

Texture Mapping

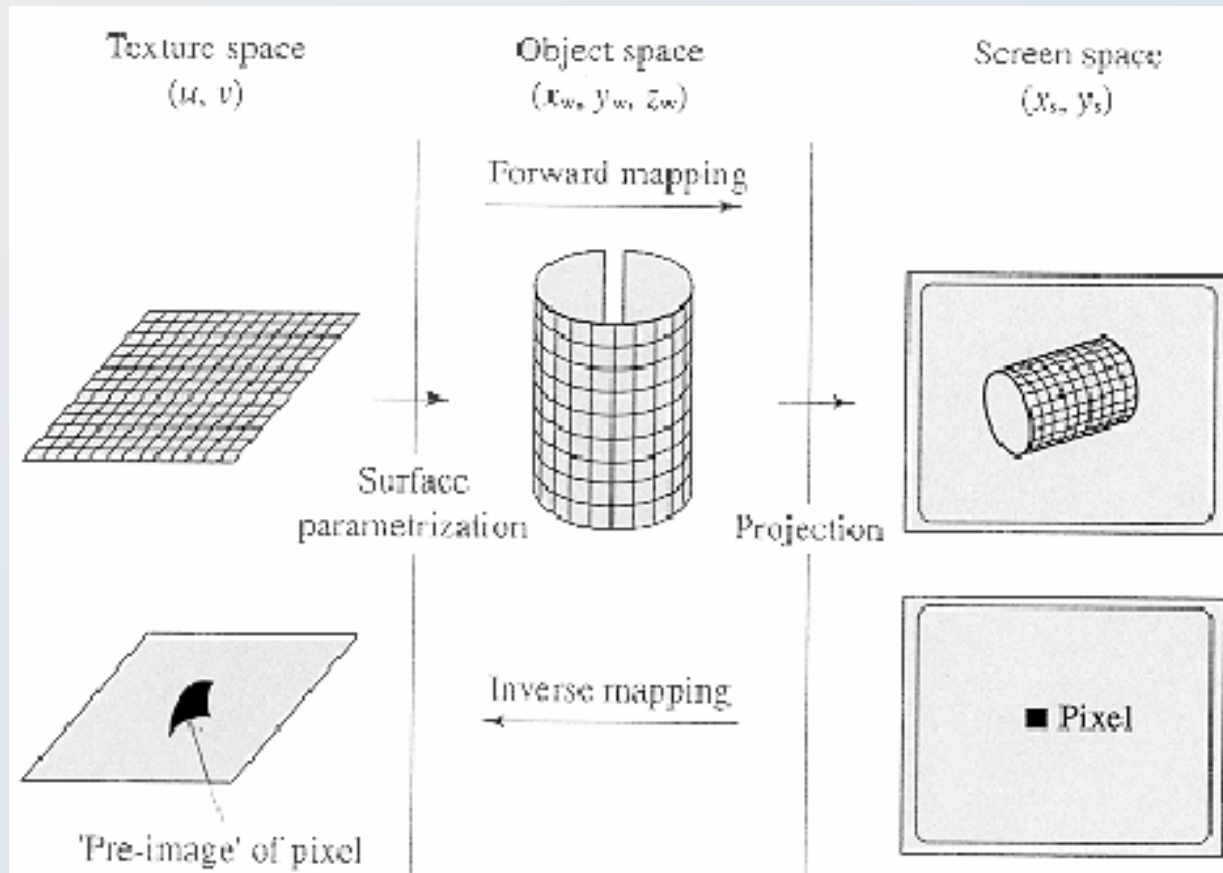
Tan-Chi Ho

**CGGM Lab.
Dep. Computer Science
National Chiao Tung University**

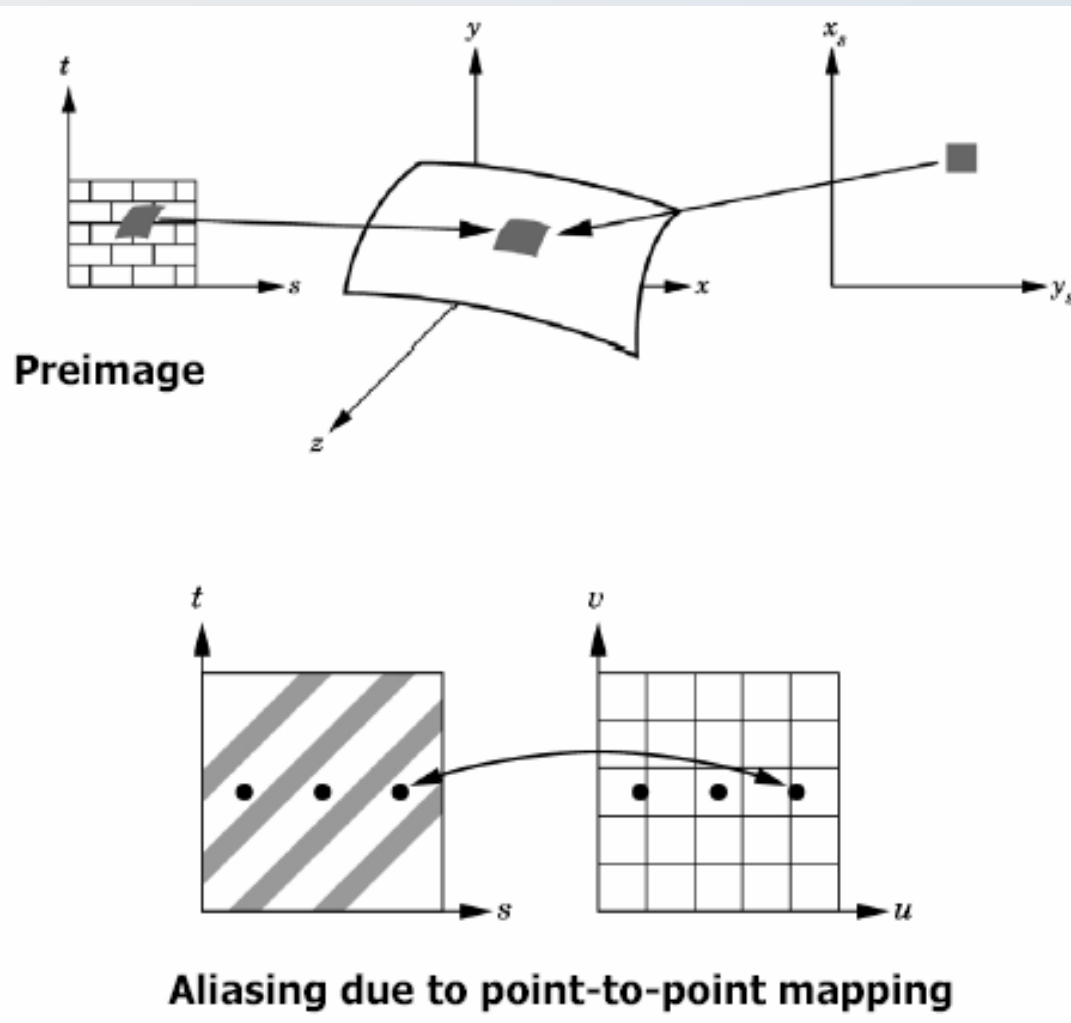
Introduction

- Textures are simply rectangular arrays of data.
 - Ex. Color, luminance, alpha, normal...
 - The individual values in a texture array are called texels.
- In OpenGL, the length and width of the texture array must be **power of 2**.
 - Non-power of 2 texture is allowed in newer spec.

Forward Mapping



Backward Mapping



Steps in Texture Mapping

- Specify a texture.
- Indicate how the texture is applied to objects.
- Enable texture mapping.
 - Just use `glEnable(GL_TEXTURE_2D)` or `glEnable(GL_TEXTURE_1D)`
- Draw the scene.

Specify the Texture^{1/7}

- **Define a 2D texture.**
 - void glTexImage2D(GLenum target, GLint level, GLenum internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels)
 - See *OpenGL Programming Guide* for more detail.

Specify the Texture_{2/7}

- **target** : GL_TEXTURE_2D, GL_PROXY_TEXTURE_1D
- **level** : resolution level of the texture (for mipmap)
- **InternalFormat** : indicate which of the R, G, B, and A or other components are selected for use in describing the texels of an image. (GL_RGB, GL_ALPHA, GL_RGBA, ...)
- **width** and **height** : size of the texture
 - Must be power of 2.

Specify the Texture_{3/7}

- **format** and **data** : the format and data type of the texture image data.
 - **format** : GL_COLOR_INDEX, GL_RGB, GL_RED, GL_ALPHA, ...)
 - **data** : GL_BYTE, GL_UNSIGNED_SHORT, GL_FLOAT, ...)
- OpenGL does not provide any loader for specific image formats.

Specify the Texture_{4/7}

- For texture which size is not the power of 2 use `glScaleImage()` to scale the texture image.
 - Linear interpolation and box filtering.
 - See OpenGL Programming Guide for more detail.
- The frame buffer can also be used as texture by `glCopyTexImage2D()`.
 - See *OpenGL Programming Guide* for more detail.

Specify the Texture_{5/7}

- **Modifying the texture data**
 - void glTexSubImage2D (GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels)
 - Define a 2D texture image to replace all or part of a contiguous subregion.
 - **xoffset** and **yoffset** : specify the texels offset in x and y direction.
 - Other parameters are the same as glTexImage2D().

Specify the Texture_{6/7}

- Also, the frame buffer can be used as source of image data to modify the texture by `glCopyTexSubImage2D()`.

Specify the Texture^{7/17}

- **One-Dimensional Textures**
 - `glTexImage1D()`
 - `glCopyTexImage1D()`
 - `glTexSubImage1D()`
 - `glCopyTexSubImage1D()`
 - The parameters are the same as those of 2D.

Apply texture to objects_{1/2}

- Using `glTexEnv()` to indicate how textures are applied to objects (how the texels are combined with the original pixels).
 - `void glTexEnv{if}[v](GLenum target, GLenum pname, TYPE param)`
 - `target` must be `GL_TEXTURE_ENV`
 - If `pname` is `GL_TEXTURE_ENV_MODE`, `param` can be `GL_DECAL`, `GL_REPLACE`, `GL_MODULATE`, `GL_BLEND`
 - » *See next page for how it works*
 - If `pname` is `GL_TEXTURE_ENV_COLOR`, `param` is an array of R, G, B, A components.

Apply texture to objects_{2/2}

Base Internal 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	$C = C_t, A = A_t$	$C = C_f C_t, A = A_f A_t$

Base Internal 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	$C = C_f(1-A_t) + C_t A_t, A = A_t$	$C = C_f(1-C_t) + C_c C_t, A = A_f A_t$

X_c indicates the values assigned with GL_TEXTURE_ENV_COLOR.
X_t is the values from texture, **X_f** is the values in the color buffers.
C is the final output color value and **A** is the output alpha value.

Assign texture coordinates

- `void glTexCoord{1234}{sifd}[v](TYPEcoords)`
 - Uses like the `glColor()`, `glNormal()`, ... described before.
 - The texture coordinates are floating points between 0.0 and 1.0.

» *Ex.*

```
glBegin(GL_TRIANGLES);
    glTexCoord2f(0.0, 0.0);
    glVertex2f(-1.0, -1.0);
    glTexCoord2f(1.0, 0.0);
    glVertex2f(1.0, -1.0);
    glTexCoord2f(0.0, 1.0);
    glVertex2f(0.0, 1.0);
glEnd();
```

Texture Example_{1/5}

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>

static GLubyte checkerImage0[256][256][4];           // the checker image
static GLuint texObject;                             // texture object

// initially make a checker image by code
void makeCheckerImage(void) {
    GLubyte c;

    for(int i=0; i<256; i++)
        for(int j=0; j<256; j++) {
            c = (((i&0x20)==0)^(j&0x20)==0)*255;
            checkerImage0[i][j][0] = checkerImage0[i][j][1] = checkerImage0[i][j][2] = c;
            checkerImage0[i][j][3] = 255;
        }
}
```

Texture Example_{2/5}

```
void GL_init(void) {
    makeCheckImage();
    glClearColor (0.0, 0.0, 0.0, 0.0);
    // specify the texture with image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA, GL_UNSIGNED_BYTE,
    checkImage0);
    // set texture function
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    // apply filters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    // enable 2D texture
    glEnable(GL_TEXTURE_2D);
}
```

Texture Example_{3/5}

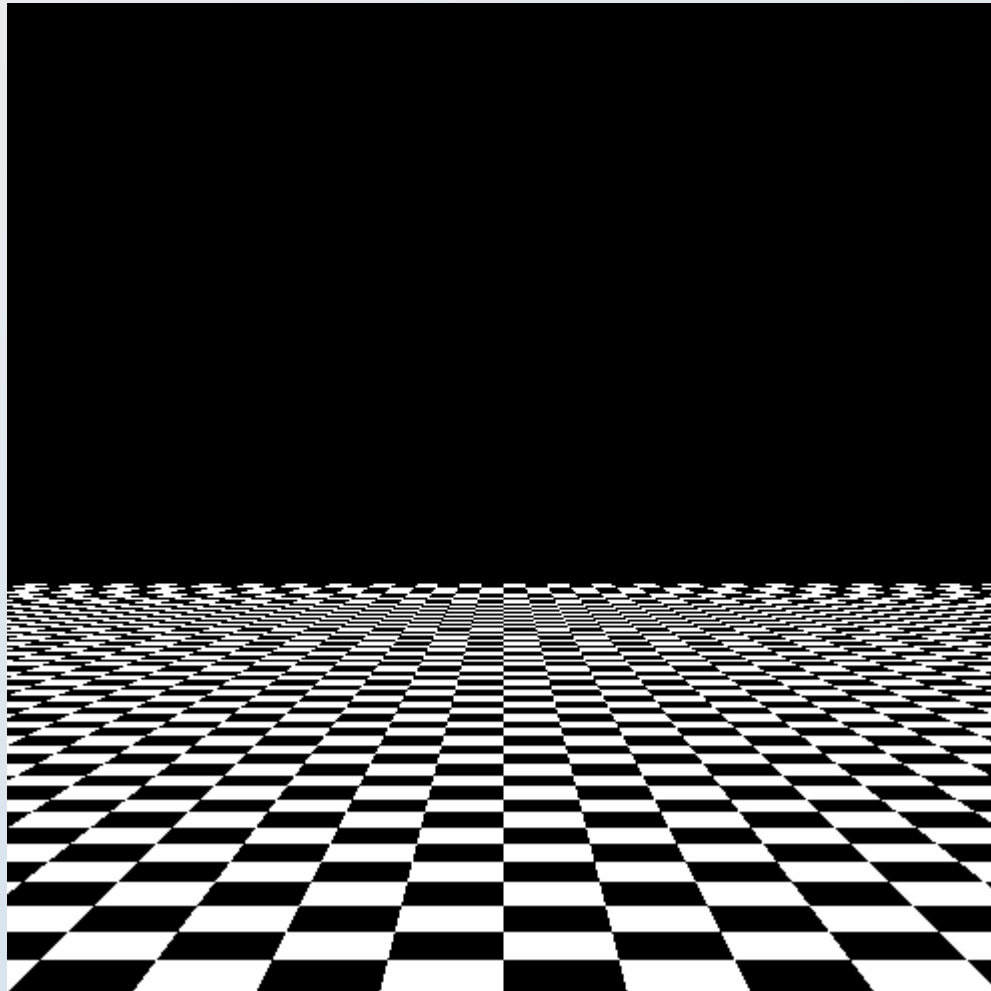
```
void GL_display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0);
        glTexCoord2f(0.0, 10.0);
        glTexCoord2f(10.0, 10.0);
        glTexCoord2f(10.0, 0.0);
    glEnd();
    glFlush();
}

void GL_reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0);
}
```

Texture Example_{4/5}

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow(argv[0]);
    GL_init();
    glutDisplayFunc(GL_display);
    glutReshapeFunc(GL_reshape);
    glutMainLoop();
    return 0;
}
```

Texture Example_{5/5}



Texture Object_{1/4}

- **Only for OpenGL 1.1 or later.**
- **The faster way to switch between textures**
 - Operate texture as object
- **Steps of texture objects**
 - Generate texture names.
 - Initially bind texture object to texture data.
 - Set properties for each texture object.
 - When rendering, just rebind the texture object and render the objects.

Texture Object_{2/4}

- **Generate texture names**
 - `void glGenTextures(GLsizei n, GLuint *textureNames)`
 - Return **n** number of unused names for texture objects in array **textureNames**.
 - `GLboolean glIsTexture(GLuint textureName)`
 - Return **true** if **textureName** is currently bind to some texture object, and **false** otherwise.

Texture Object_{3/4}

- void glBindTexture(GLenum target, GLuint textureName)
 - **target** is GL_TEXTURE_2D or GL_TEXTURE_1D.
 - **textureName** is the name generated by glGenTextures()
- void glDeleteTextures(GLsizei n, const GLuint *textureNames)
 - Delete **n** texture objects, named by elements in the array **textureNames**.

Texture Object_{4/4}

```
GLuint texObject[2];                // texture object

glGenTextures(2, texObject);        // generate a new texture object

// Bind Texture 1
glBindTexture(GL_TEXTURE_2D, texObject[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA, GL_UNSIGNED_BYTE, image1);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

// Bind Texture 2
glBindTexture(GL_TEXTURE_2D, texObject[1]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA, GL_UNSIGNED_BYTE, image2);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND);

-----
// Render a sphere using texture 1:
glBindTexture(GL_TEXTURE_2D, texObject[0]);
glutSolidSphere(1.0, 16, 16);

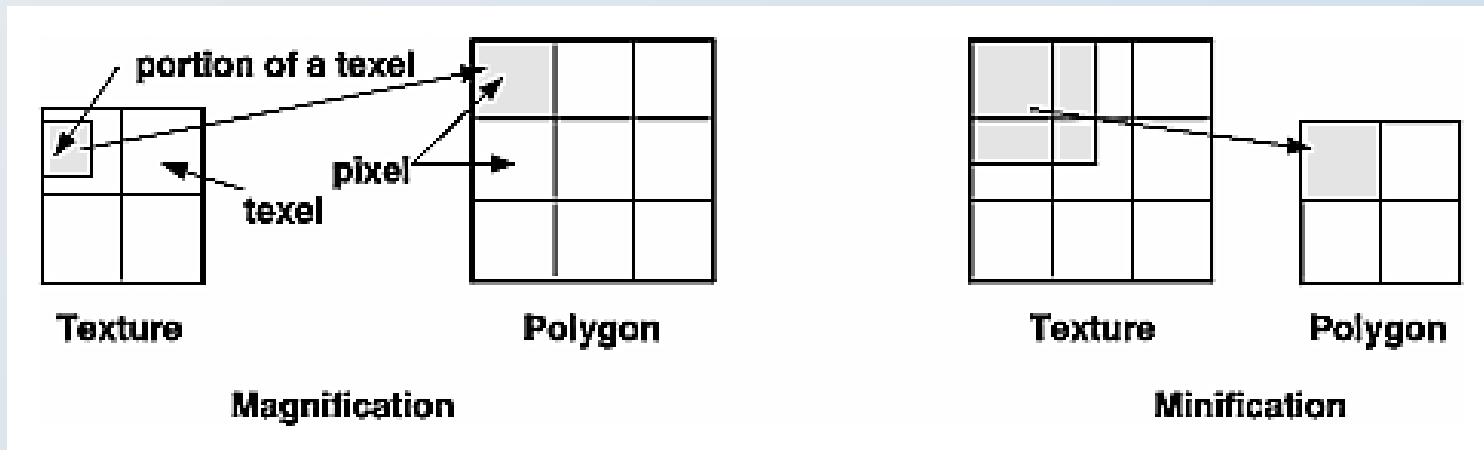
// Render a teapot using texture 2:
glBindTexture(GL_TEXTURE_2D, texObject[0]);
glutSolidTeapot(1.0);
...
```

Filtering_{1/3}

- **The problem**
 - After being mapped to a polygon and transformed into screen coordinates, the texels rarely correspond to individual pixels of the final screen image.

Filtering_{2/3}

- Two cases of problem
 - Magnification
 - Minification

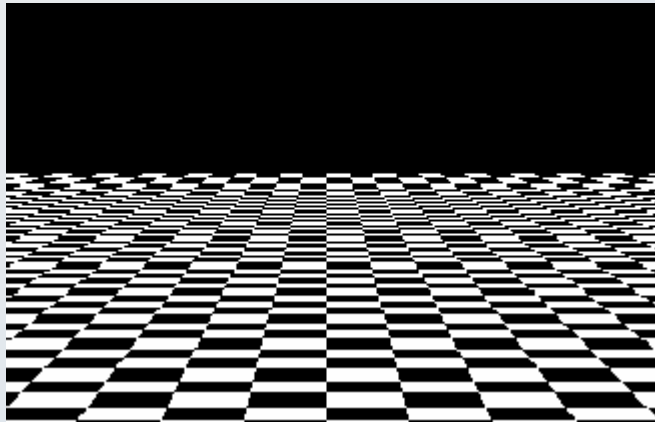


Filtering^{3/3}

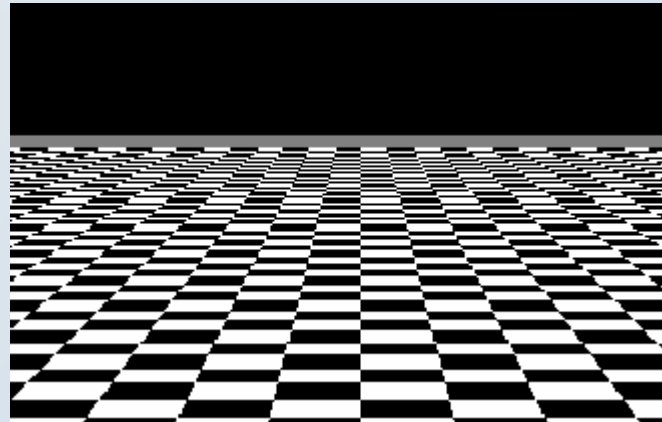
- `glTexParameter{if}(GLenum target, GLenum pname, TYPE param)`
 - **target** : `GL_TEXTURE_2D`, `GL_TEXTURE_1D`
 - **pname** : `GL_TEXTURE_MAG_FILTER`,
`GL_TEXTURE_MIN_FILTER`
 - **param** : `GL_NEAREST`, `GL_LINEAR`, ...
 - To apply mipmap, using the following parameters (only for `GL_MIN_FILTER`) :
`GL_NEAREST_MIPMAP_NEAREST` `GL_NEAREST_MIPMAP_LINEAR`
`GL_LINEAR_MIPMAP_LINEAR` `GL_LINEAR_MIPMAP_NEAREST`
 - See *OpenGL Programming Guide* for more detail.

Mipmap_{1/5}

- The Problem



Without Mipmap

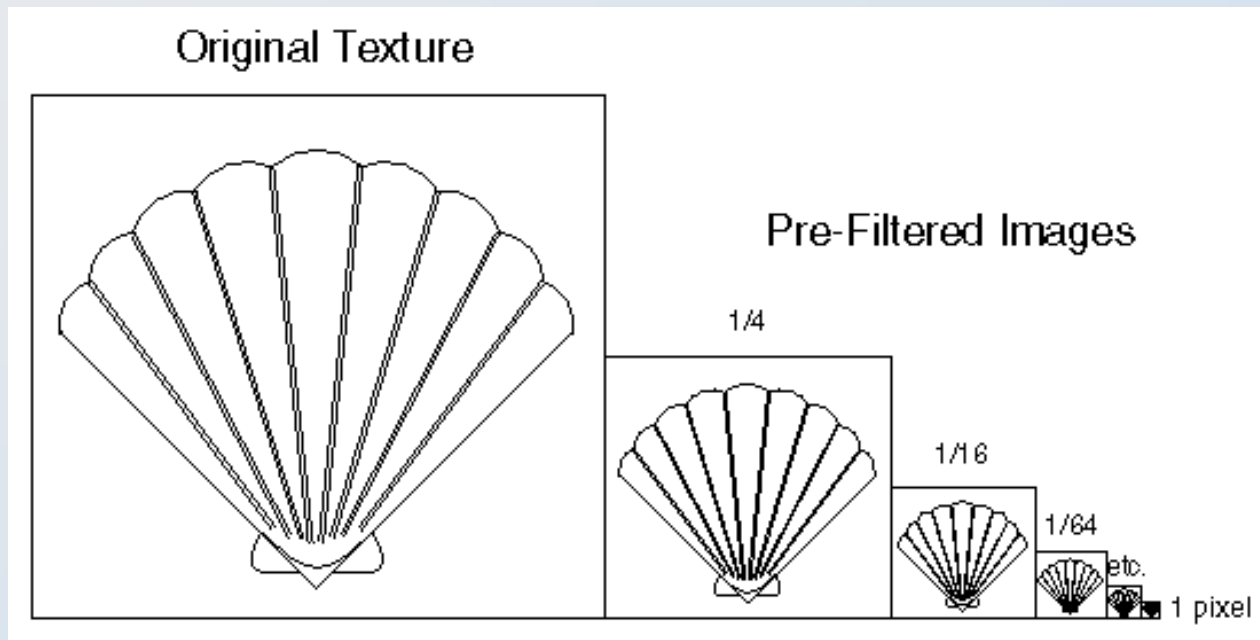


With Mipmap

Mipmap_{2/5}

- Mipmap

- A series of prefiltered texture maps of decreasing resolutions.



Mipmap_{3/5}

- Call the `glTexImage2D()` with different level, width, height and image.
- OpenGL automatically determines which texture map to use based on the size (pixels) of the objects being mapped.
- To use mipmap, all sizes of texture in powers of 2 between largest size and 1x1 map are required.

Mipmap_{4/5}

- Ex.

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 8, 8, 0,  
             GL_RGB, GL_FLOAT, checkImage0);
```

```
glTexImage2D(GL_TEXTURE_2D, 1, GL_RGB, 4, 4, 0,  
             GL_RGB, GL_FLOAT, checkImage0);
```

```
glTexImage2D(GL_TEXTURE_2D, 2, GL_RGB, 2, 2, 0,  
             GL_RGB, GL_FLOAT, checkImage0);
```

```
glTexImage2D(GL_TEXTURE_2D, 3, GL_RGB, 1, 1, 0,  
             GL_RGB, GL_FLOAT, checkImage0);
```

- For a 8x8 texture, 4 levels mipmaps are required.

Mipmap_{5/5}

- Automatic Mipmap generation
 - Construct the highest level (level-0) map
 - Use `gluBuild2DMipmaps()`.
 - int `gluBuild2DMipmaps`(GLenum target, GLint components, GLint width, GLint height, GLenum format, GLenum type, void* data)
 - » *Parameters are the same as `glTexImage2D()`.*

Repeating and Clamping^{1/3}

- Texture coordinate is defined between 0.0 - 1.0
- `glTexParameter*()` can also be used to define how values beyond the range will be treated.
 - `void glTexParameter{if}(GLenum target, GLenum pname, TYPE param)`
 - `target` : `GL_TEXTURE_2D`, `GL_TEXTURE_1D`
 - `pname` : `GL_TEXTURE_WRAP_S`,
`GL_TEXTURE_WRAP_T`

Repeating and Clamping^{2/3}

pname	param
GL_TEXTURE_WRAP_S	GL_REPEAT, GL_CLAMP
GL_TEXTURE_WRAP_T	GL_REPEAT, GL_CLAMP
GL_TEXTURE_MAG_FILTER	GL_NEAREST, GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST
GL_TEXTURE_BORDER_COLOR	color value
GL_TEXTURE_PRIORITY	priority of current texture image between 0.0 – 1.0

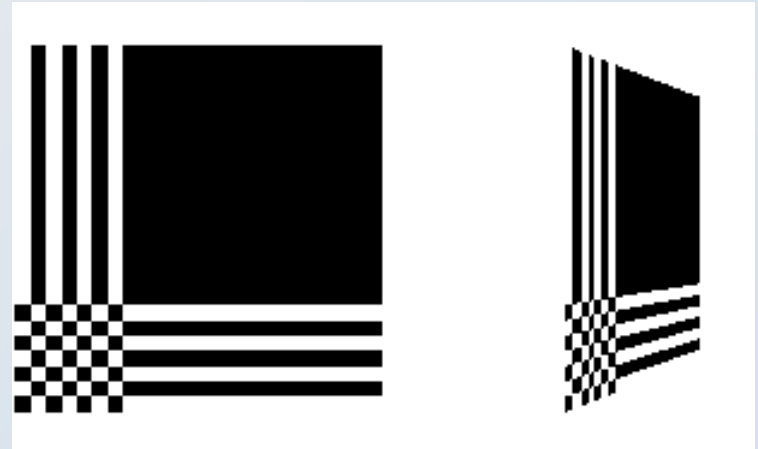
Repeating and Clamping_{3/3}

```
glBegin(GL_QUADS);  
glTexCoord2f(0.0, 0.0); glVertex2f(-2.0, -1.0);  
glTexCoord2f(0.0, 3.0); glVertex2f(-2.0, 1.0);  
glTexCoord2f(0.0, 3.0); glVertex2f(0.0, 1.0);  
glTexCoord2f(3.0, 0.0); glVertex2f(0.0, -1.0);  
glEnd();
```

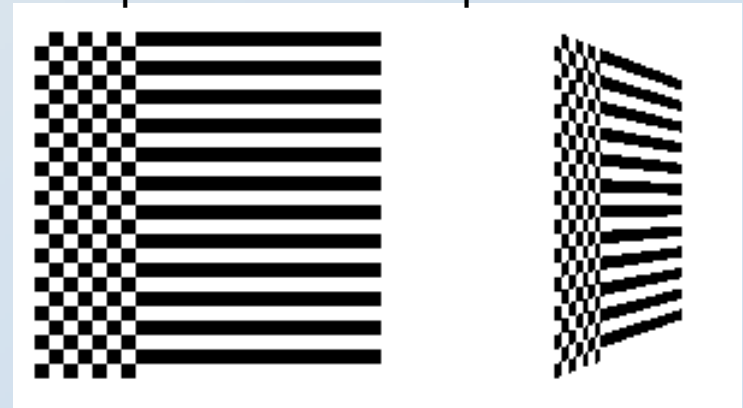
Repeat both S, T



Clamp both S, T



Repeat T but Clamp S



Texture Example_{1/3}

```
void GL_init(void) {
    makeCheckImage();
    glClearColor (0.0, 0.0, 0.0, 0.0);

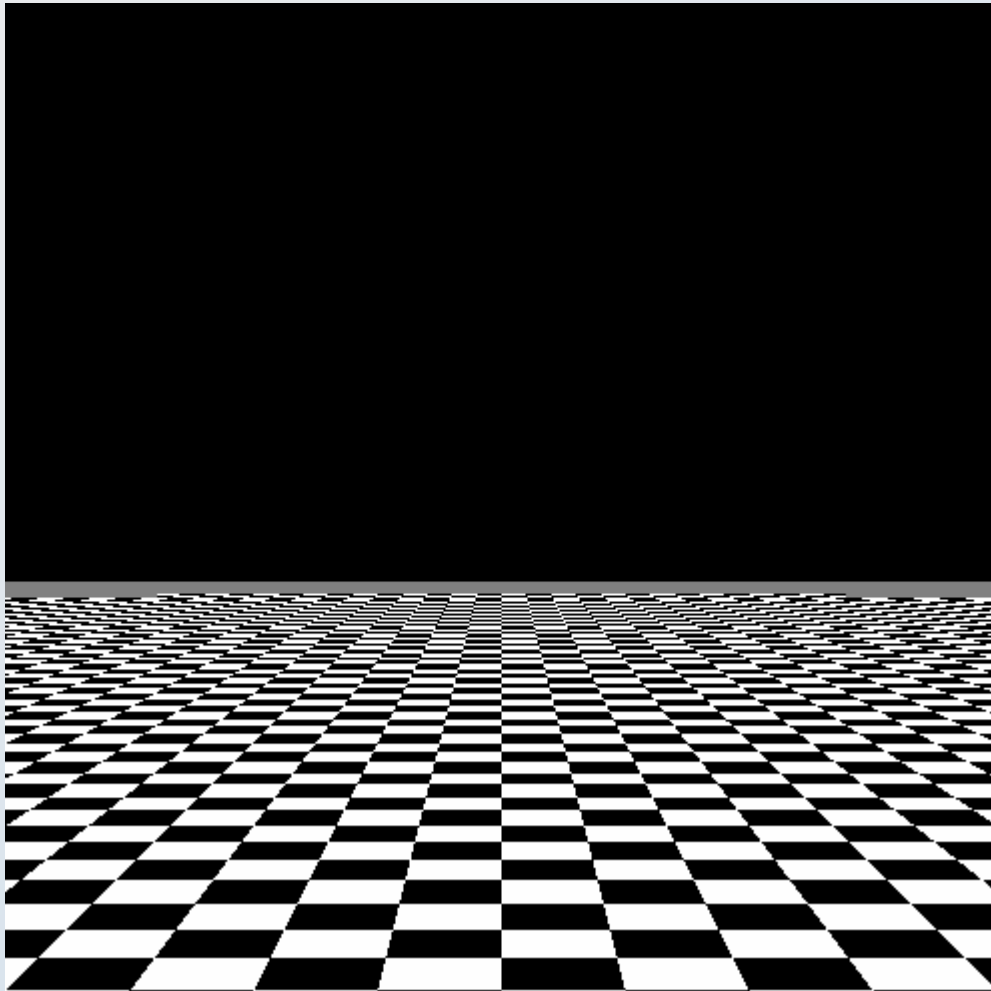
    glGenTextures(1, &texObject);           // generate a new texture object
    glBindTexture(GL_TEXTURE_2D, texObject); // bind texture object for later use
    // repeat texture
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // filter
    glTexParameteri(GL_TEXTURE_2D,
        GL_TEXTURE_MAG_FILTER, GL_NEAREST_MIPMAP_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
        GL_NEAREST_MIPMAP_NEAREST);
    // specify the texture with image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA, GL_UNSIGNED_BYTE,
        checkImage0);
}
```

Texture Example_{2/3}

```
// automatically generate mipmaps
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, 256, 256, GL_RGBA, GL_UNSIGNED_BYTE,
checkImage0);
// set texture function
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
// enable 2D texture
glEnable(GL_TEXTURE_2D);
}

void GL_display(void) {
glClear(GL_COLOR_BUFFER_BIT);
// bind the texture object for rendering
glBindTexture(GL_TEXTURE_2D, texObject);
glBegin(GL_QUADS); // assign texture coordinate and render it
    glTexCoord2f(0.0, 0.0); glVertex3f(-1000.0, -100.0, 1000.0);
    glTexCoord2f(0.0, 10.0); glVertex3f(1000.0, -100.0, 1000.0);
    glTexCoord2f(10.0, 10.0); glVertex3f(1000.0, -100.0, 0.0);
    glTexCoord2f(10.0, 0.0); glVertex3f(-1000.0, -100.0, 0.0);
glEnd();
glFlush();
}
```

Texture Example_{3/3}



Automatic Texture-Coordinate Generation_{1/4}

- `void glTexGen{ifd}[v](GLenum coord, GLenum pname, TYPEparam)`
 - `coord` can be `GL_S`, `GL_T`, `GL_R`, `GL_Q`

pname

param

`GL_TEXTURE_GEN_MODE`

`GL_OBJECT_LINEAR`,
`GL_GL_EYE_LINEAR`,
`GL_SPHERE_MAP`

`GL_OBJECT_PLANE`

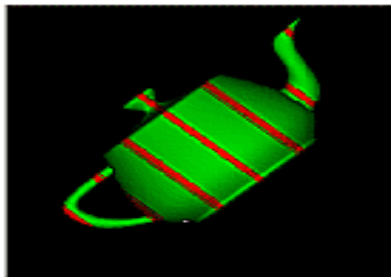
Plane coefficient array

`GL_EYE_PLANE`

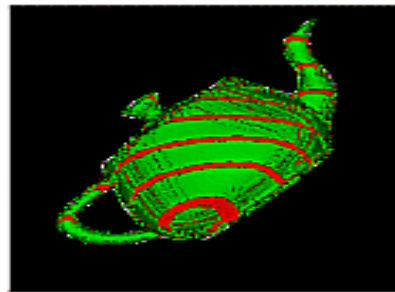
Plane coefficient array

Automatic Texture-Coordinate Generation_{2/4}

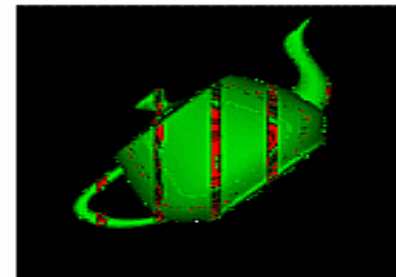
- Plane coefficient array
 - Generated plane = $p1X + p2Y + p3Z + p4W$
 - $p1, p2, p3, p4$ are components of plane coefficient array.
- `GL_TEXTURE_GEN_MODE`
 - (a) `GL_OBJECT_LINEAR` with plane $x = 0$
 - (b) `GL_OBJECT_LINEAR` with plane $x+y+z = 0$
 - (c) `GL_EYE_LINEAR` with plane $x = 0$



(a)



(b)



(c)

From "OpenGL
Programming Guide"

Automatic Texture-Coordinate Generation_{3/4}

- **Ex. Environment Map**

- Apply the following code

```
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);
```

- The `GL_SPHERE_MAP` constant creates the proper texture coordinates for the environment mapping

Automatic Texture-Coordinate Generation_{4/4}

Sphere Map



Object with Environment Map



From "OpenGL
Programming Guide"

Any Question

