Texture Mapping

527970
Fall 2020
11/19/2020
Kyoung Shin Park
Computer Engineering
Dankook University

OpenGL Texturing

- Three steps to applying a texture mapping in OpenGL
  - Enable texture mapping

    ```
    glEnable(GL_TEXTURE_2D)
    ```

  - Specify texture parameters (wrapping, filter modes)

    ```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
       GL_UNSIGNED_BYTE, imageData)
    ```

  - Assign texture coordinates to vertices

    ```
    // Square Vertices Texture Coordinates
    squareTextureCoords.push_back(glm::vec2(0.0f, 0.0f));
    squareTextureCoords.push_back(glm::vec2(1.0f, 0.0f));
    squareTextureCoords.push_back(glm::vec2(1.0f, 1.0f));
    squareTextureCoords.push_back(glm::vec2(0.0f, 1.0f));
    ```

OpenGL Texturing

- In Modern OpenGL, texture mapping is specified as sampler in fragment shading
  - Sampler gives the texture color

```
//TextureFragmentShader.fs
in vec2 texCoordPass; // texure coordinate from rasterizer
out vec4 Color;  // output color
uniform sampler2D gTextureSampler; //texture object from application
void main()  {
    Color = texture2D( gTextureSampler, texCoordPass );
}
//TransformVertexShader.vs
in vec3 vPosition;// vertex position (in model space)
in vec2 vTexCoord;// texture coordinate (in model space)
out vec2 texCoordPass; // Output data will be interpolated for each fragment.
uniform mat4 gMVP; // Values that stay constant for the whole mesh.
void main() {
    gl_Position =  gMVP * vec4(vPosition,1); // Output position of the vertex
    texCoordPass = vTexCoord;
}
```

OpenGL Texture Names

- Set texture with textureID

```
GLuint textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_2D, textureID);
glTexImage2D(...);
glTexParameterf(.....);
glBindTexture(GL_TEXTURE_2D, 0);
...
```

- Activate texture mapping with textureID

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID);
```

glTexImage2D

- glTexImage2D(GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texels);
  - target: texture type, e.g., GL_TEXTURE_2D
  - level: mipmapping level
  - components: texel components, e.g., RGB, RGBA
  - width, height: texel width & height (in pixels) must be power of 2
  - border: used for smoothing (generally 0)
  - format: texel format
  - type: texel type
  - texels: texels pointer

glTexImage1D

- glTexImage1D(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLint border, GLenum format, GLenum type, const GLvoid *pixels);
- The glTexImage2D() function defines a 2D texture image
- The glTexImage1D() function defines a 1D texture image.
- The usage of the glTexImage1D() and glTexImage2D() functions and the meaning of the arguments are almost the same, and the height parameter of the image texture is added only to the glTexImage2D() function.

glTexSubImage2D

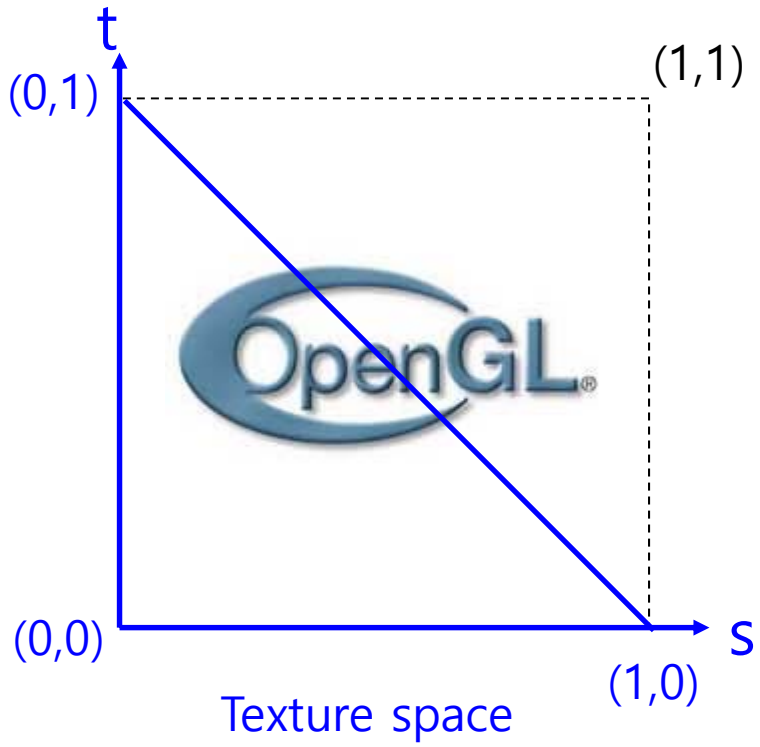
- glTexSubImage2D function is used to read portion of the texture image when the texture size is not a power of 2.
- glTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels);

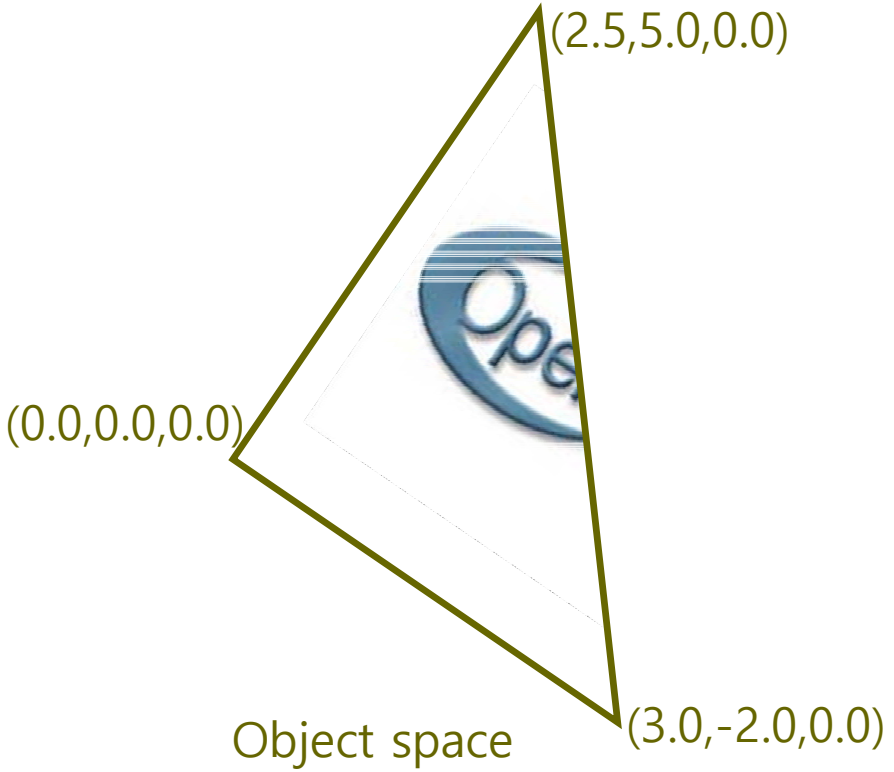
OpenGL Texture Coordinates

- For texture mapping, you need to define texture coordinates to the vertices of object.
- The GLUT teapot contains texture coordinates.
- The GLU quadrics also contains texture coordinates defined as an option
  - Enable texture mapping using gluQuadricTexture(quadric, GL_TRUE).

OpenGL Texture Coordinates

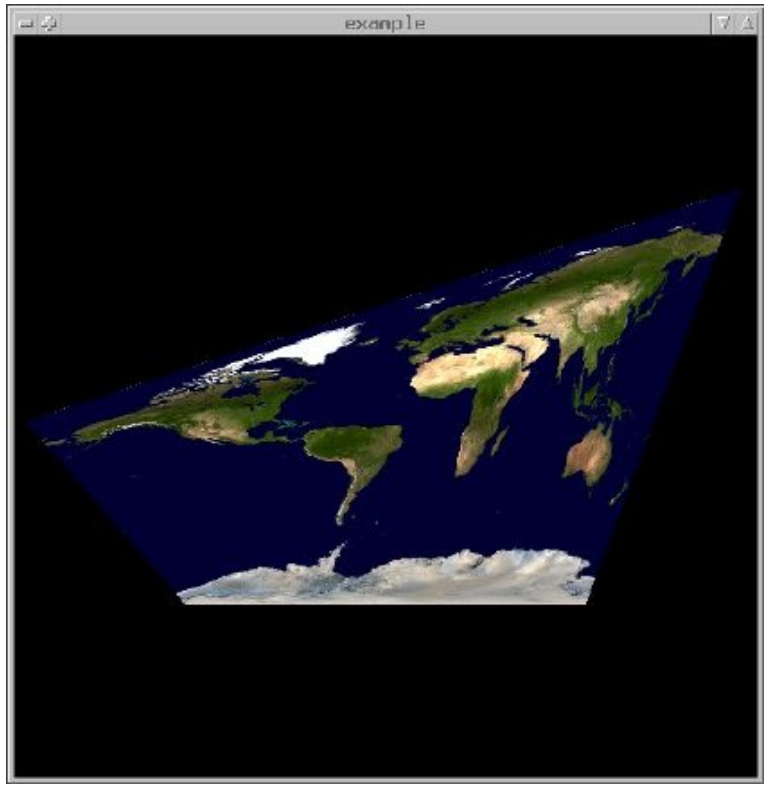
- In OpenGL, texture coordinates are formed from 0 to 1 in (S, T) of the texture image.
- Texture coordinates are interpolated on the object surface.





OpenGL Texture Coordinates

- The texture does not need to be evenly applied to the polygon.
- Depending on the user of geometric models or texture coordinates, the image may sometime appear distorted.



Texture Coordinates

(0,0)
(1,0)
(1,1)
(0,1)

Vertex Positions

(-1.0, -1.0, 0.0)
(1.0, -1.0, 0.0)
(rMousePosX, rMousePosY, 0.0)
(lMousePosX, lMousePosY, 0.0)

OpenGL Texture Filtering

- In OpenGL, when texture coordinates are out of [0, 1], it is defined as a texture wrapping:
  - **GL_CLAMP, GL_CLAMP_TO_EDGE, GL_REPEAT, GL_MIRRORED_REPEAT**
- Following example defines 4 texture coordinates in a rectangle as (0,0), (0,3.5), (3.5,0), (3.5,3.5).

Repeat

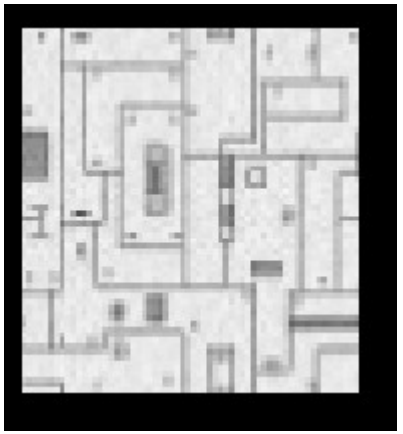
Clamp

Repeat &Clamp

OpenGL Texture Filtering

- Mipmap is half the width and height of the previous mipmap texture.
  - The smaller the texture, the more texels must be applied to one pixel, so the GL_NEAREST or GL_LINEAR filter may not produce accurate calculation results. Therefore, texture may appear flickering when the object moves.
  - Mipmaps can reduce this flickering problem. However, it is generally blurred.

GL_LINEAR



GL_LINEAR_MIPMAP_LINEAR

OpenGL Texture Filtering

- glTexParameter{if}v(GLenum target, GLenum pname, TYPE *param);
  - GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T
    - GL_CLAMP, GL_CLAMP_TO_EDGE, GL_REPEAT, GL_MIRRORED_REPEAT
  - GL_TEXTURE_MAG_FILTER
    - GL_NEAREST, GL_LINEAR
  - GL_TEXTURE_MIN_FILTER
    - GL_NEAREST, GL_LINEAR (When Mipmap is not used)
    - GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_NEAREST
  - GL_TEXTURE_BORDER_COLOR
    - [0.0, 1.0]
  - GL_TEXTURE_PRIORITY
    - 0 or 1

Texture Environment Parameters

- glTexEnv{fi}[v](..) set how to blend the texture color (C_t, A_t) and the framebuffer color (C_f, A_f).
  - GL_TEXTURE_ENV_MODE:
    - GL_MODULATE: modulate the lighting color with texture color
    - GL_DECAL: the pixel color is determined by texture color
    - GL_BLEND: combine with environment color
    - GL_REPLACE: use only texture color
- GL_BLEND color is specified as GL_TEXTURE_ENV_COLOR

```
GLfloat blendcolor[] = {0.0, 1.0, 0.0, 0.5};
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR,
    blendcolor);
```

Texture Environment Parameters

int. format	GL_REPLACE	GL_MODULATE
GL_ALPHA	$C = C_f,\ A = A_t$	$C = C_f,\ A = A_f A_t$
GL_LUMINANCE	$C = L_t,\ A = A_f$	$C = C_f L_t,\ A = A_f$
GL_LUMINANCE_ALPHA	$C = L_t,\ A = A_t$	$C = C_f L_t,\ A = A_f A_t$
GL_INTENSITY	$C = I_t,\ A = I_t$	$C = C_f I_t,\ A = A_f I_t$
GL_RGB	$C = C_t,\ A = A_f$	$C = C_f C_t,\ A = A_f$
GL_RGBA	$C = C_t,\ A = A_t$	$C = C_f C_t,\ A = A_f A_t$

int. format	GL_DECAL	GL_BLEND
GL_ALPHA	undefined	$C = C_f,\ A = A_f A_t$
GL_LUMINANCE	undefined	$C = C_f(1 - L_t) + C_c L_t,\ A = A_f$
GL_LUMINANCE_ALPHA	undefined	$C = C_f(1 - L_t) + C_c L_t,\ A = A_f A_t$
GL_INTENSITY	undefined	$C = C_f(1 - I_t) + C_c I_t,\ A = A_f(1 - I_t) + A_c I_t$
GL_RGB	$C = C_t,\ A = A_f$	$C = C_f(1 - C_t) + C_c C_t,\ A = A_f$
GL_RGBA	$C = C_f(1 - A_t) + C_t A_t,\ A = A_f$	$C = C_f(1 - C_t) + C_c C_t,\ A = A_f A_t$

OpenGL Texture Movies

- To create flipbook animation using texture image sequence
  - initTexture() function reads the entire texture images.
  - idle() function updates currentTextureID.
  - drawTexture() function binds one texture per frame using glBindTexture(GL_TEXTURE_2D, **currentTextureID**) to the same vertex coordinates and texture coordinates – giving animation effects.

image1 image2 image3 image4 image5 image6

OpenGL Texture Coordinate Generation

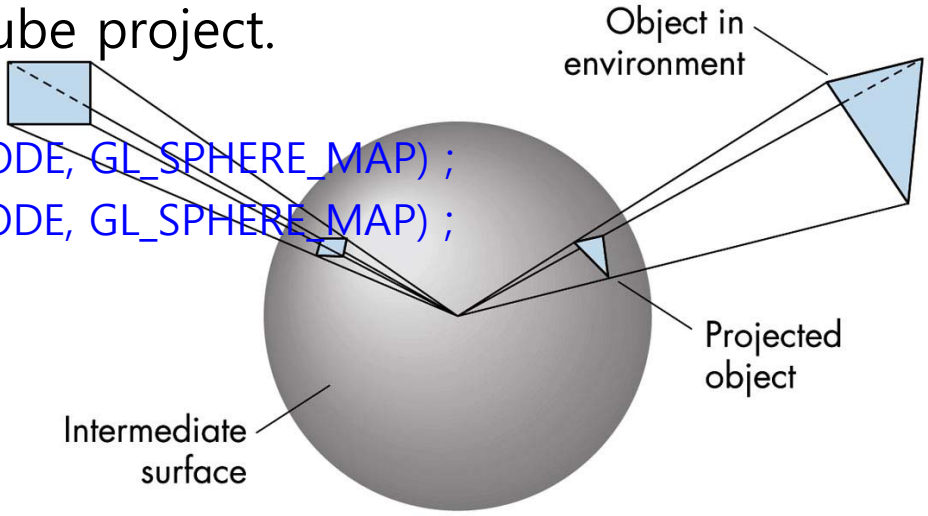
- Texture coordinates can be automatically generated.
  - Automatic generation of texture coordinates in S, T must be enabled.
    - glEnable(GL_TEXTURE_GEN_S), glEnable(GL_TEXTURE_GEN_T)
  - GL_TEXTURE_GEN_MODE 모드:
    - GL_OBJECT_LINEAR, GL_EYE_LINEAR, GL_SPHERE_MAP
  - Specify the plane – to create texture coordinates based on the distance from the plane.
    - glTexGenfv(GL_S, GL_OBJECT_PLANE, planeCoefficients)

```
planeCoefficients = { 1, 0, 0, 0 };
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR) ;
glTexGenfv(GL_S, GL_OBJECT_PLANE, planeCoefficients);
glEnable(GL_TEXTURE_GEN_S) ;
glBegin(GL_QUADS) ;
glVertex3f(-3.25, -1, 0); glVertex3f(-1.25, -1, 0) ;
glVertex3f(-1.25, 1, 0); glVertex3f(-3.25, 1, 0) ;
glEnd() ;
```

OpenGL Sphere Mapping

- OpenGL sphere mapping
  - The spherical mapping texture coordinates are calculated as the reflection vector reflected by the view vector to the normal vector of the spherical surface.
  - It is simple to map the reflection vector to 2D texture coordinates and can be implemented in hardware or software.
  - However, it is difficult to obtain a spherical image(it must be an image containing the surrounding environment of 360 degrees). Perspective projection by a very wide optical lens, or approximation by using a cube project.

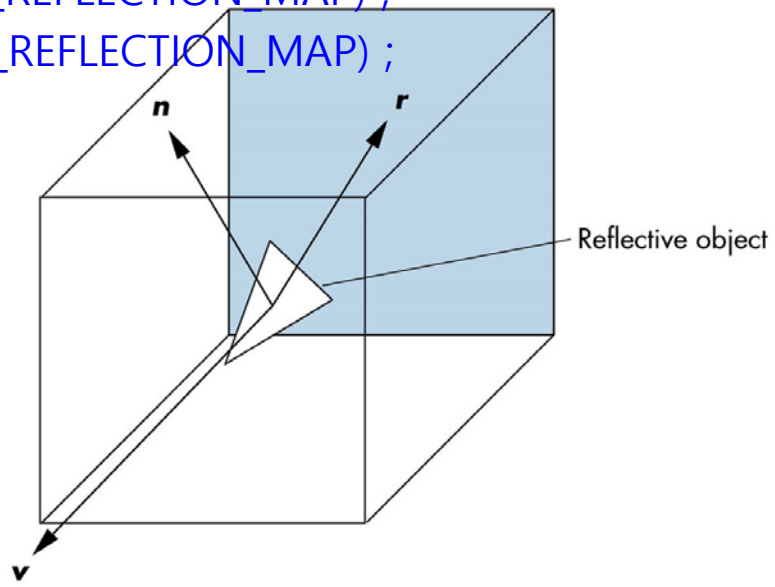
```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP) ;
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP) ;
glEnable(GL_TEXTURE_GEN_S) ;
glEnable(GL_TEXTURE_GEN_T) ;
```



OpenGL Box Mapping

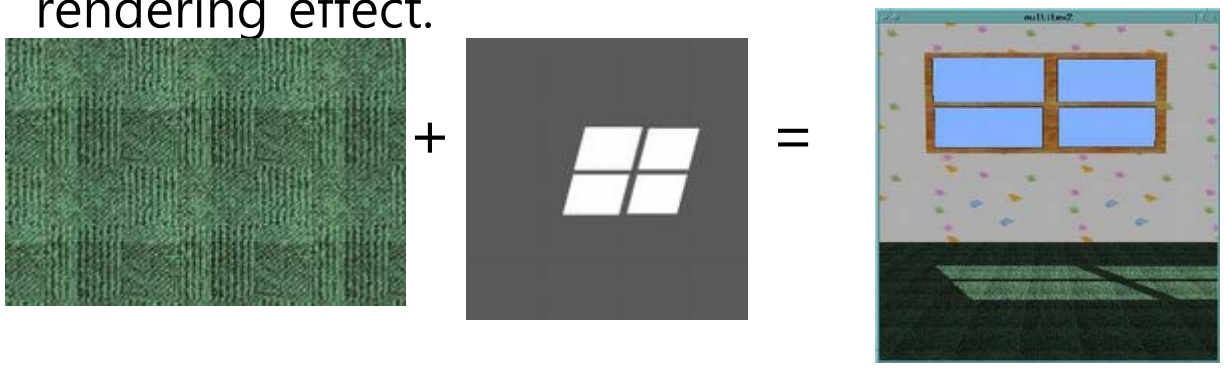
- OpenGL box mapping
  - The cube map is one of reflection mapping.
  - However, the cube map must be 3D texture coordinates.
  - The reflective texture is a 6-sided two-dimensional texture of a cube surrounding the environment.

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP) ;
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP) ;
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP) ;
glEnable(GL_TEXTURE_GEN_S) ;
glEnable(GL_TEXTURE_GEN_T) ;
glEnable(GL_TEXTURE_GEN_R) ;
glEnable(GL_TEXTURE_CUBE_MAP);
```

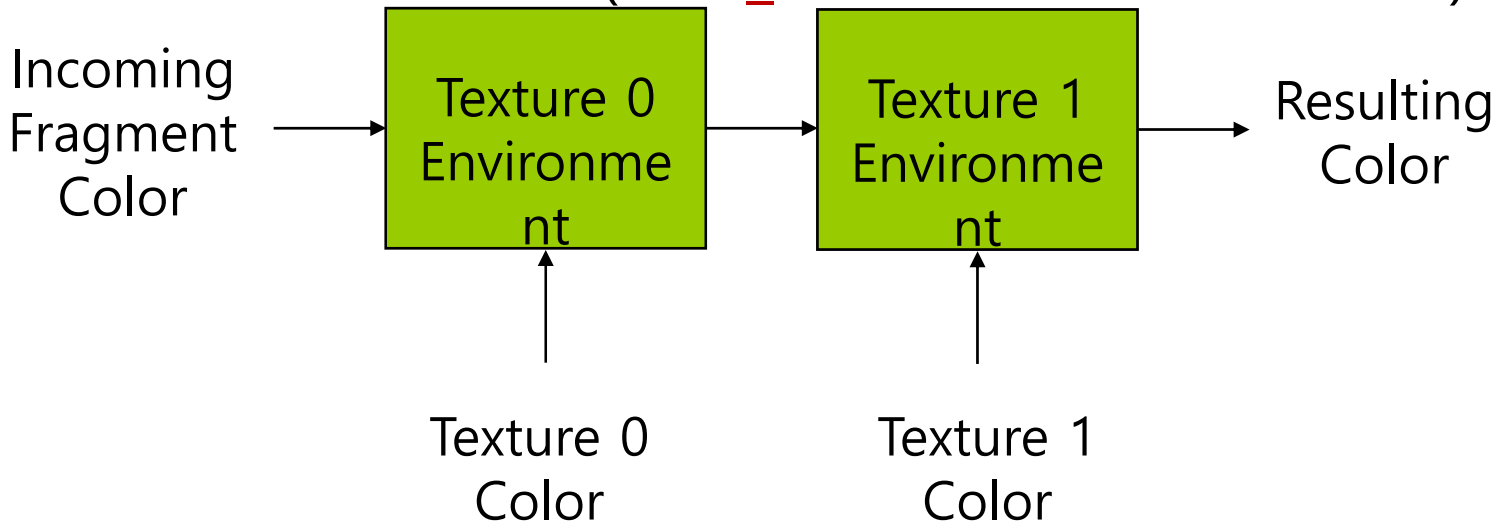


Multitexturing

- Multitexturing
  - Apply more than one texture to an object to enhance the rendering effect.

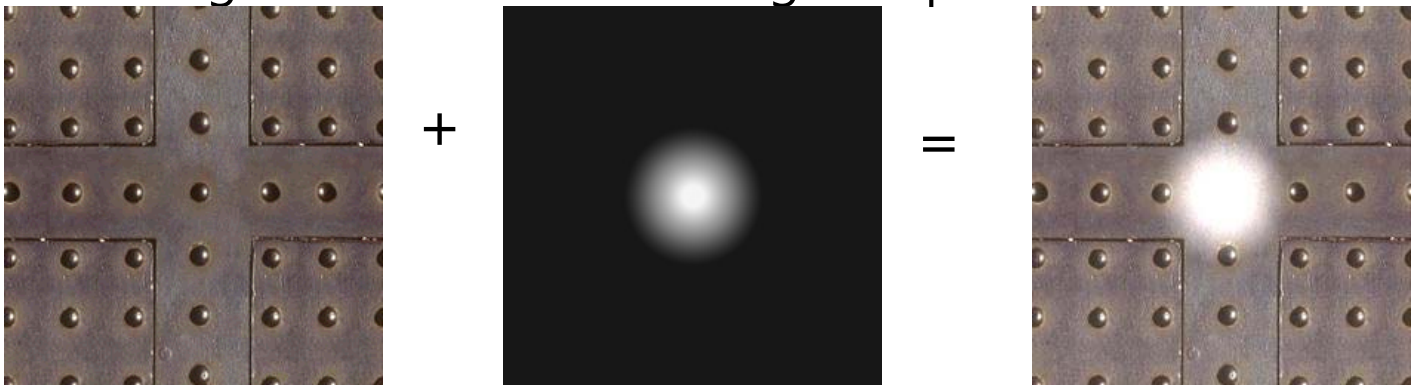


- OpenGL 1.2.1 revision (**ARB_multitexture** extension)

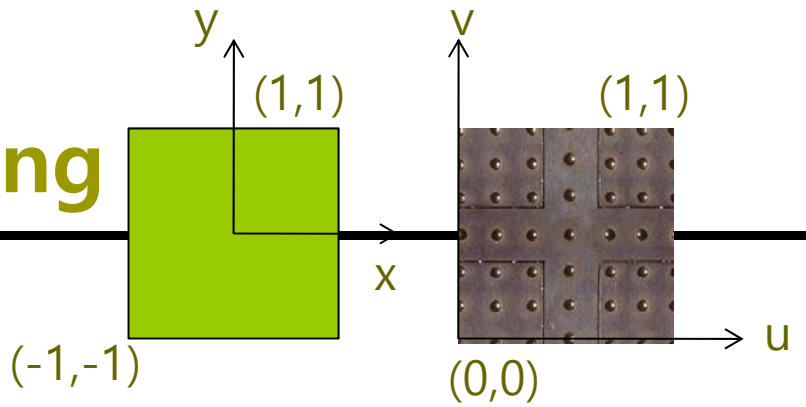


Multitexturing

- Single-Pass vs. Multi-Pass Multitexturing
  - Single-pass multitexturing means applying multiple textures within one rendering pass.
  - Multi-pass multitexturing is the rendering of the scene or the polygon itself multiple times by blending.
- Light Mapping
  - Instead of calculating the lighting of the object surface, the resulting image is directly applied to the object surface by blending normal texture and lightmap texture.



Single-Pass Multitexturing

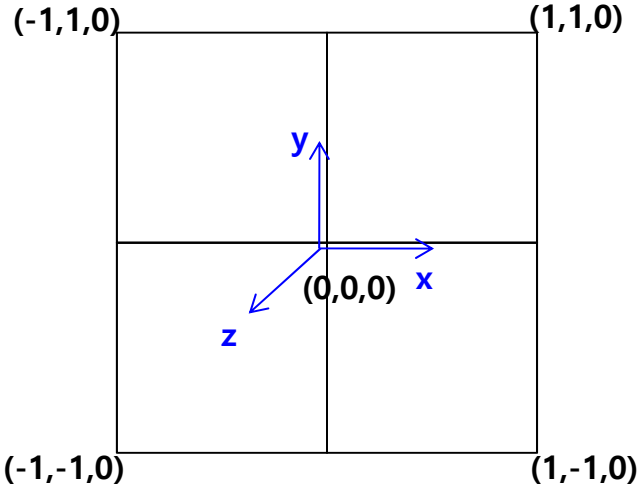



```
void SetMultitexturSquareData() { // ..
glGenBuffers(4, &vbo[0]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec3), &quadVertices[0], GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec3), &quadNormals[0], GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec2), &quadTextureCoords[0], GL_STATIC_DRAW
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, vbo[3]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec2), &quadTextureCoords[0], GL_STATIC_DRAW
glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(3);
}
```

Single-Pass Multitexturing

- Bind and enable two 2D multitextures to draw a quad

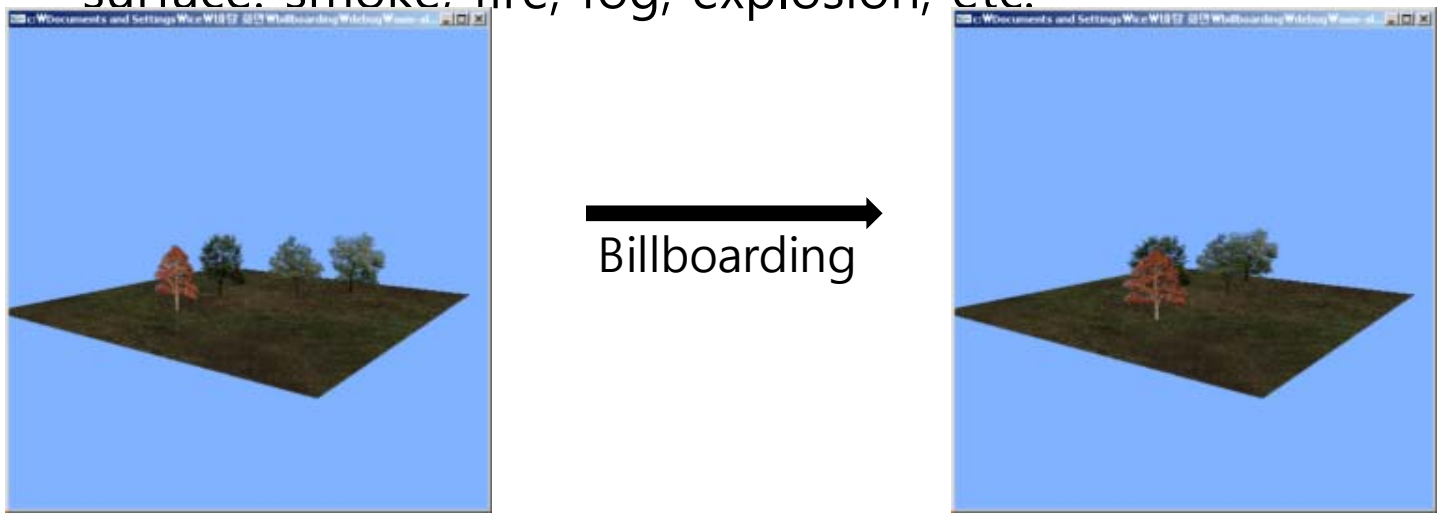
```
// stage 0 activate
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture0);
// stage 1 activate
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture1);

drawSquare();
// texture disabled
glBindTexture(GL_TEXTURE_2D, 0);
```


(-1,1,0)
(1,1,0)
y
x
(0,0,0)
z
(-1,-1,0)
(1,-1,0)


OpenGL

Billboarding

- Billboard technique
  - The front of billboard rectangle is made to always look toward the camera, and as a result, the billboard always shows the same side no matter which direction the camera is viewed.
  - For example, tree billboard images are used to create a forest, instead of using tree mesh models.
  - The billboard technique combined with the alpha texture is used to express various natural phenomena that do not have a solid surface: smoke, fire, fog, explosion, etc.



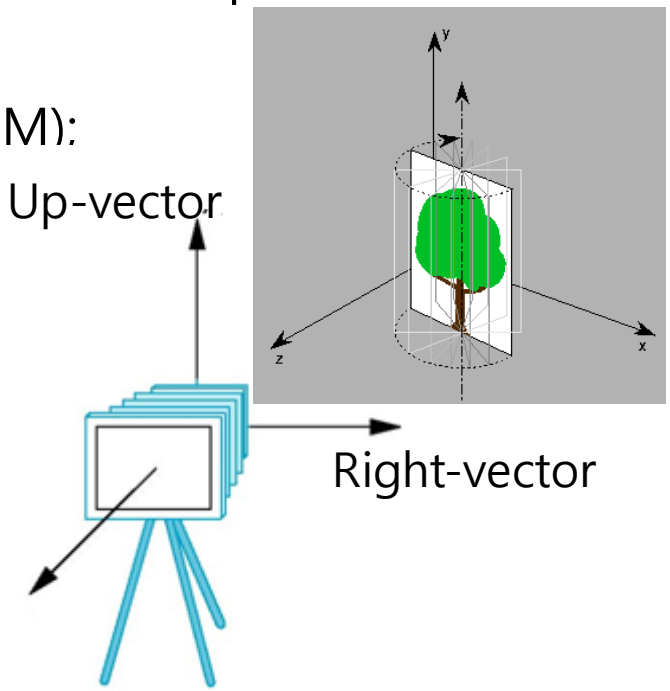
Billboarding

- Billboarding principle
  - The key to implementation is to adjust the vertices that make up the billboard square using the Modelview matrix so that the user always look at the viewpoint.
  - The Modelview matrix contains information about the up vector and the right vector of the viewer's viewpoint.

```
Glfloat M[16];
glGetFloatv(GL_MODELVIEW_MATRIX, M);
```

Right-vector Up-vector Look-vector

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$



Billboarding

- Axial Symmetry
  - Billboard rectangle should rotate around the vertical axis.
  - Calculate the camera's yaw angle from the Modelview matrix, M.

```
theta = atan2f(M[8], M[10]);
```

Look.x Look.z

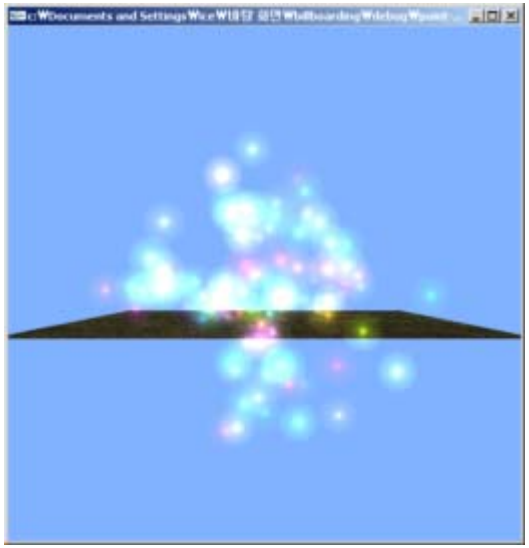
  - The rotation matrix, R, of the billboard rectangle is calculated as an arbitrary axis (typically, up vector=(0, 1, 0)) and angle (inverse of the camera yaw angle).

$$R = \mathrm{I}\cos\theta + \mathbf{Symmetric}\ (1-\cos\theta) + \mathbf{Skew}\ \sin\theta$$

$$= \begin{bmatrix} a_x^2 + \cos\theta(1-a_x^2) & a_x a_y(1-\cos\theta) - a_z\sin\theta & a_x a_z(1-\cos\theta) + a_y\sin\theta \\ a_x a_y(1-\cos\theta) + a_z\sin\theta & a_y^2 + \cos\theta(1-a_y^2) & a_y a_z(1-\cos\theta) - a_x\sin\theta \\ a_x a_z(1-\cos\theta) - a_y\sin\theta & a_y a_z(1-\cos\theta) + a_x\sin\theta & a_z^2 + \cos\theta(1-a_z^2) \end{bmatrix}$$

Point Sprites

- Point Sprites is a hardware-based billboard technique, and used for particle system (using points instead of a textured quadrangle mesh).
- OpenGL extension (**GL_ARB_point_parameters** & **GL_ARB_point_sprite**)



Reference

- OpenGL Billboarding Tutorial
  http://www.lighthouse3d.com/opengl/billboarding
- SIGGRAPH'97 Advanced OpenGL Programs
  http://www.opengl.org/resources/code/samples/advanced/advanced97/programs/programs.html
- OpenGL Point Sprites
  http://www.informit.com/articles/article.aspx?p=770639&seqNum=7
- Particle Systems http://www.gamedev.net/