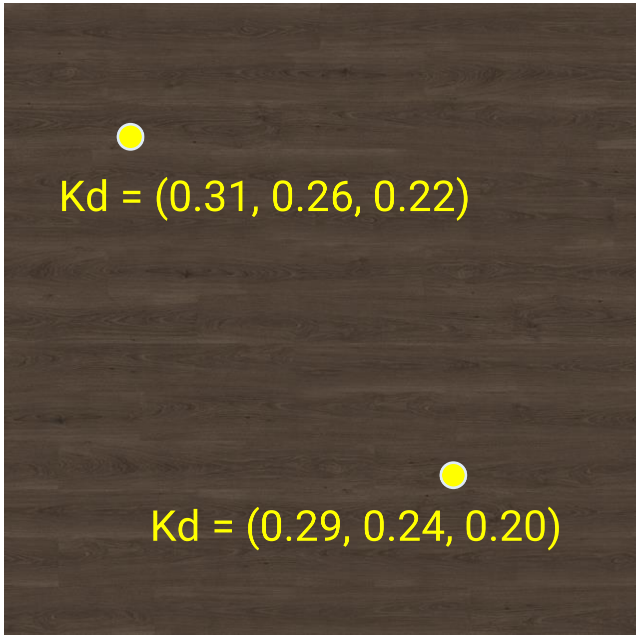






Textures (cont.)

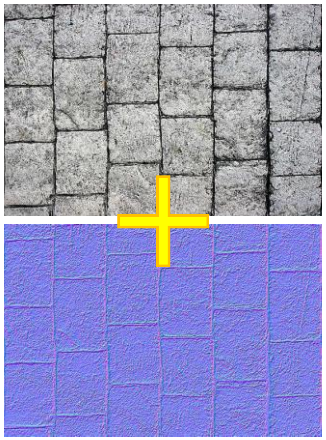
- 2D image texture for spatially-varying material

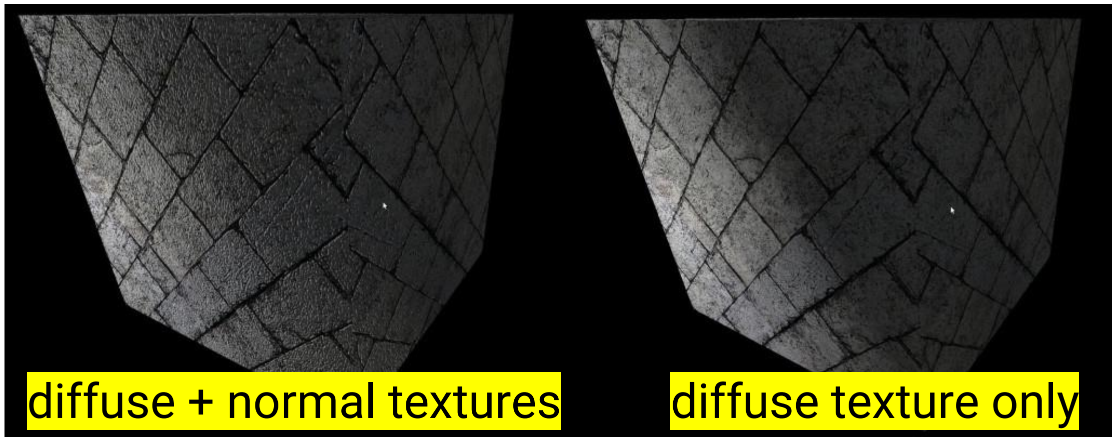


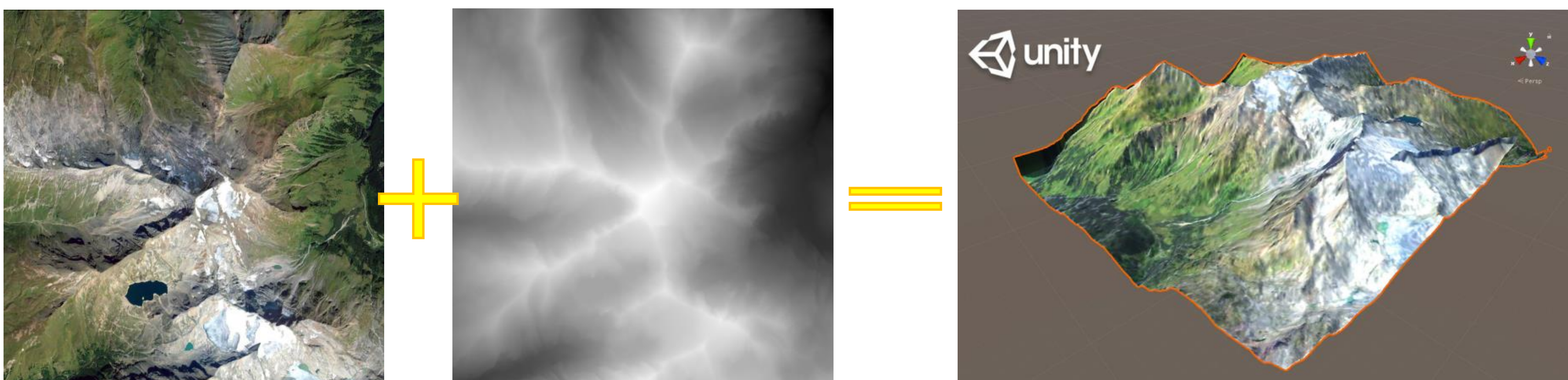
diffuse coefficient (Kd)

Types of Textures (cont.)

- **2D** image texture for **spatially-geometry data**

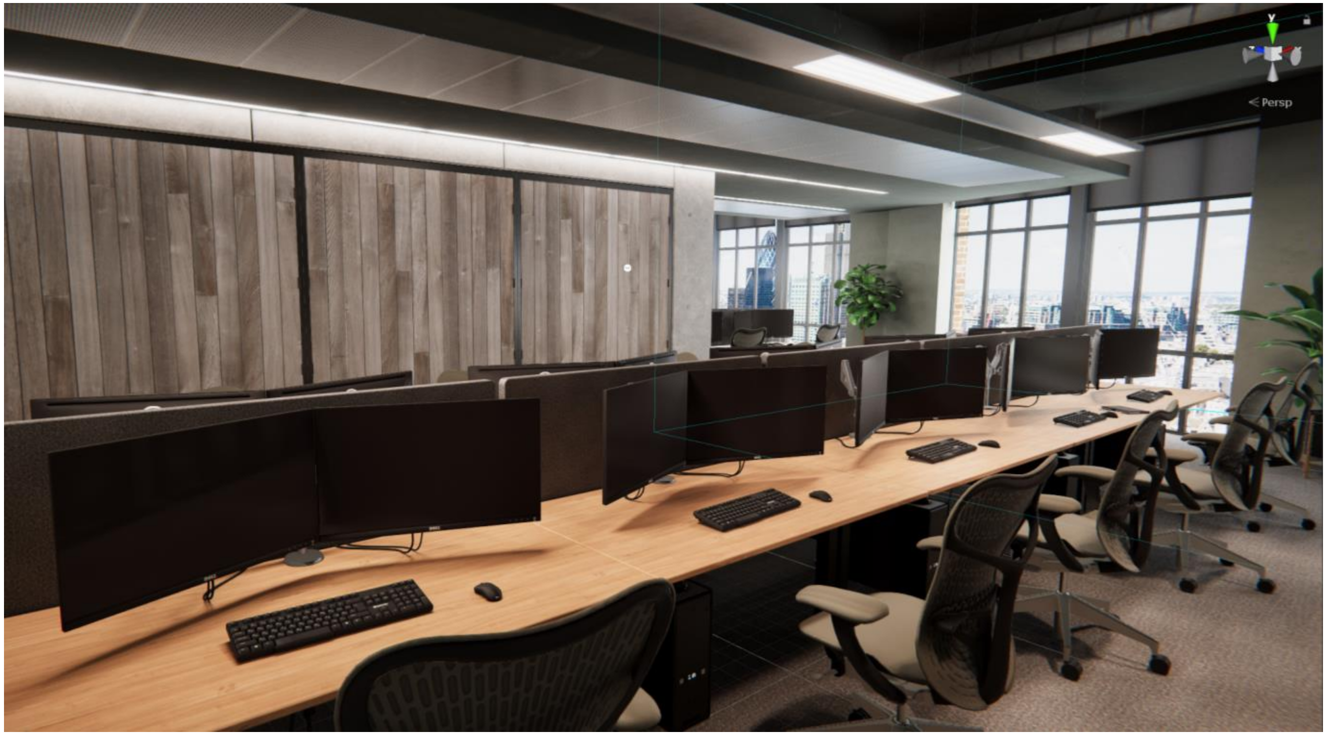




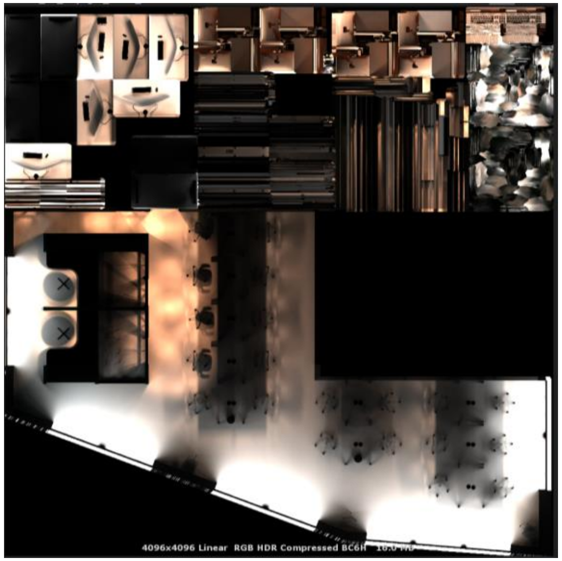


Types of Textures (cont.)

- 2D image texture for precomputed lighting data



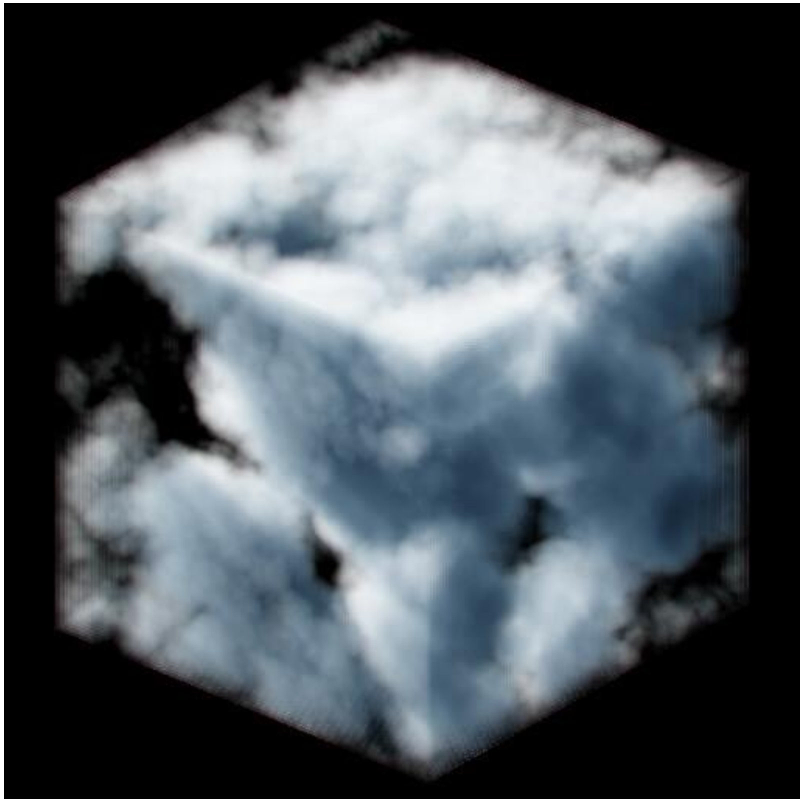
real-time rendered result

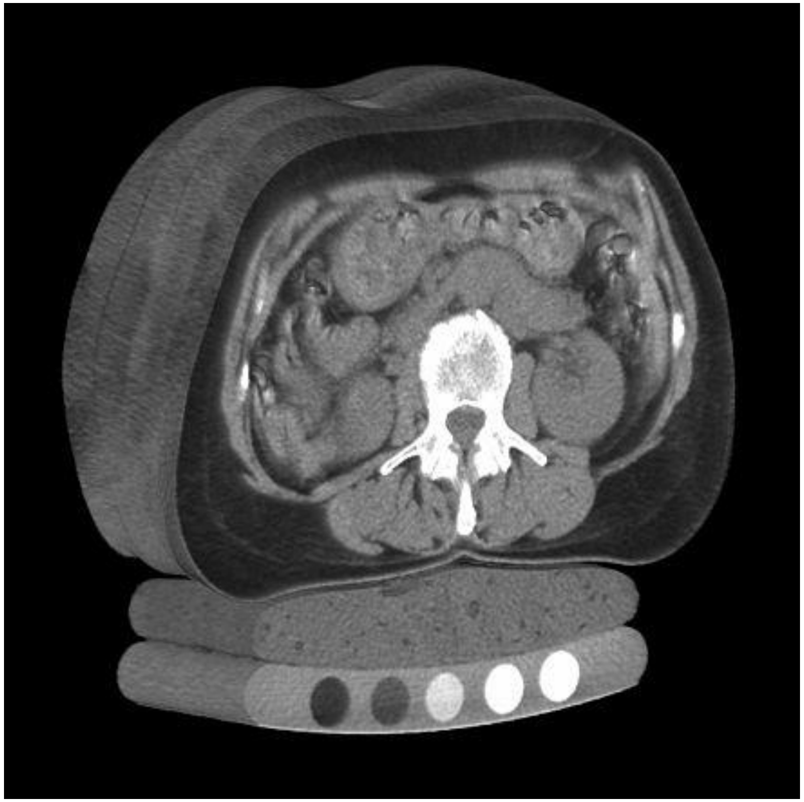


precomputed lightmaps

Types of Textures (cont.)

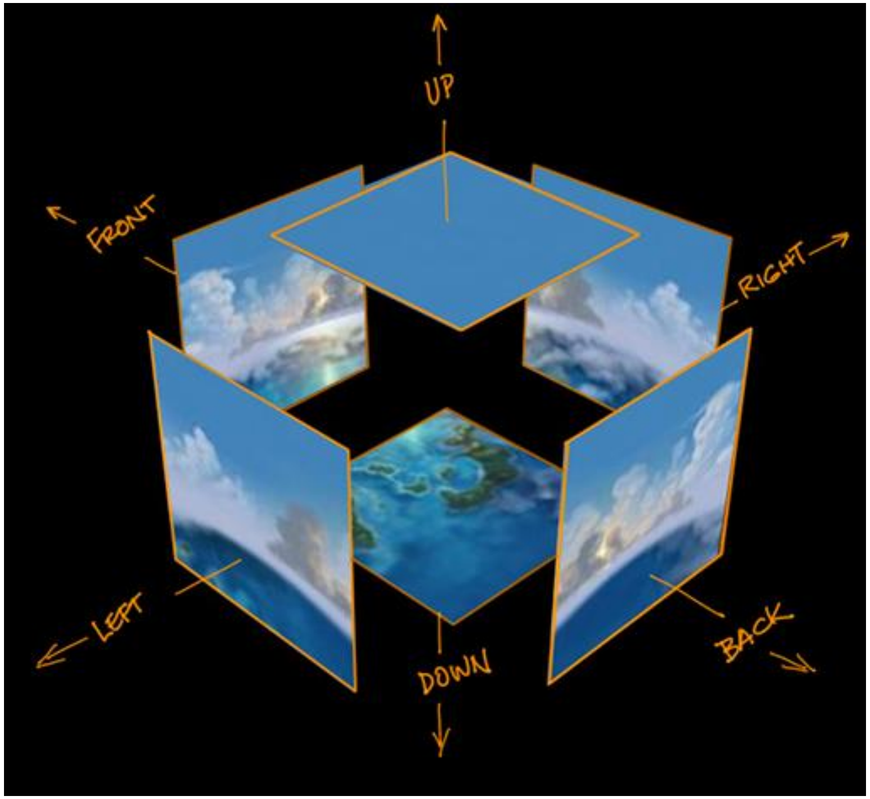
- 3D volume texture
  - Lookup by a 3D texture coordinate (u, v, s)

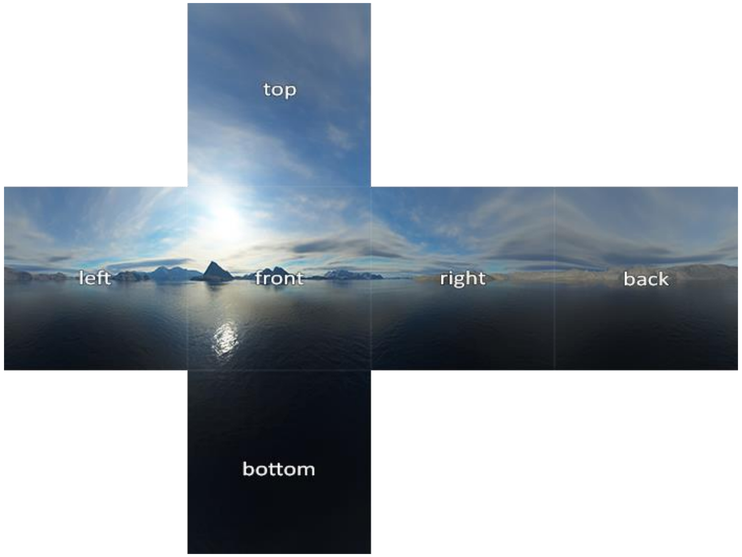




Types of Textures (cont.)

- Cubemap





Outline

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

Texture Data in Wavefront OBJ File

- TexCube.obj

TexCube.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

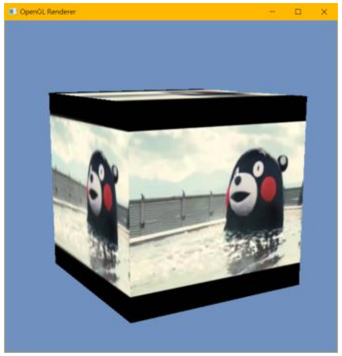
```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
mtllib TexCube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000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

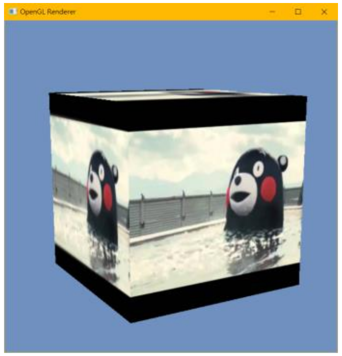
```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

TexCube.mtl - 記事本

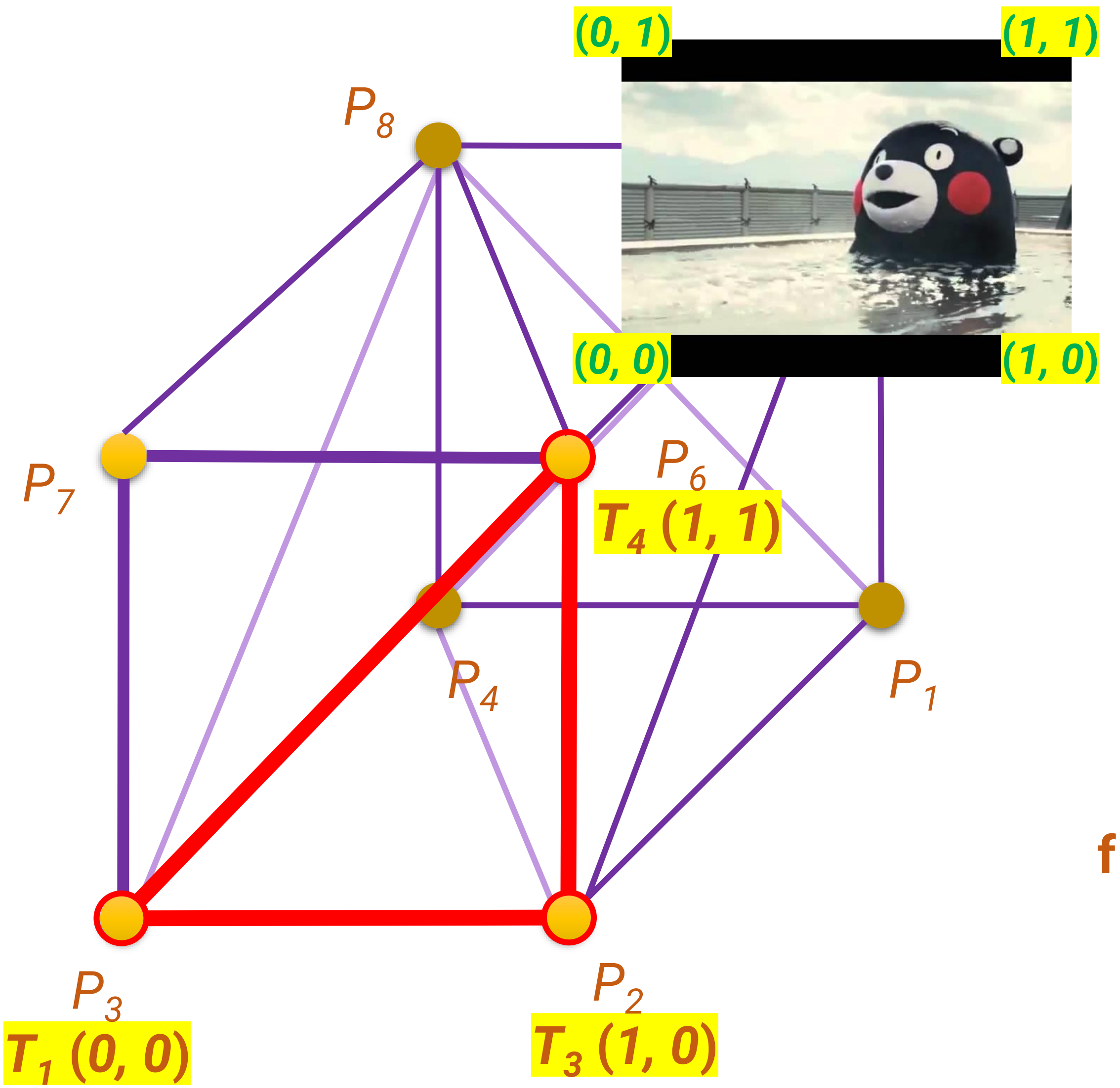
檔案(F) 編輯(E) 格式(O) 檢視

```
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



Interpret the Texture Data



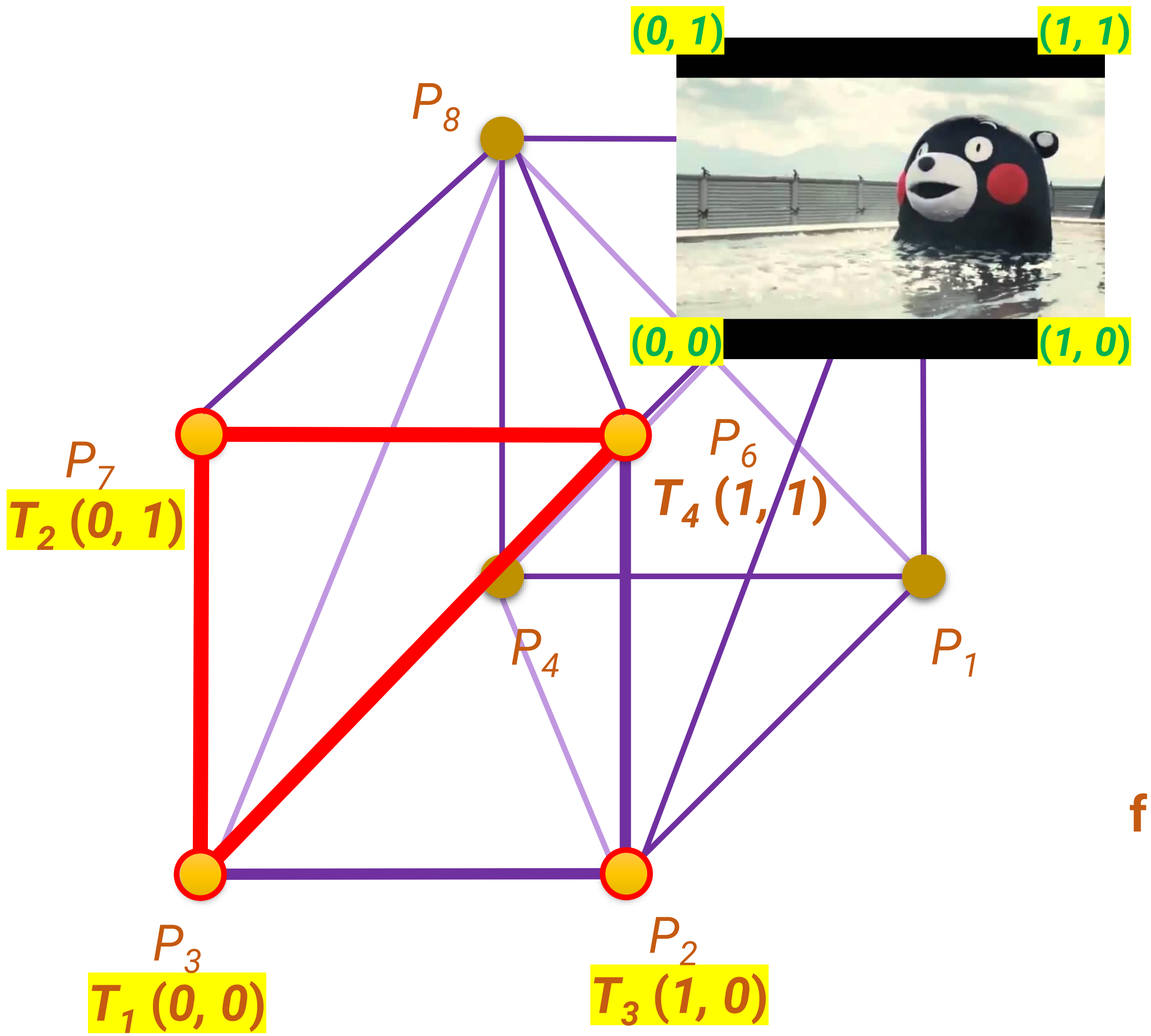

```
vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0
```

```
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
```

	vertex1	vertex2	vertex3
f	P/T/N	P/T/N	P/T/N

P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Interpret the Texture Data (cont.)




```
vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0
```

```
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
```

	vertex1	vertex2	vertex3
f	**P/T/N**	**P/T/N**	**P/T/N**

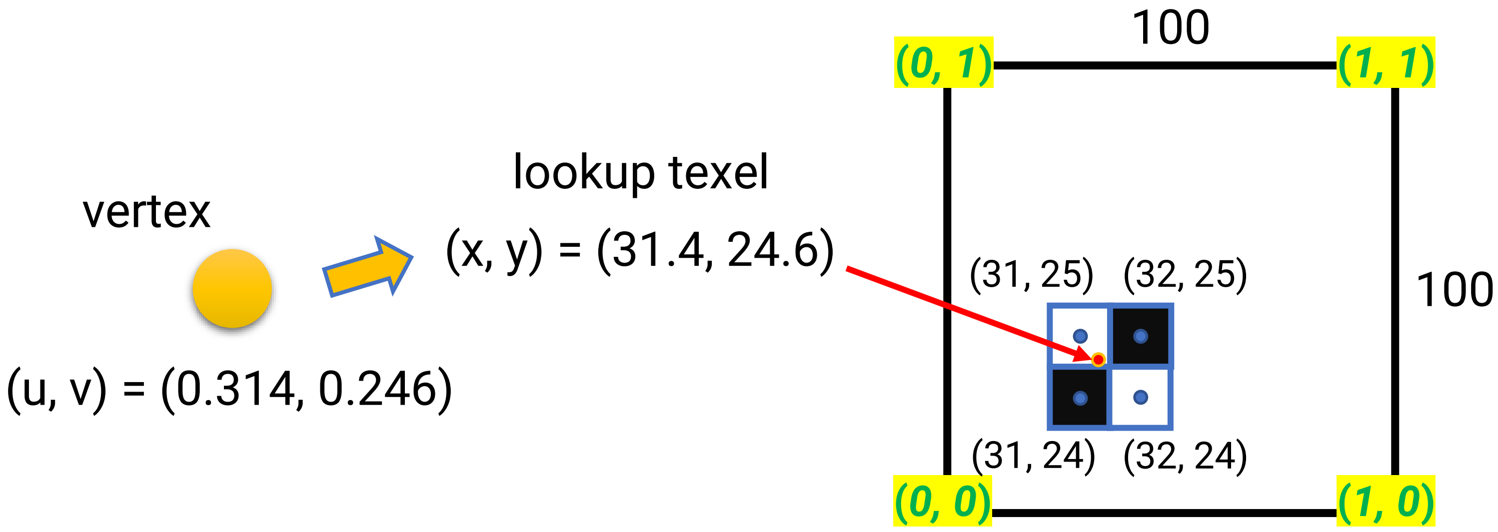
P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Outline

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

Texture Filtering

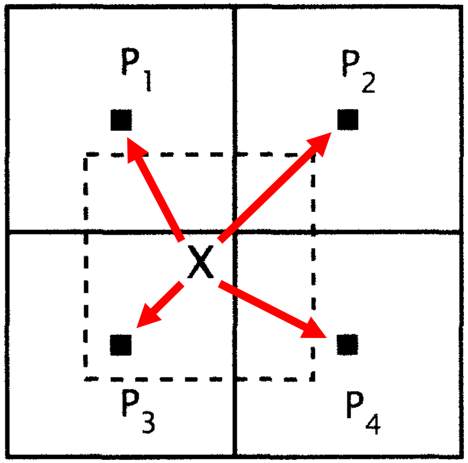
- Like an image, the content in a 2D texture is **discretely** represented by texels
- The texture coordinates can be **continuous** (especially after interpolation by the rasterization)
- How to determine the texture value if the lookup point is not at the center of a texel?



Texture Filtering (cont.)

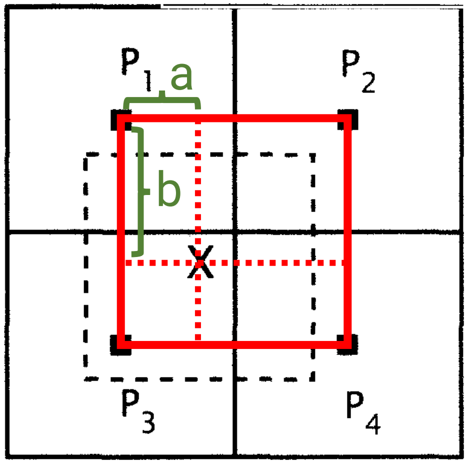
- Strategies
  - **Nearest neighbor**
  - **Bilinear interpolation**

PXU = (1-a)P1 + (a)P2
PXD = (1-a)P3 + (a)P4
PX = (1-b)PXU + (b)PXD



nearest neighbor

P_3 is closest
Use P_3's pixel value



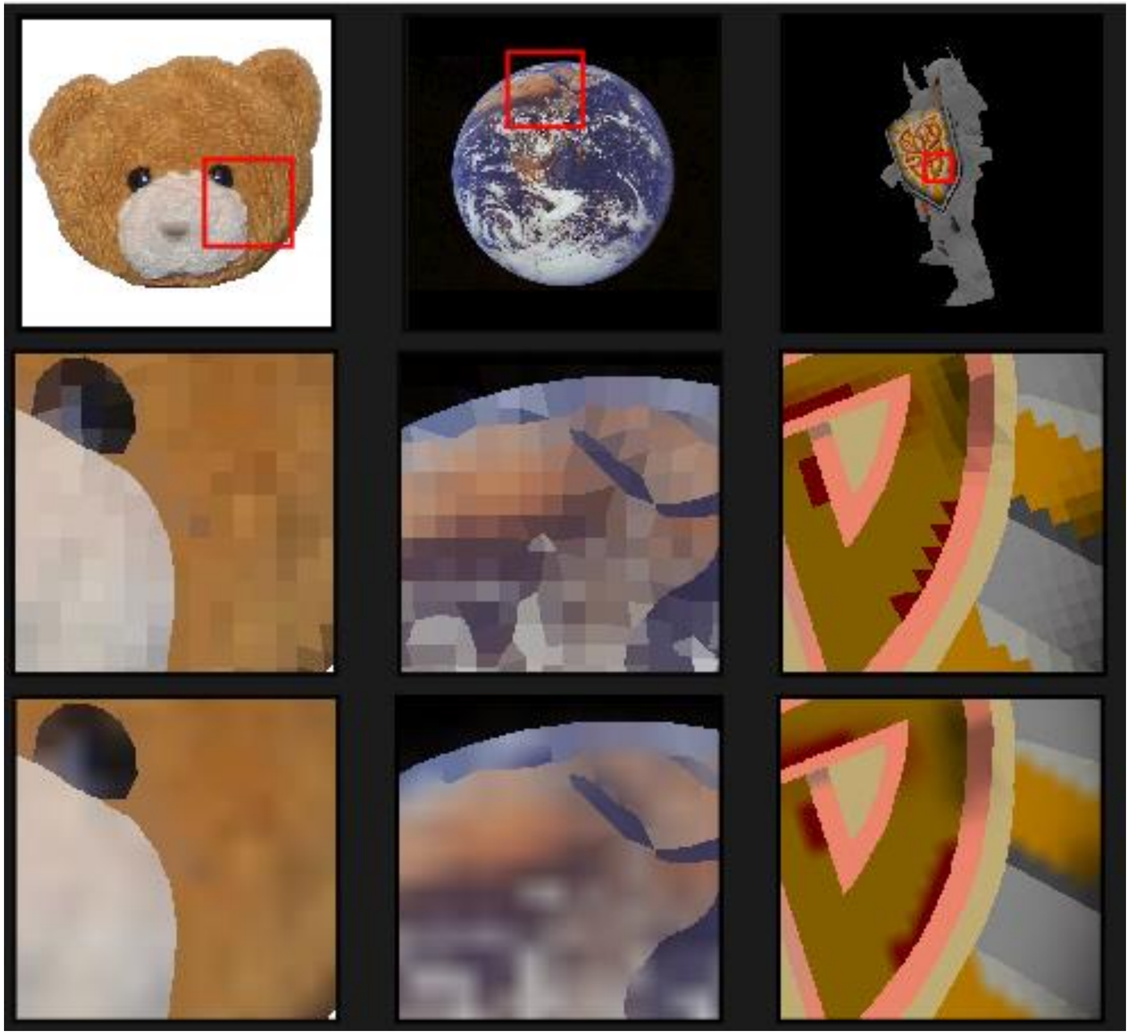
bilinear interpolation

$(1\text{-}a)(1\text{-}b)P_1 + (a)(1\text{-}b)P_2 +$
$(1\text{-}a)(b)P_3 + (a)(b)P_4$

Texture Filtering (cont.)

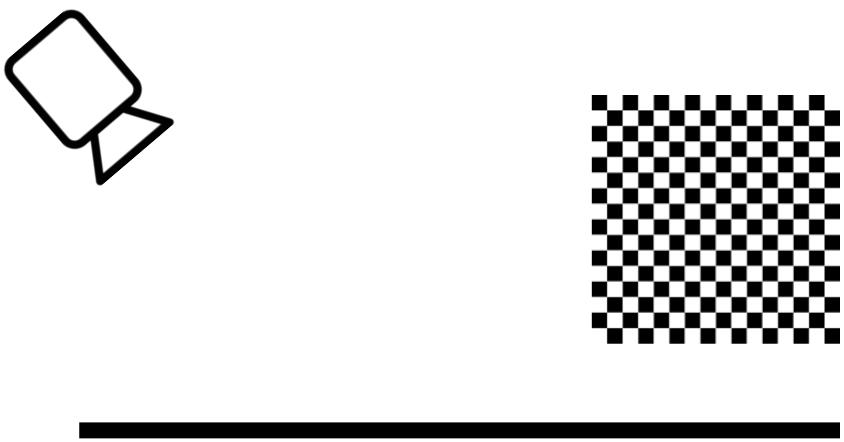
nearest neighbor

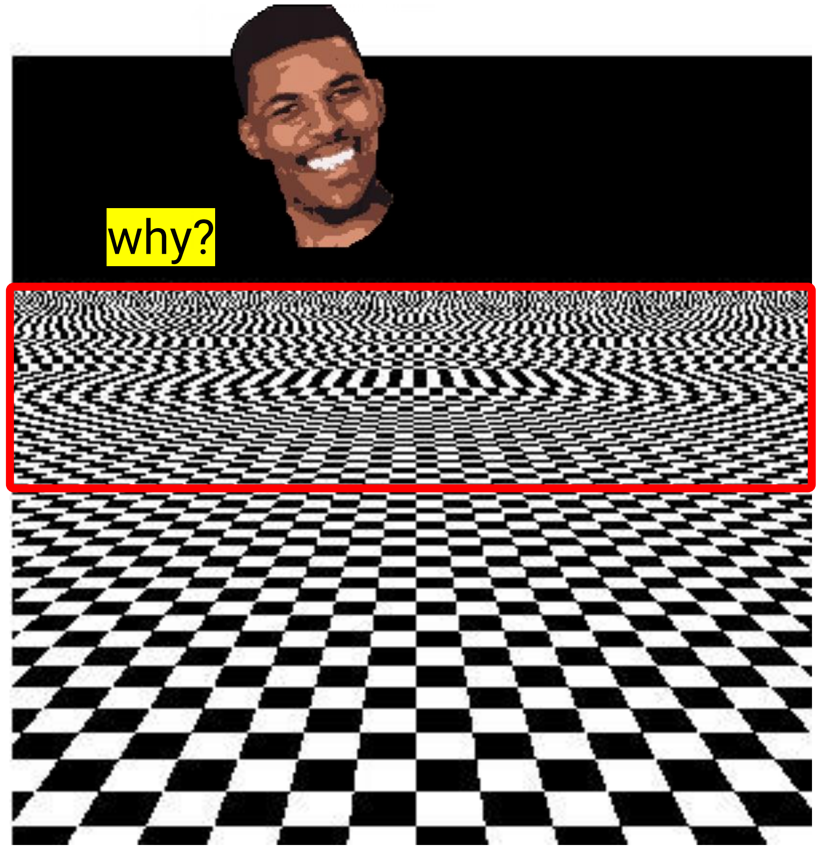
bilinear interpolation



Problems with Texture Mapping

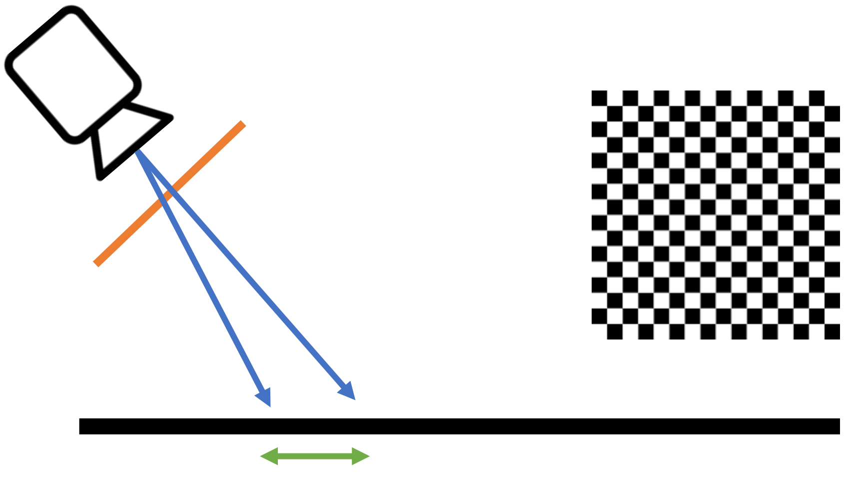
- Consider the following plane with a check-board pattern texture

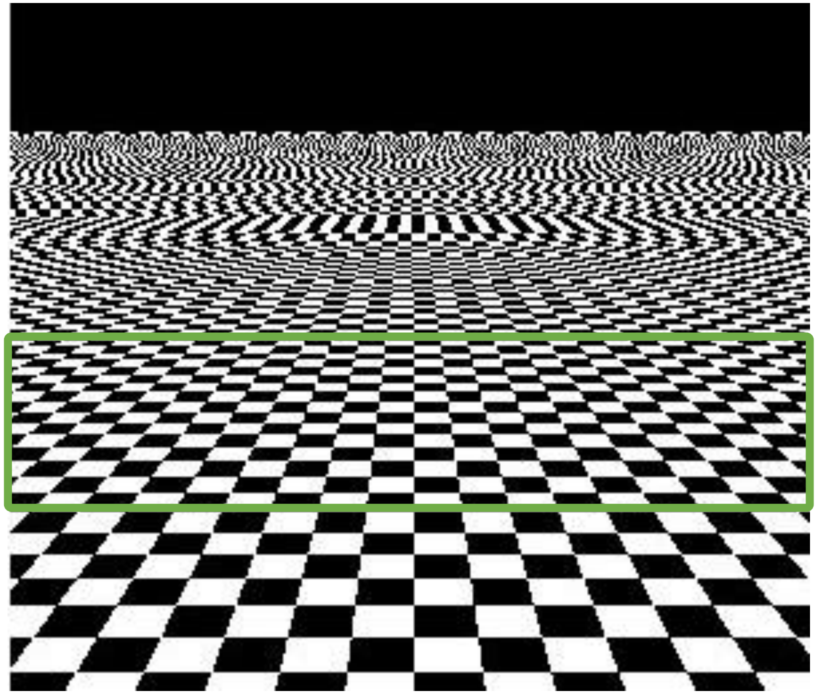




Texture Aliasing (cont.)

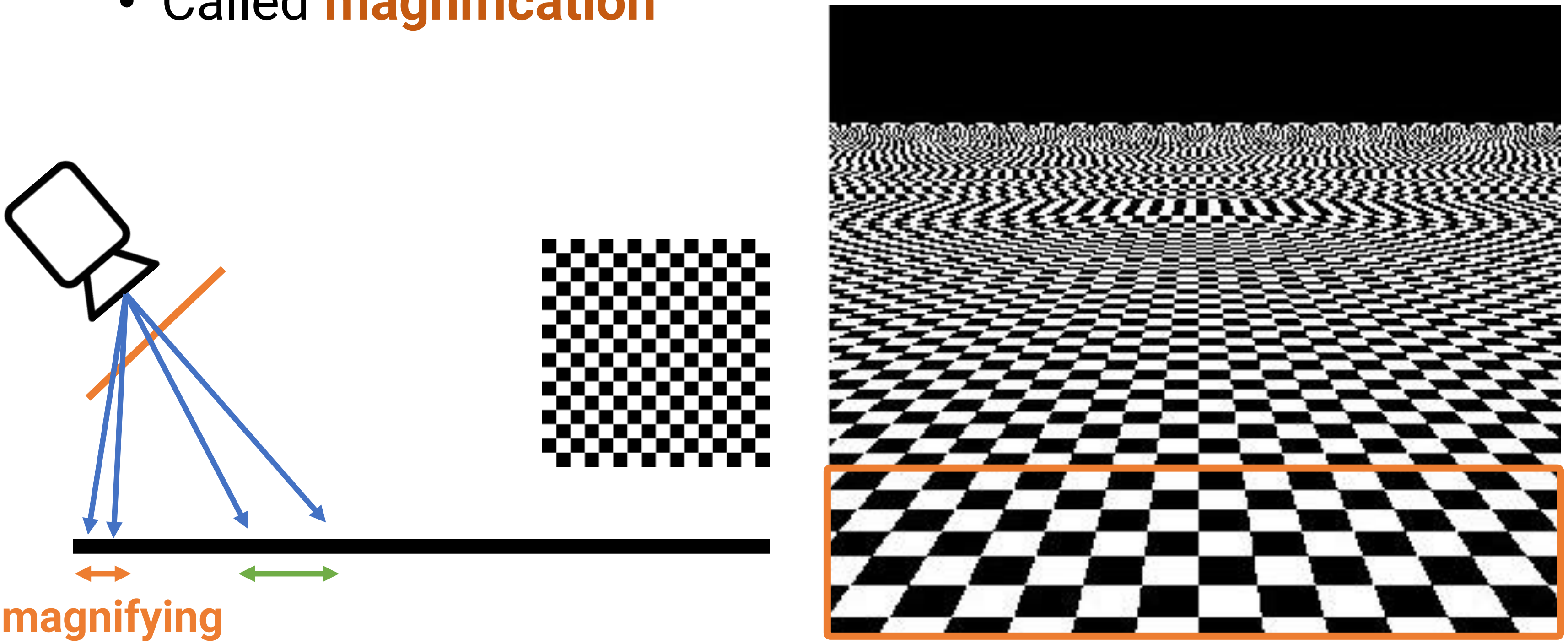
- Example
  - For the **green** area, one pixel covers a surface that is roughly one texel in the texture





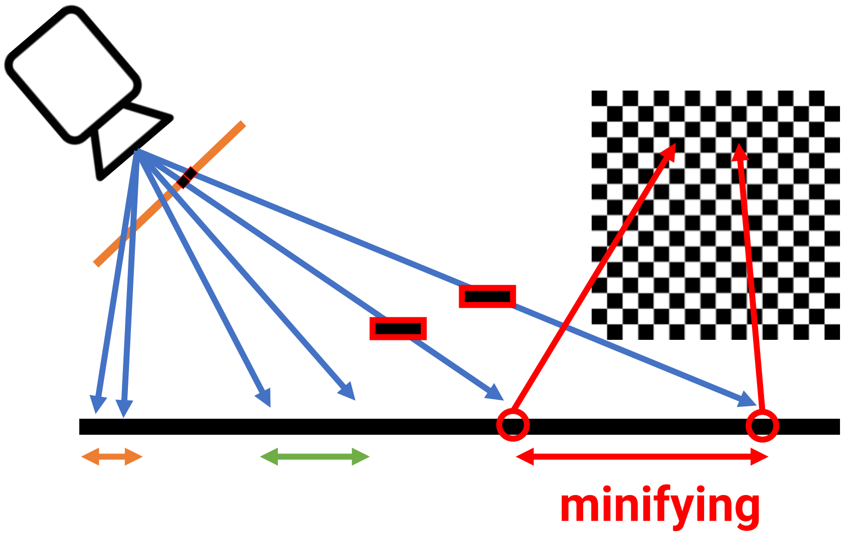
Texture Aliasing (cont.)

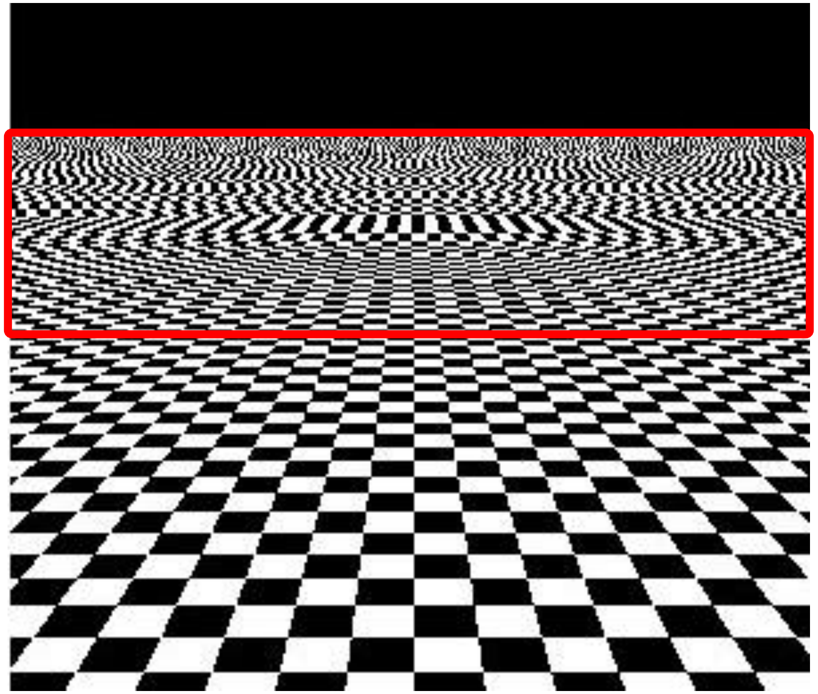
- Example
  - For the **orange** area, one pixel covers a surface that is **smaller** than one texel in the texture
  - Called **magnification**



Texture Aliasing (cont.)

- Example
  - For the **red** area, one pixel covers a surface that is **larger** than one texel in the texture
  - Called **minification**



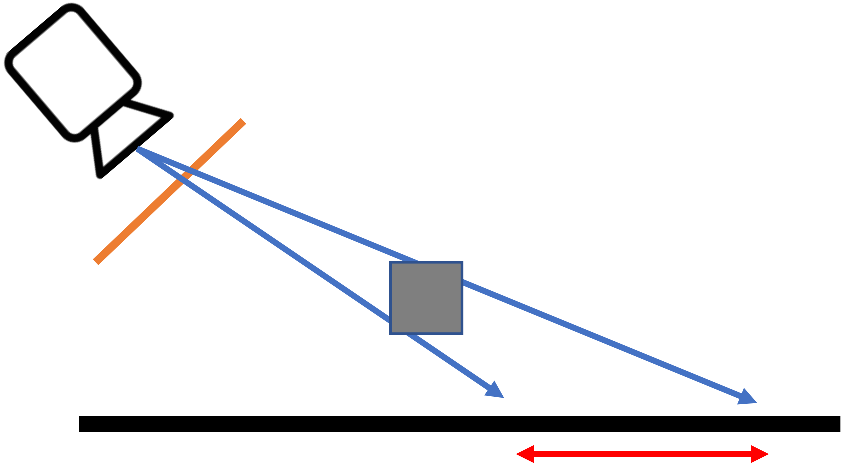


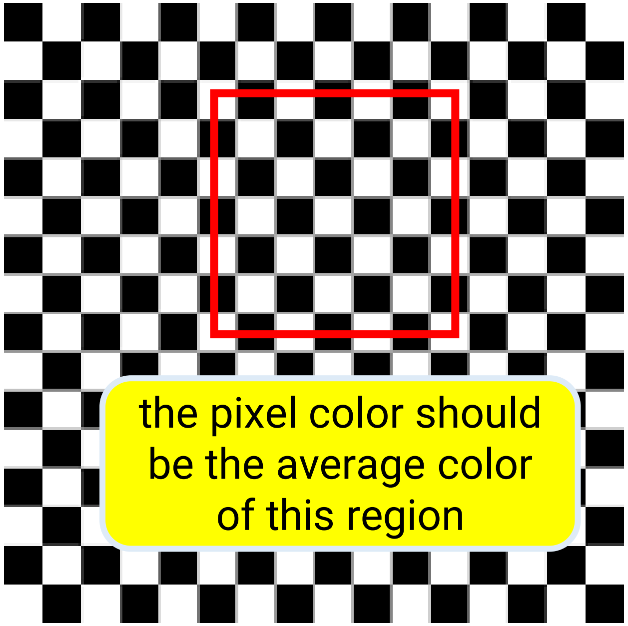
Texture Aliasing (cont.)

- Example
  - For the **red** area, one pixel covers a surface that is **larger** than one texel in the texture
  - Called **minification**
  - Might produce **flickering** for distant objects

Mipmap

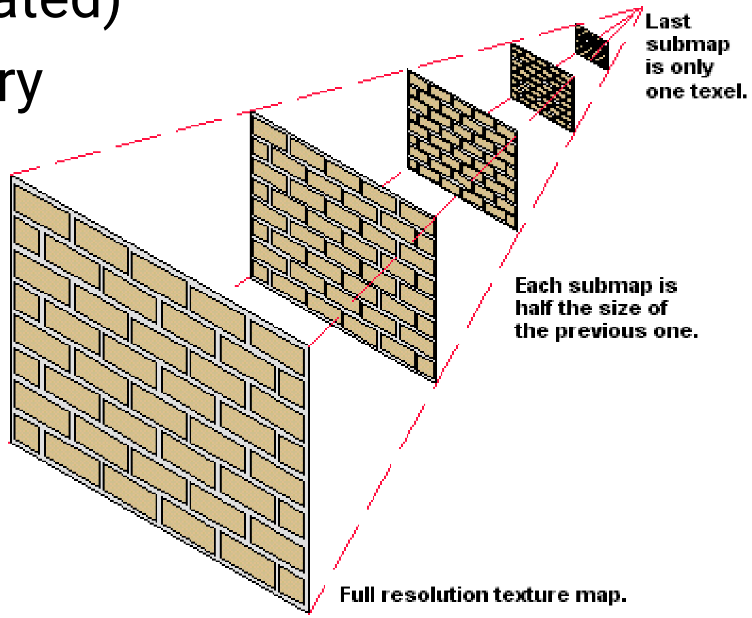
- To avoid aliasing, we should determine the regions a pixel covers (footprint) and average all the texture values inside the regions
- Time-consuming to do this in the run time!





Mipmap (cont.)

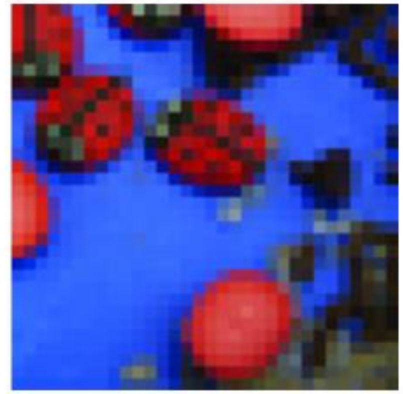
- Mipmap provides a clever way to solve this problem
- **Pre-process**
  - Build a **hierarchical representation** of the texture image
  - Each level has a half resolution of its previous level (generated by linearly interpolated)
  - Take at most **1/3** more memory

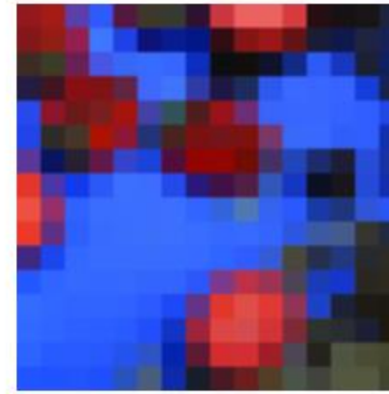


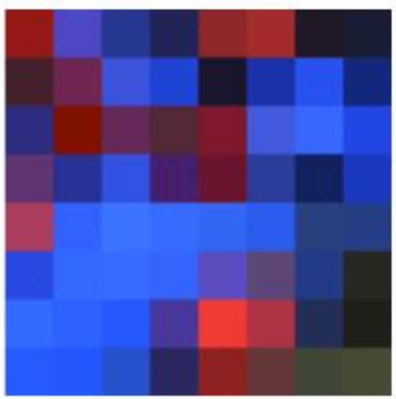
Mipmap (cont.)

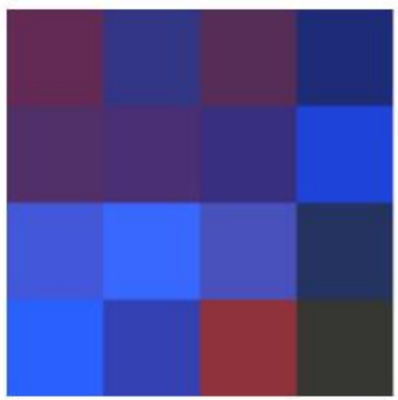

Level 0 = 128x128

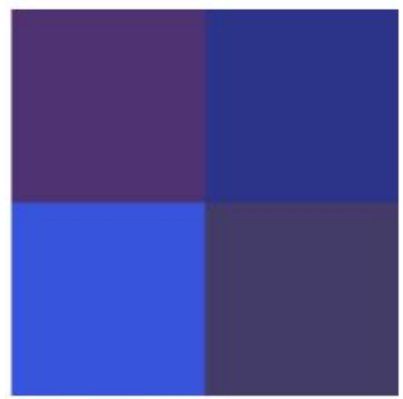

Level 1 = 64x64

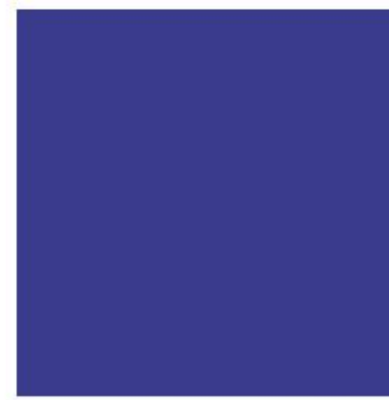

Level 2 = 32x32


Level 3 = 16x16


Level 4 = 8x8


Level 5 = 4x4

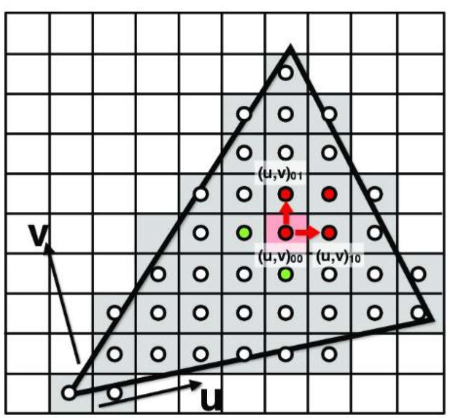

Level 6 = 2x2


Level 7 = 1x1

Mipmap (cont.)

- **Run-time lookup**
  - Use **screen-space texture coordinate** to estimate its footprint in the texture space
  - Choose two levels ***l*** and ***l+1*** based on the footprint
  - Perform linear interpolation at level ***l*** to obtain a value V_D
  - Perform linear interpolation at level ***l+1*** to obtain V_{D+1}
  - Perform linear interpolation between V_D and V_{D+1}

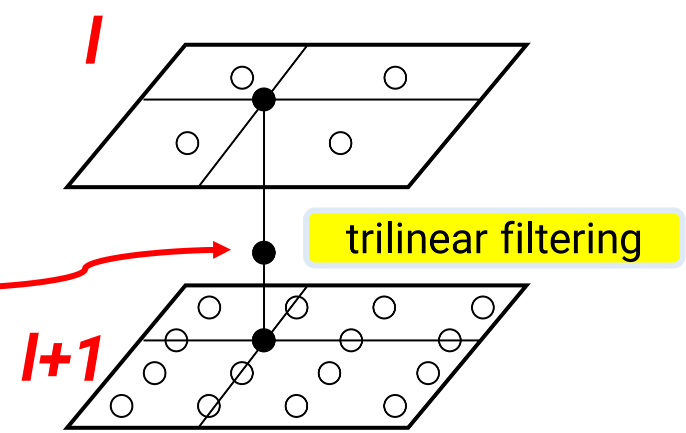
trilinear



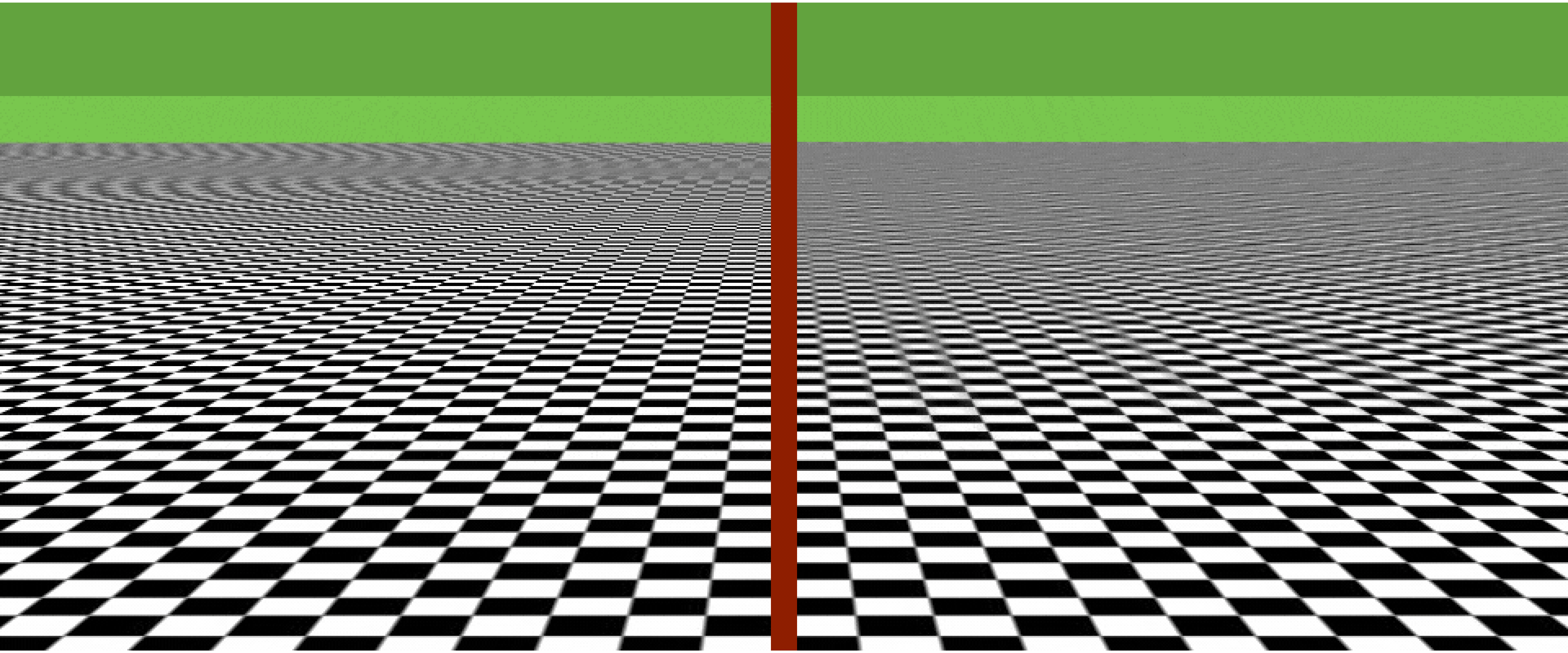
level	Res.
0	2^{n-1}
1	
:	:
l	2^{n-1-l}
:	:
n-1	1

$$\frac{1}{w} = 2^{n-1-l}$$

$$l = n - 1 + \log w$$



Mipmap (cont.)



without mipmap　　　　　　with mipmap

Mipmap (cont.)


tom's hardware guide
No Mipmapping
tom's hardware guide
With Mipmapping
tom's hardware guide
Colored Mipmaps

Outline

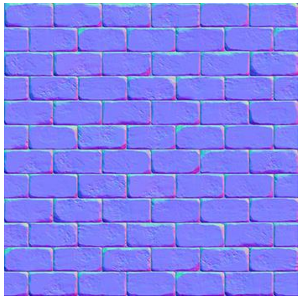
- Overview
- Texture data
- Texture filtering
- **Applications**
- OpenGL implementation

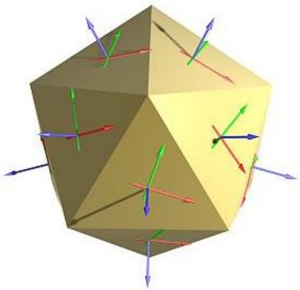
Normal Mapping

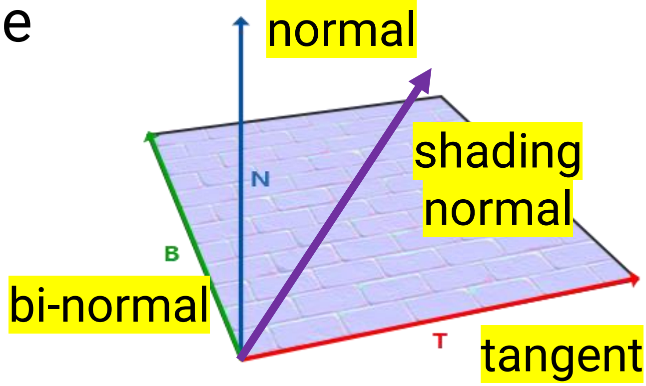
- Improve geometry details without adding vertices and triangles
  - Reduce the time of geometry processing
  - Only increase shading cost
  - Can also shorten the efforts of producing assets

Normal Mapping (cont.)

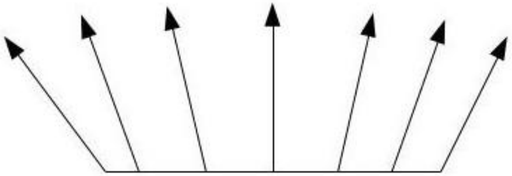
- Encode normal as texture color
  - (nx, ny, nz) = normalize(2 * TexColorRGB – 1)
  - The normal is defined in ***TBN*** space

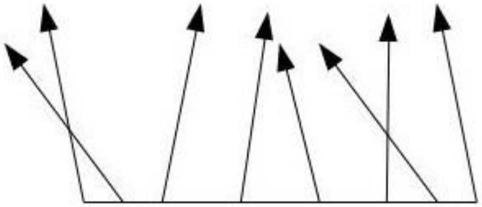






- During rendering, use shading normal instead of geometry normal





Normal Mapping (cont.)

- Recap: build camera matrix with viewing direction, right vector, and up vector

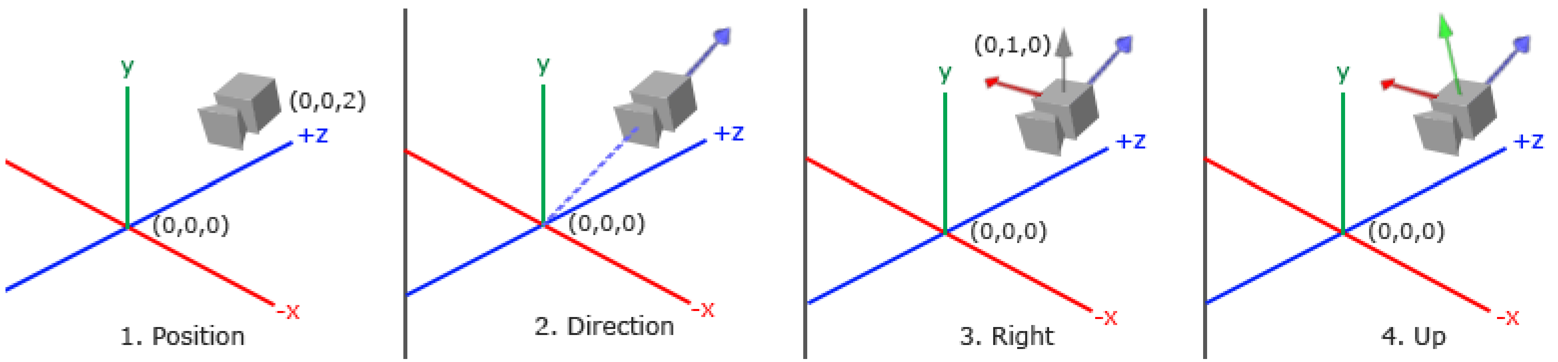
right vector

up vector

viewing vector

$$\begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation matrix translation matrix



Normal Mapping (cont.)

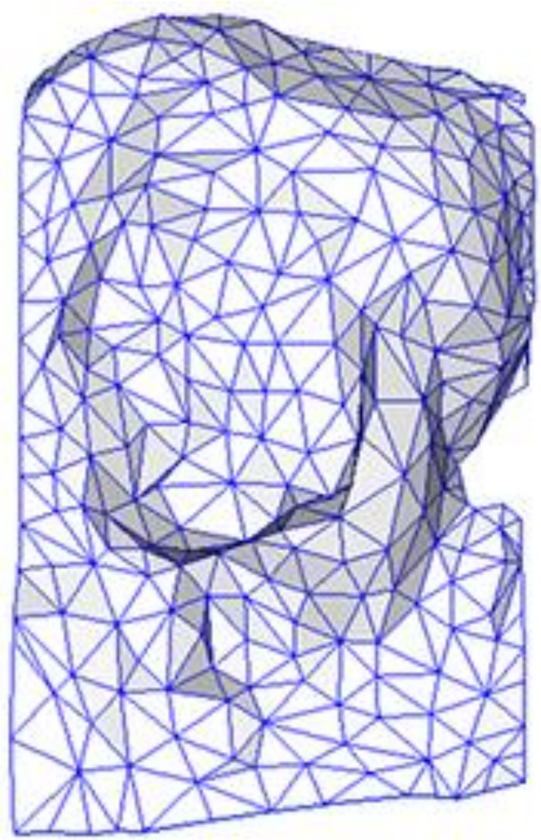
- Implementation
  - Calculate vertex tangent and bitangent as new vertex attributes
    - Calculate **per-face tangent** and **bi-normal** and obtain **per-vertex tangent** and **bi-normal** by averaging the face tangents of all adjacent faces
  - In the shader, build a ***TBN*** matrix and use it to transform the normal

tangent vector	T_x	T_y	T_z
bi-normal vector	B_x	B_y	B_z
normal vector	N_x	N_y	N_z

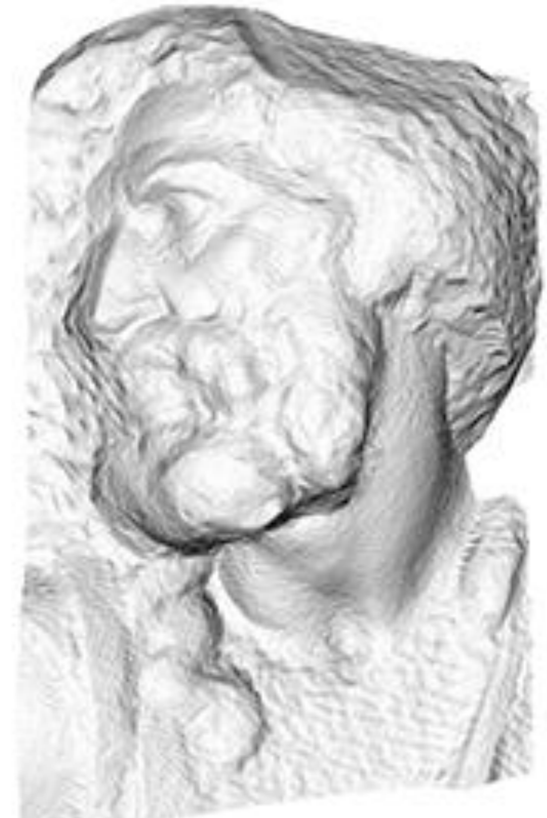
$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

Normal Mapping (cont.)

original mesh
4M triangles



simplified mesh
500 triangles



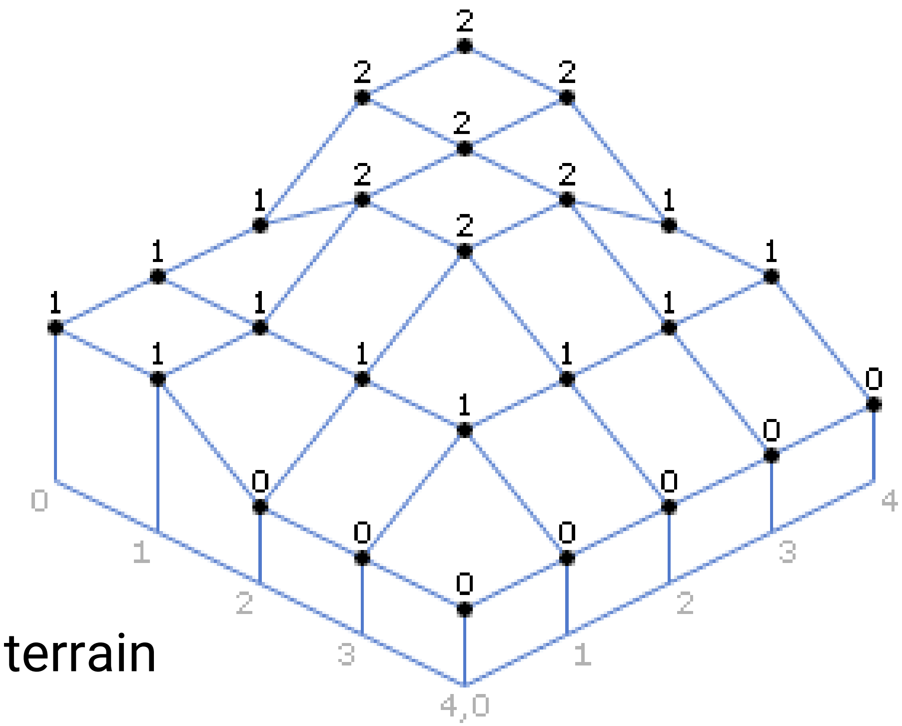
simplified mesh
and normal mapping
500 triangles

Height Map

- Use a scalar texture to represent the **vertex displacement** along the surface normal of a **base mesh**
- Widely used for **terrain** design

	0	1	2	3	4
0	1	1	1	2	2
1	1	1	2	2	2
2	0	1	2	2	1
3	0	1	1	1	1
4	0	0	0	0	0

heightmap

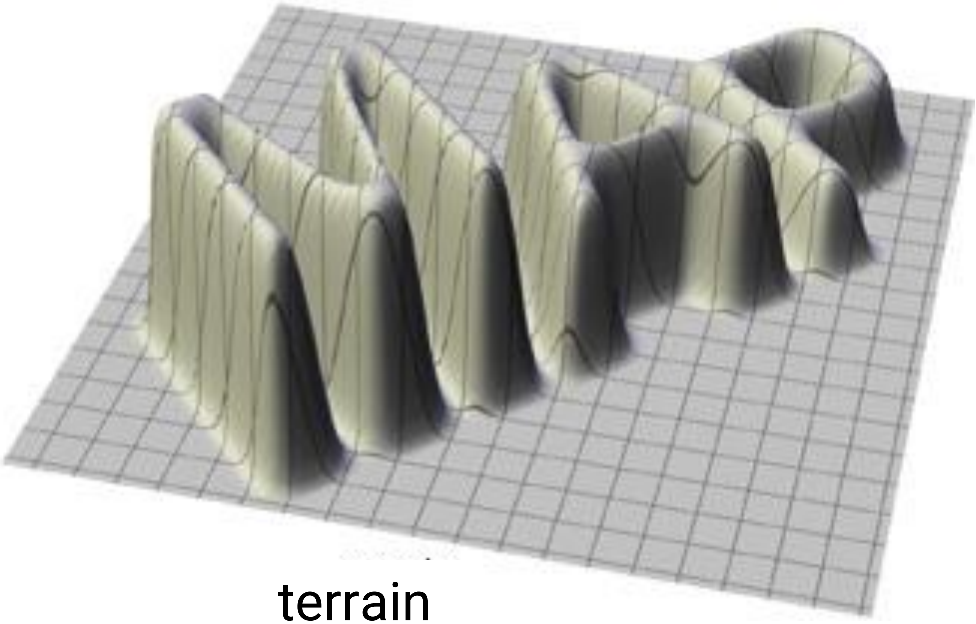


terrain

Height Map (cont.)

- Use a scalar texture to represent the **vertex displacement** along the surface normal of a **base mesh**
- Perturb vertex position in the **vertex shader**
- Widely used for **terrain** design

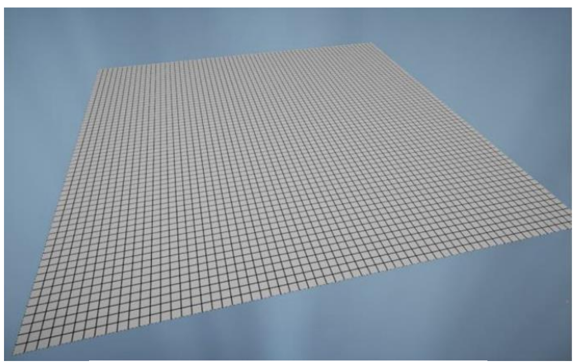
heightmap



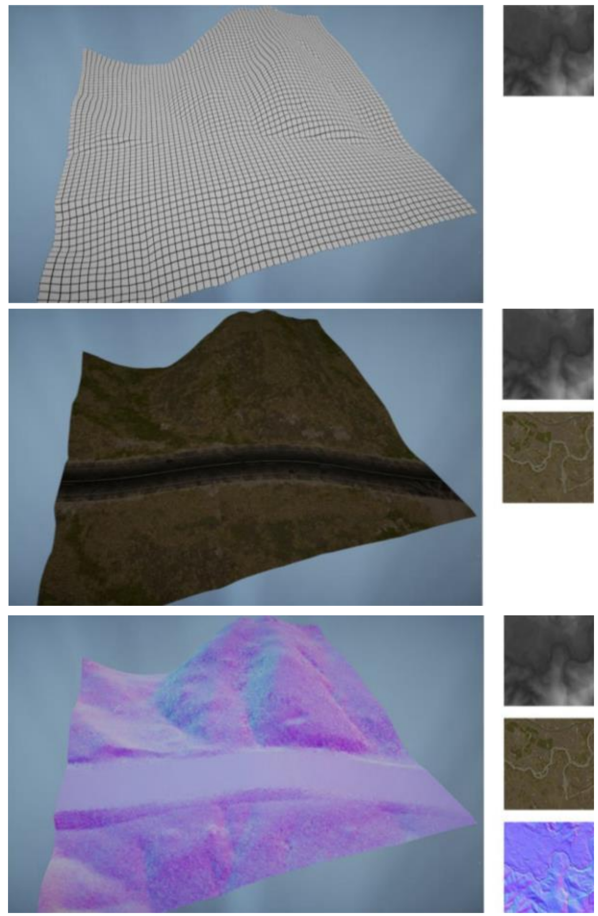
terrain

Height Map (cont.)

- Usually combined with an albedo texture and a normal map for shading



base mesh



rendered terrain

Height Map (cont.)

- Terrain management in ***FarCry 5***

Height Map (cont.)

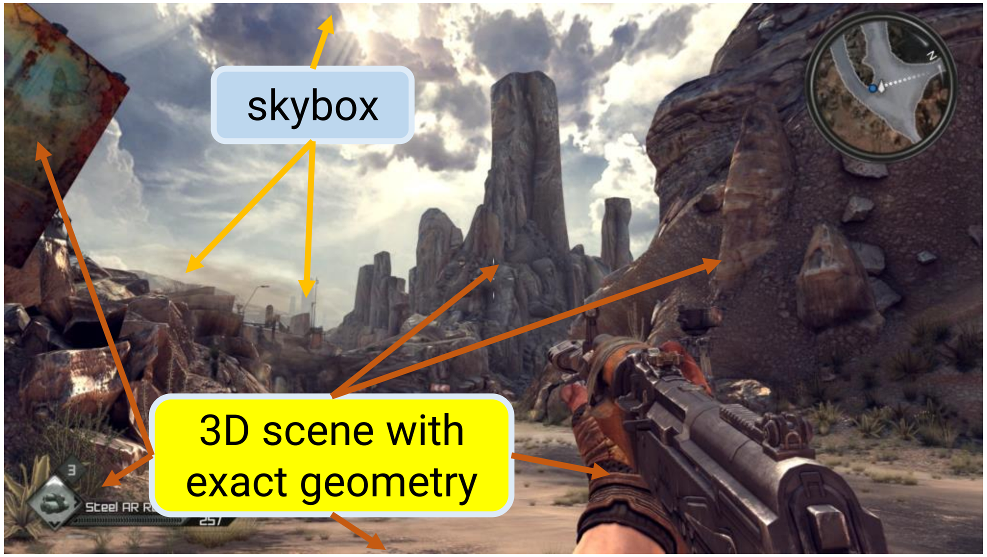
- Implementation
  - For each vertex in the base mesh, lookup the **height map** to displace the vertex (in the Vertex Shader)

    *new vertex position = original vertex position + normal * height*

  - For each fragment, lookup the **normal map** for the detailed shading normal and the **albedo texture** for the material property (in the Fragment Shader)

Skybox

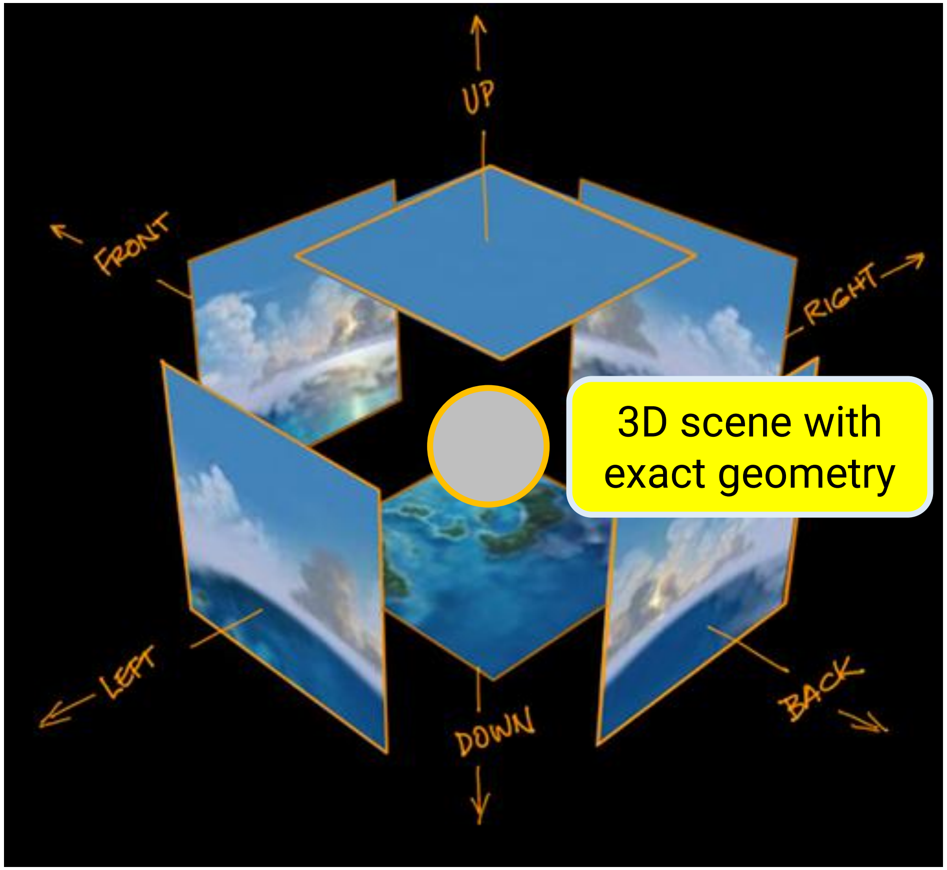
- Use a texture-mapped simple proxy geometry to represent far-away objects

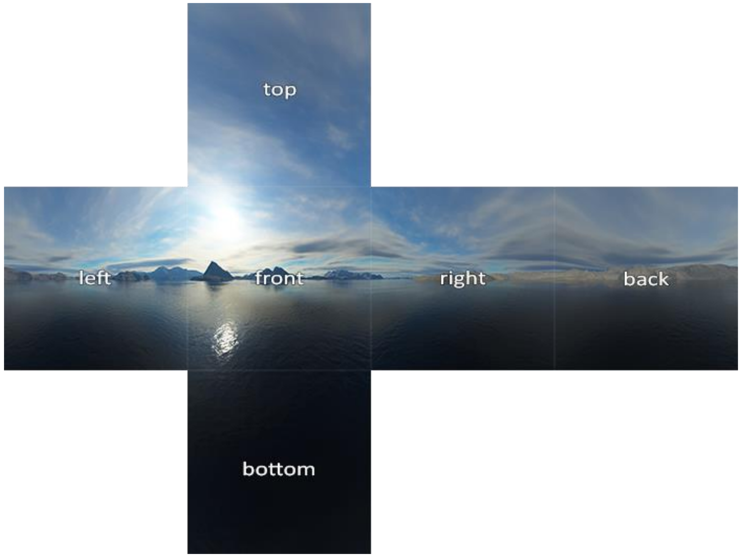


- Two approaches
  - Cube + **cube map** texture
  - Sphere + **longitude-latitude** image

Skybox (cont.)

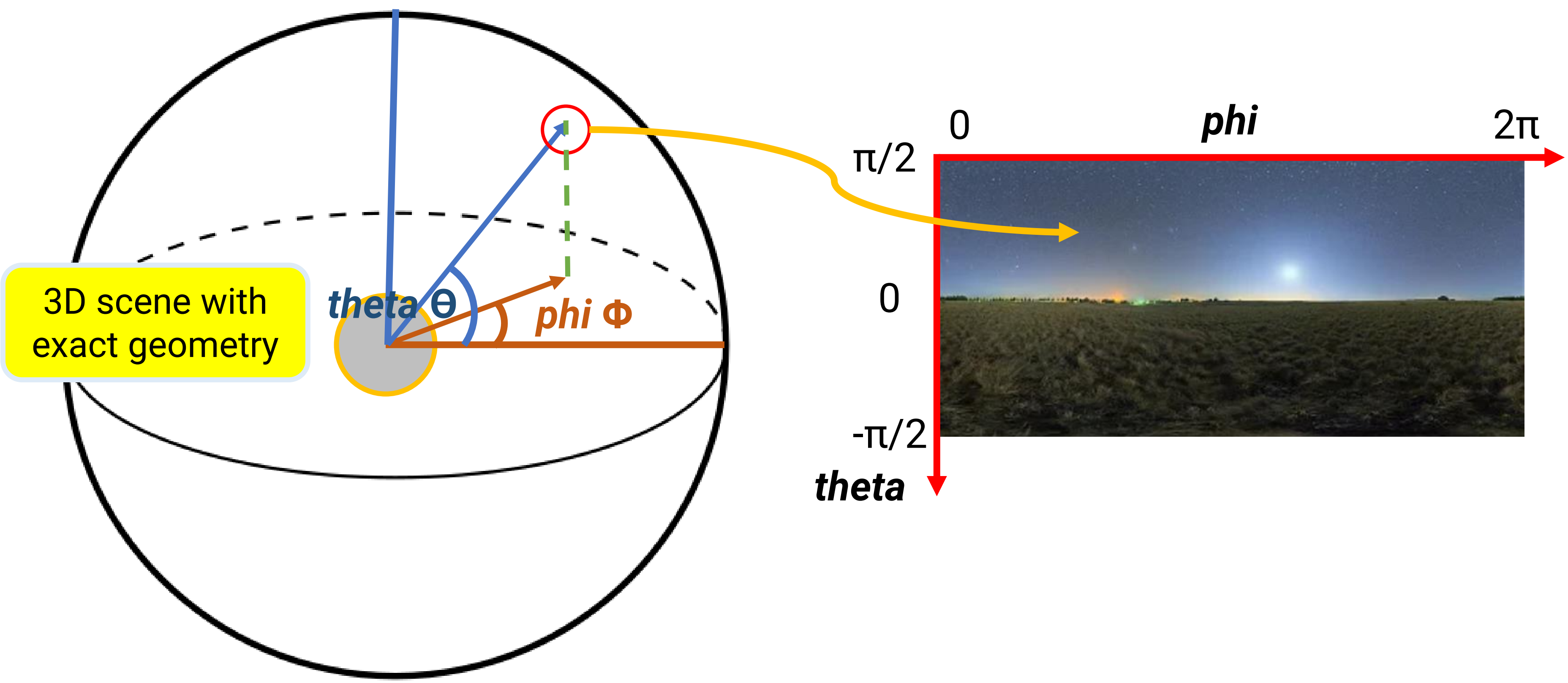
- Cube + **cube map** texture
  - Centered at world-space origin, with a significant long extent





Skybox (cont.)

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



Reflection of the Skybox

- When rendering the scene, compute a reflected direction based on the viewing direction
- Use the reflected direction to lookup the skybox texture and obtain the reflected contribution
- Add the reflected contribution to the surface color

Reflection (cont.)

Ray Traced

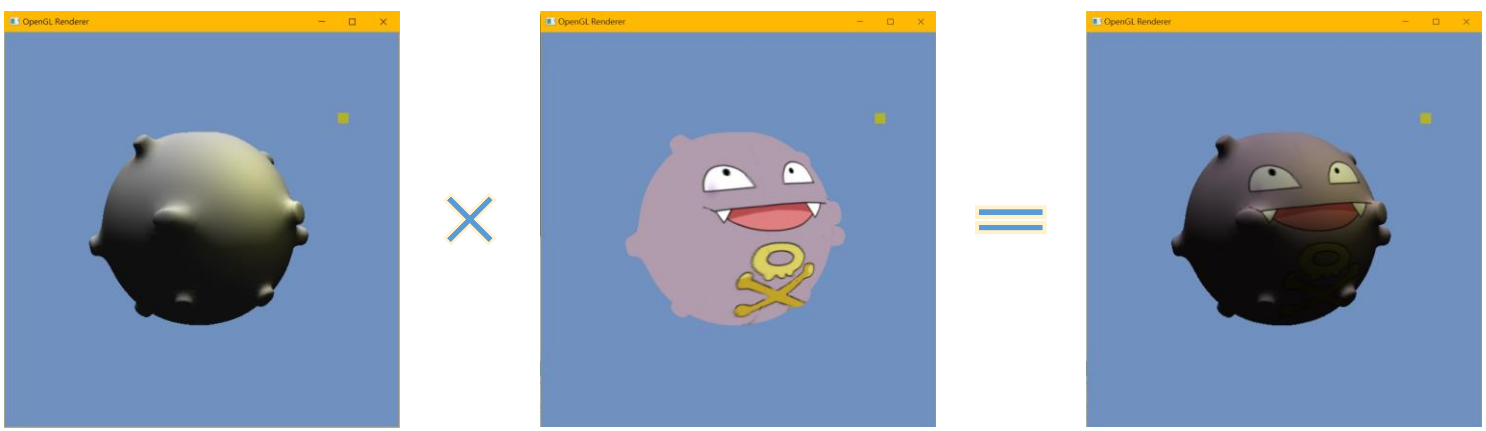
Environment Map

Outline

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

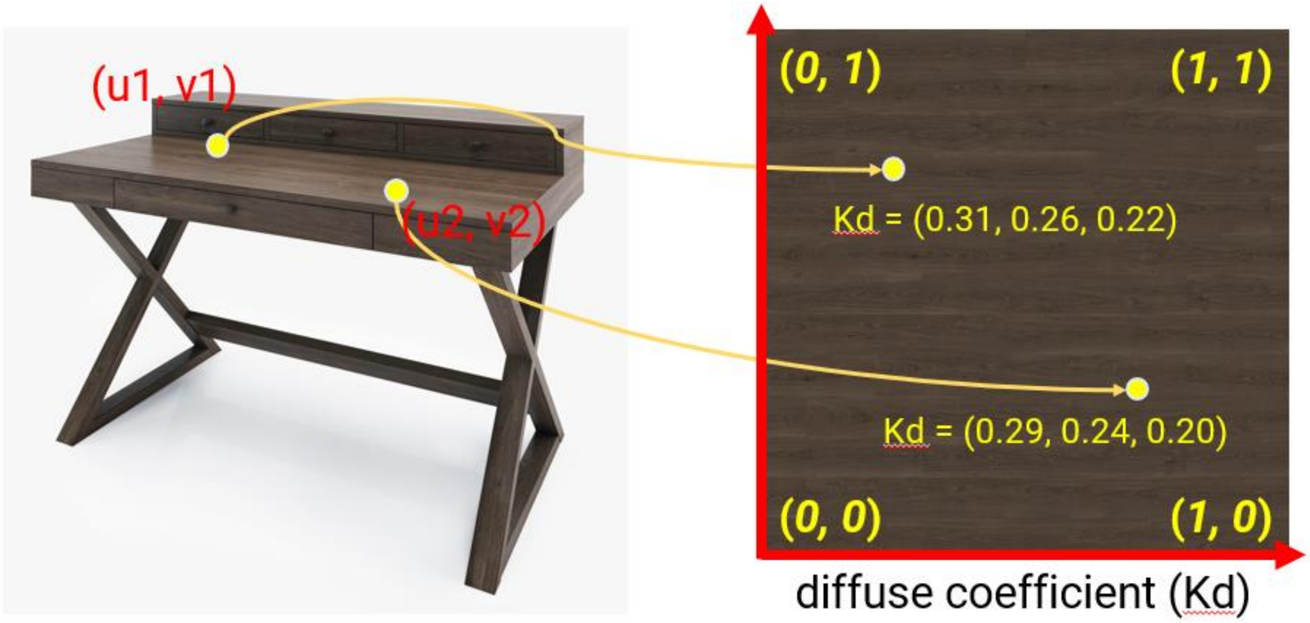
Overview

- 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.)

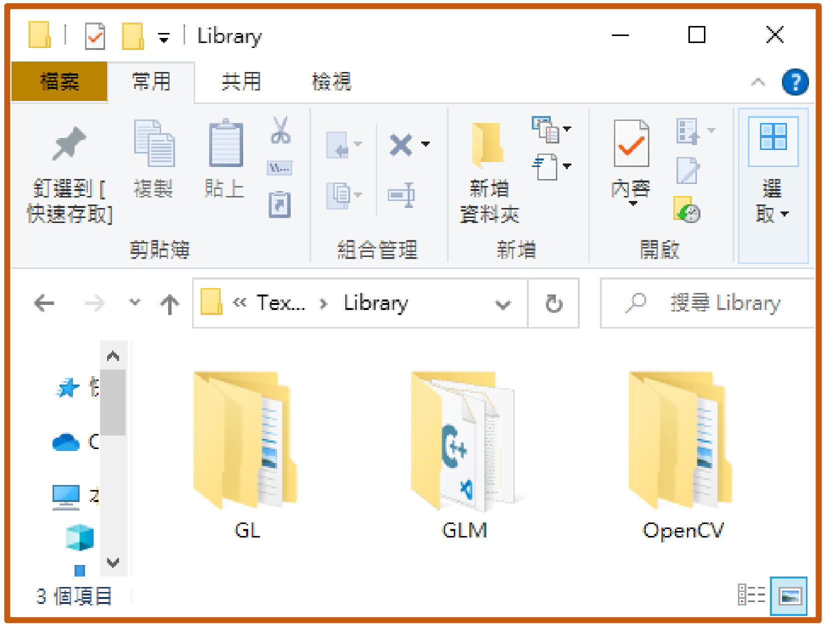
- In OpenGL 2.0 and after, the correct way to handle texture is to use the texture color as diffuse coefficients (Kd)

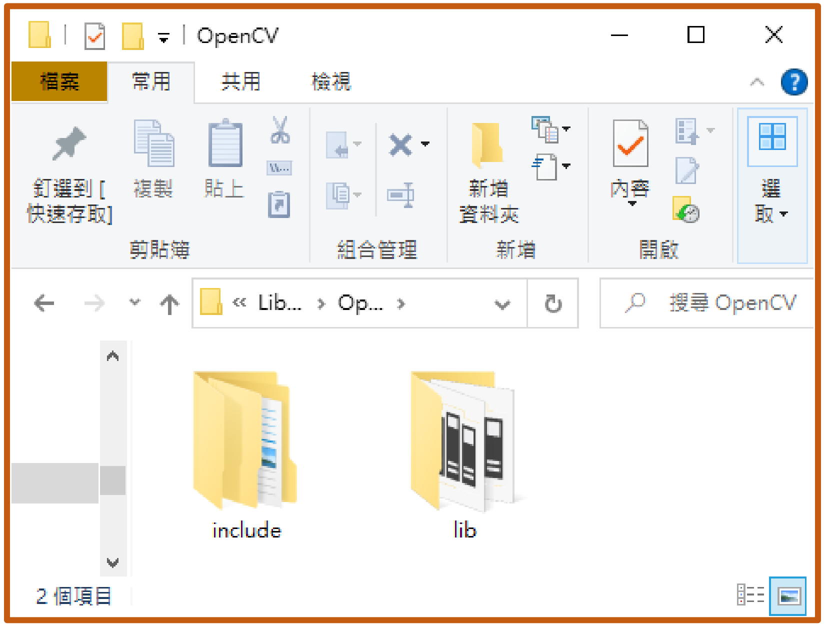


- This needs **per-fragment lighting**, which is part of your HW2/HW3

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

```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)

```cpp
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.)

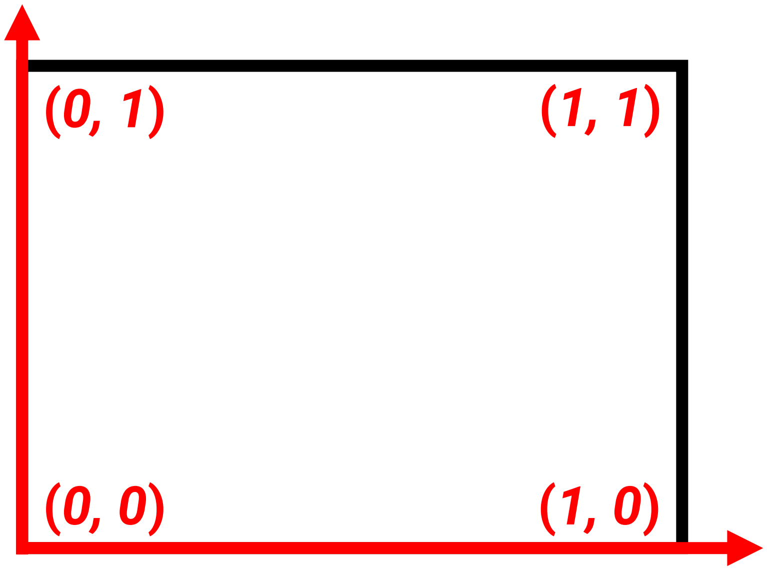
```
ImageTexture::ImageTexture(const std::string filePath)
    : texFileName(filePath)
{
    imageWidth = 0;
    imageHeight = 0;
    numChannels = 0;
    textureObj = 0;

    // 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)

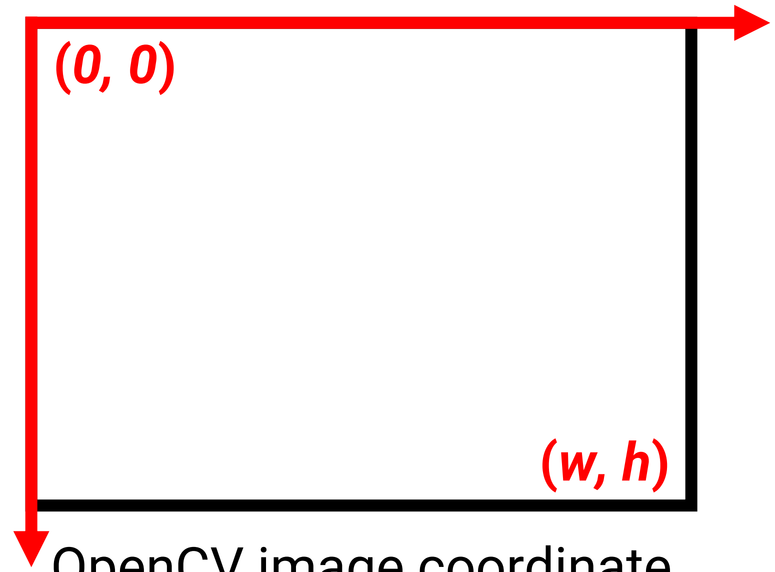
3 for RGB images
4 for RGBA images

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, — GL_TEXTURE_2D, GL_TEXTURE_CUBE_MAP_POSITIVE_X, … etc.
GLint level, — level of details, usually set to 0
GLint internalformat, — the internal format of the **texture** GL_RED, GL_RG, GL_RGB, GL_RGBA, GL_DEPTH_COMPONENT … etc.
GLsizei width,
GLsizei height,
GLint border, — must be 0
GLenum format, — the format of the **image data** GL_RED, GL_RG, GL_RGB, GL_RGBA … etc.
GLenum type, — the data type of the **pixel data** GL_UNSIGNED_BYTE, GL_FLOAT … etc.
const void * data — 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
);
```

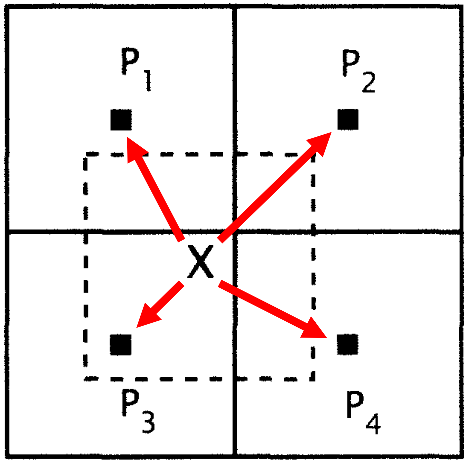
pname: Specifies the symbolic name of a single-valued texture parameter, such as GL_TEXTURE_MIN_FILTER GL_TEXTURE_MAG_FILTER GL_TEXTURE_WRAP_S (T) … etc.

param: 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);
```

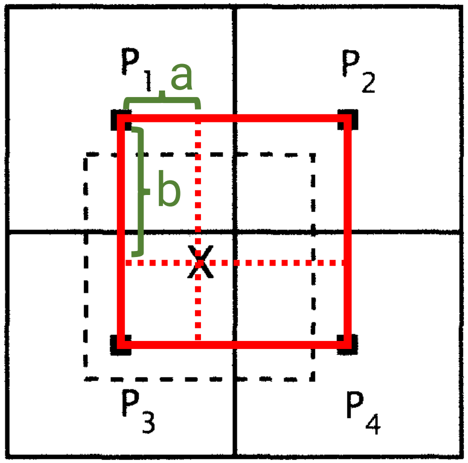
Recap: Texture Filtering

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



nearest neighbor

P_3 is closest
Use P_3's pixel value

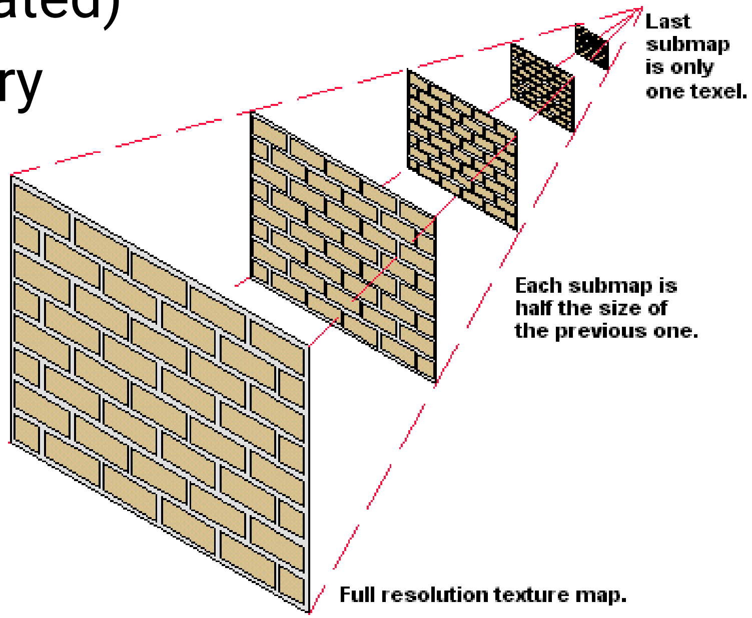


bilinear interpolation

$$(1\text{-}a)(1\text{-}b)P_1 + (a)(1\text{-}b)P_2 + (1\text{-}a)(b)P_3 + (a)(b)P_4$$

Recap: Mipmap

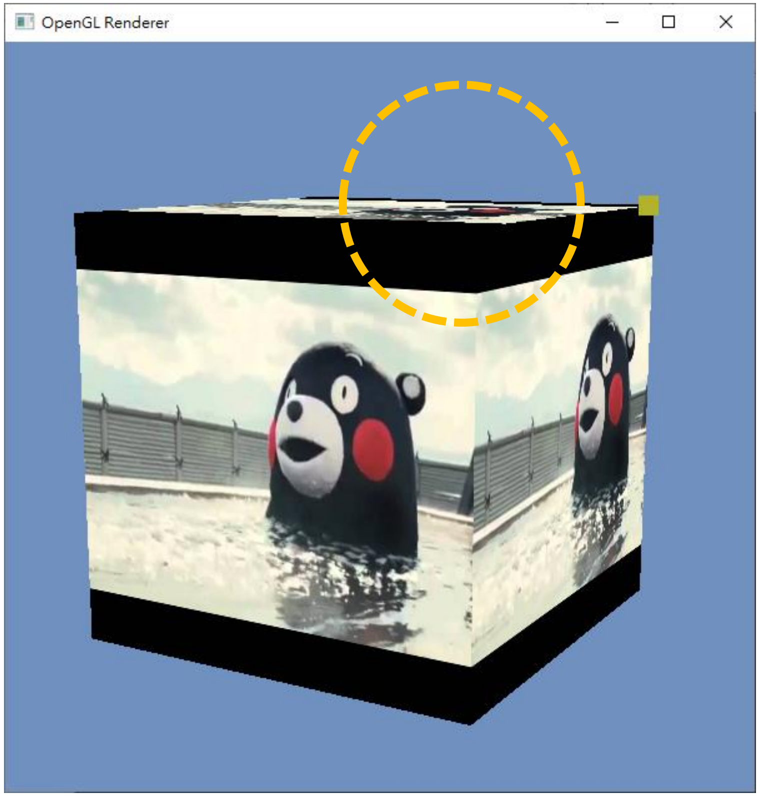
- Mipmap provides a clever way to solve this problem
- **Pre-process**
  - Build a **hierarchical representation** of the texture image
  - Each level has a half resolution of its previous level (generated by linearly interpolated)
  - Take at most **1/3** more memory



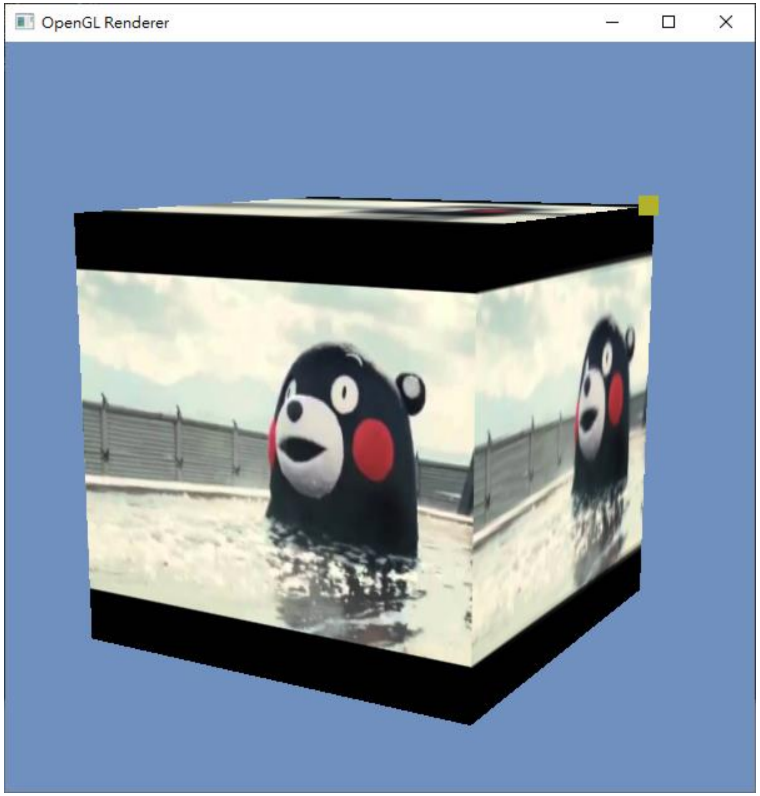

```
glGenerateMipmap(GL_TEXTURE_2D);
```

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);
}
```

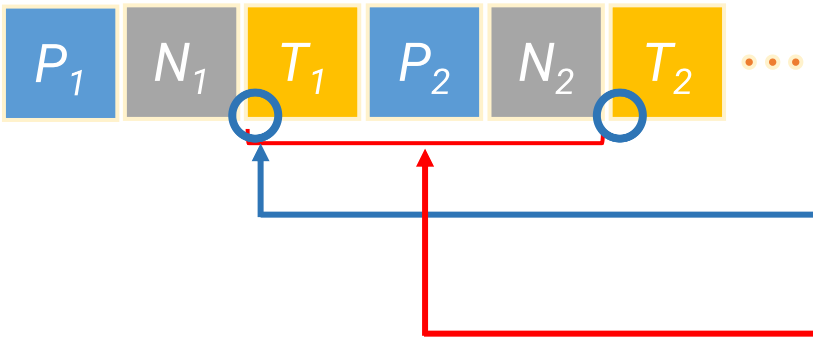
interpolated texture coordinate

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

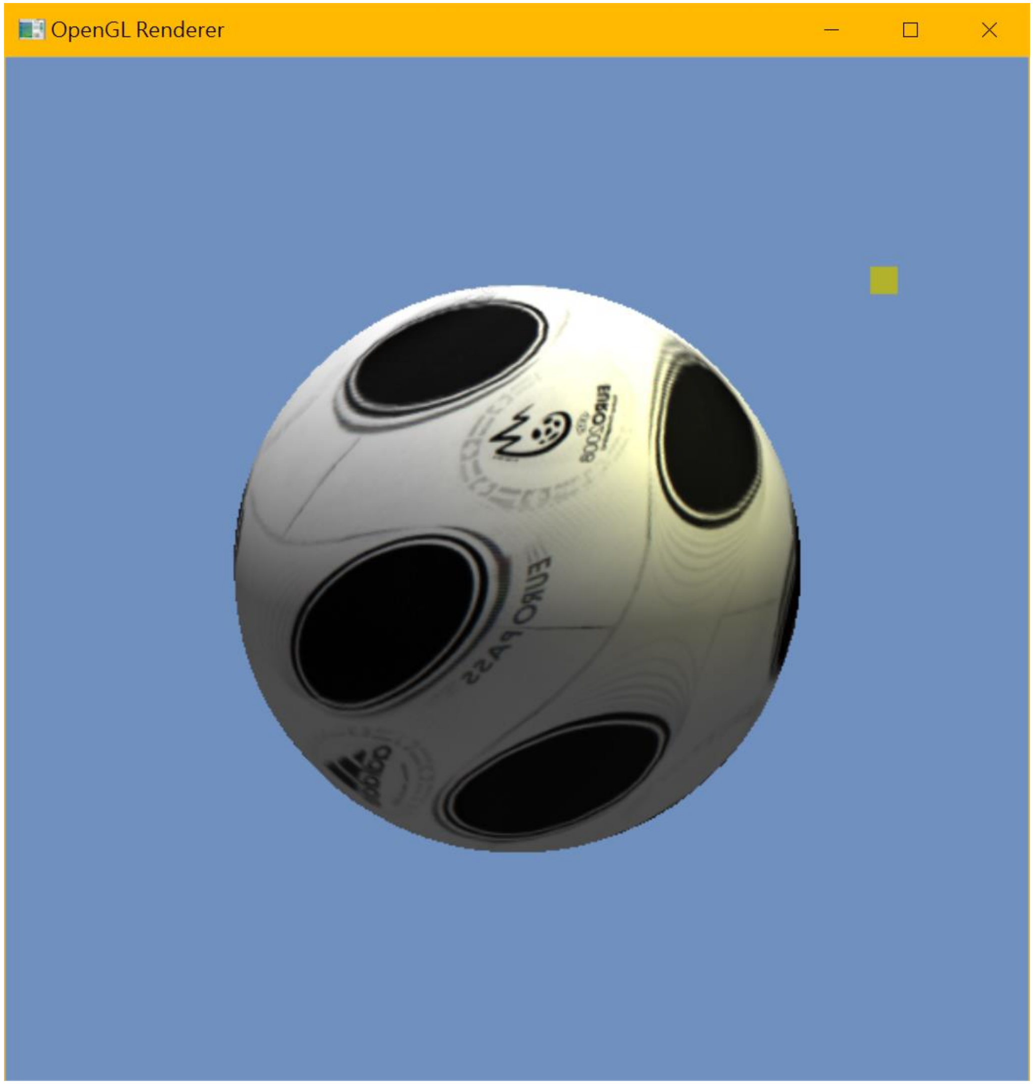
```cpp
void RenderSceneCB()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // 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->GetLocK
        glUniform3fv(gouraudShadingShader->GetLocK
        glUniform3fv(gouraudShadingShader->GetLocK
        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();
    }
```

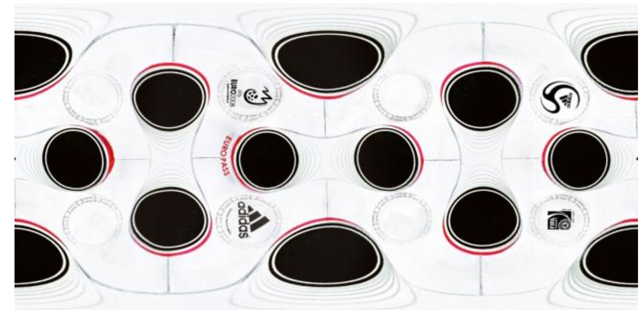
```cpp
void ImageTexture::Bind(GLenum textureUnit)
{
    glActiveTexture(textureUnit);
    glBindTexture(GL_TEXTURE_2D, textureObj);
}
```

the n^{th} texture in the shader

```cpp
// 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?