The University of New Mexico

OpenGL Texture Mapping

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

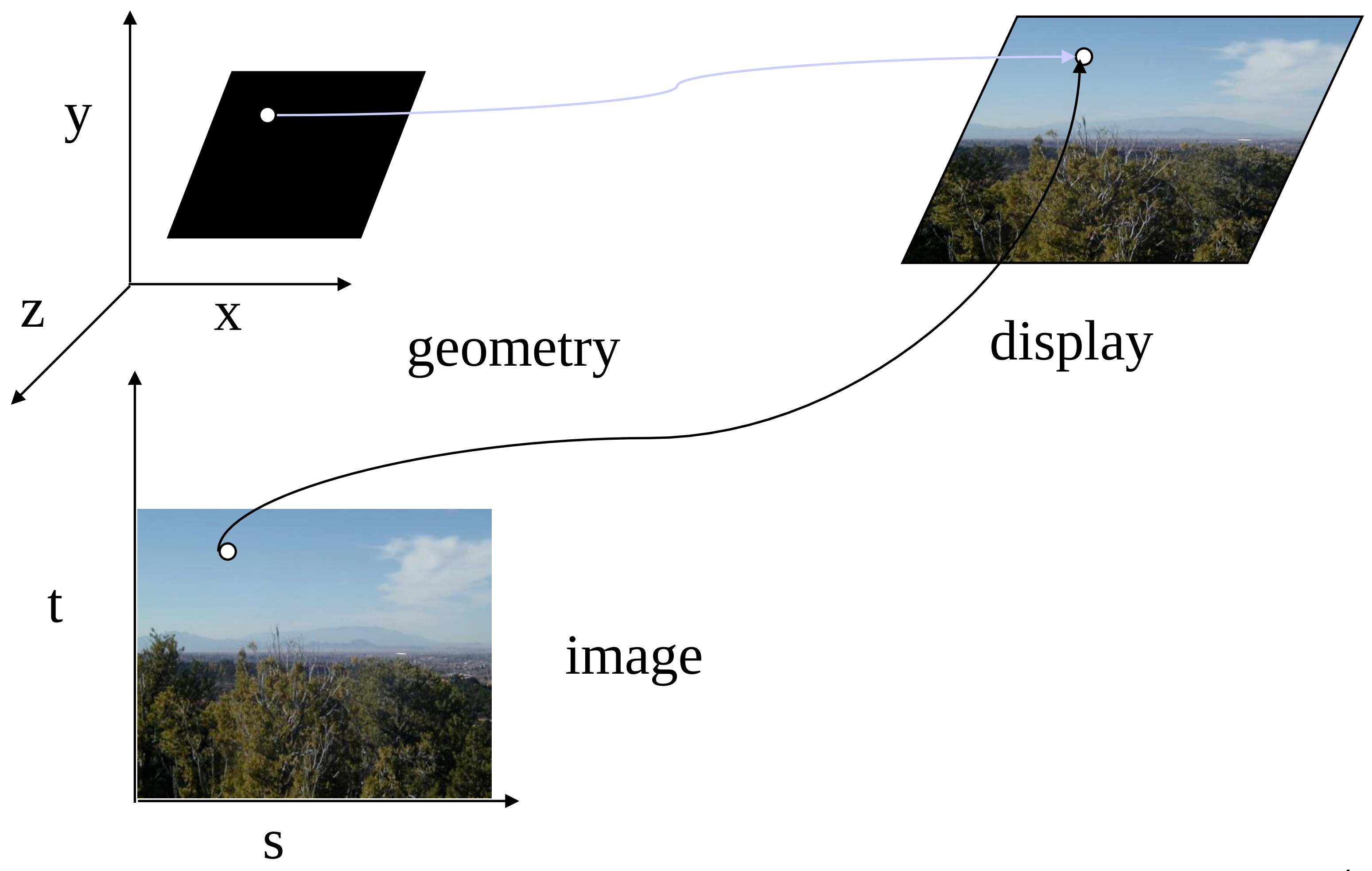
- Introduce the OpenGL texture functions and options

Basic Stragegy

Three steps to applying a texture

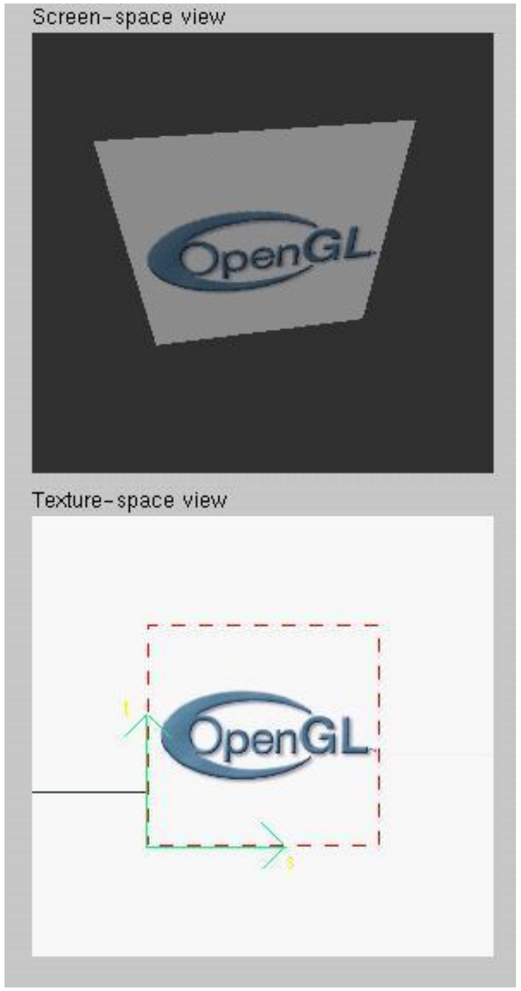
1. specify the texture
   - read or generate image
   - assign to texture
   - enable texturing
2. assign texture coordinates to vertices
   - Proper mapping function is left to application
3. specify texture parameters
   - wrapping, filtering

Texture Mapping



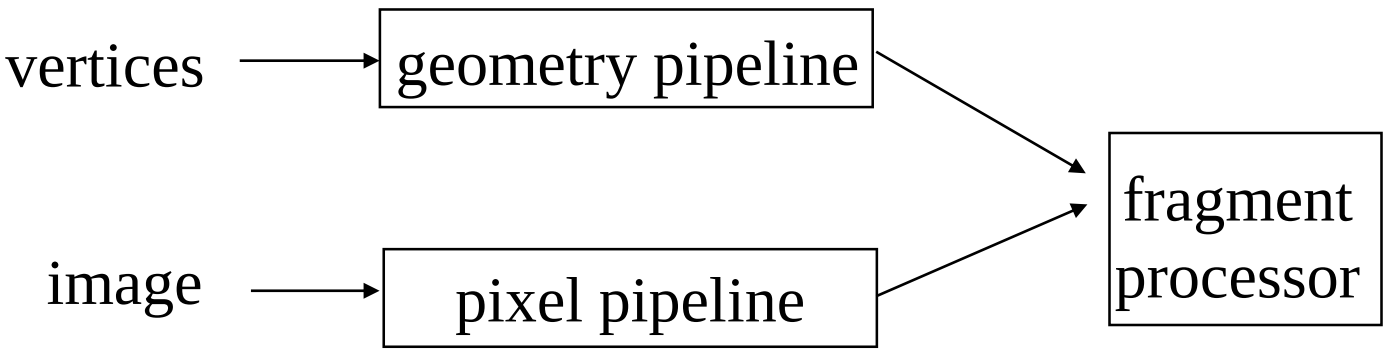
Texture Example

- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



Texture Mapping and the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join during fragment processing
  - "complex" textures do not affect geometric complexity



Specifying a Texture Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory

```
Glubyte my_texels[512][512][3];
```

- Define as any other pixel map
  - Scanned image
  - Generate by application code
- Enable texture mapping
  - `glEnable(GL_TEXTURE_2D)`
  - OpenGL supports 1-4 dimensional texture maps

The University of New Mexico

Define Image as a Texture

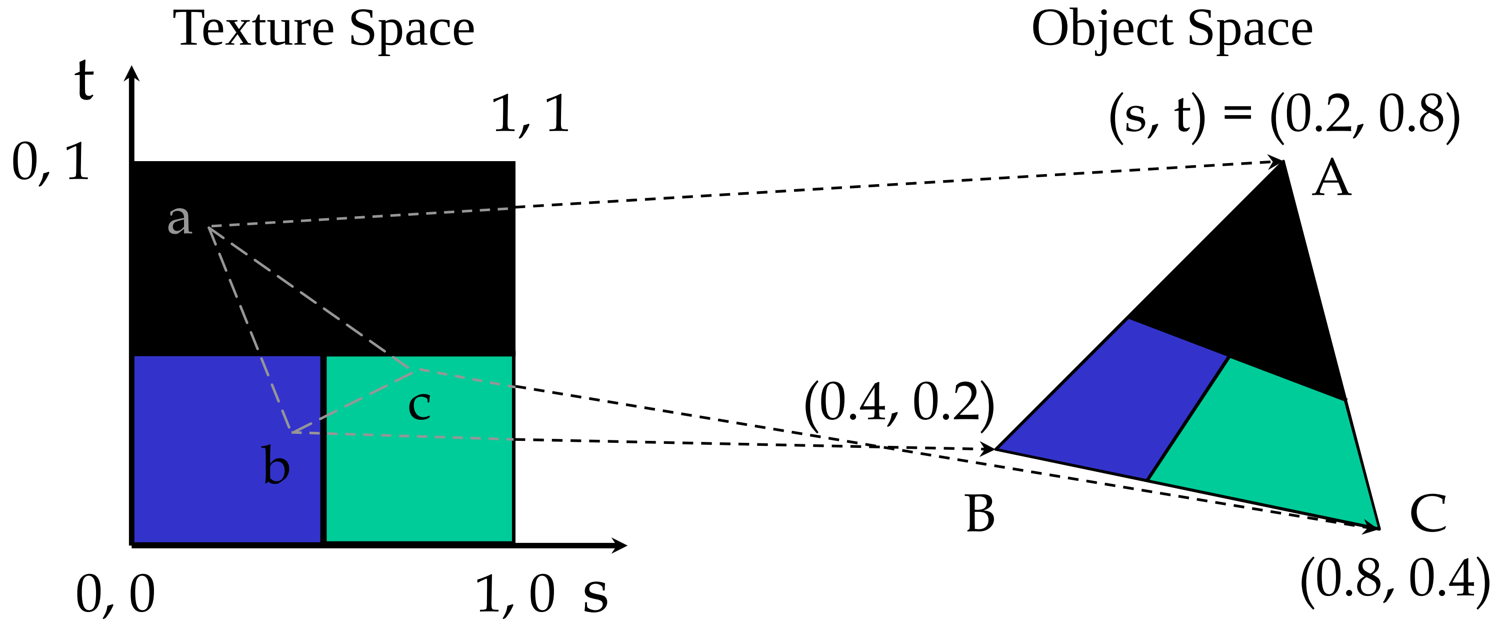
```
glTexImage2D( target, level, components,
    w, h, border, format, type, texels );
```

- `target`: type of texture, e.g. `GL_TEXTURE_2D`
- `level`: used for mipmapping (discussed later)
- `components`: elements per texel
- `w, h`: width and height of `texels` in pixels
- `border`: used for smoothing (discussed later)
- `format and type`: describe texels
- `texels`: pointer to texel array

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
 GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

Mapping a Texture

- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex



Typical Code

```
offset = 0;
GLuint vPosition = glGetAttribLocation( program,
 "vPosition" );
glEnableVertexAttribArray( vPosition );
glVertexAttribPointer( vPosition, 4, GL_FLOAT,
 GL_FALSE, 0,BUFFER_OFFSET(offset) );

offset += sizeof(points);
GLuint vTexCoord = glGetAttribLocation( program,
 "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2,GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(offset) );
```

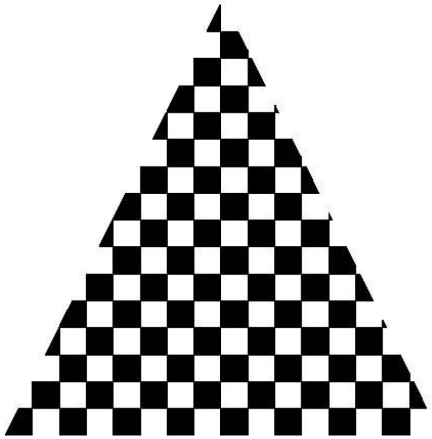
The University of New Mexico

Interpolation

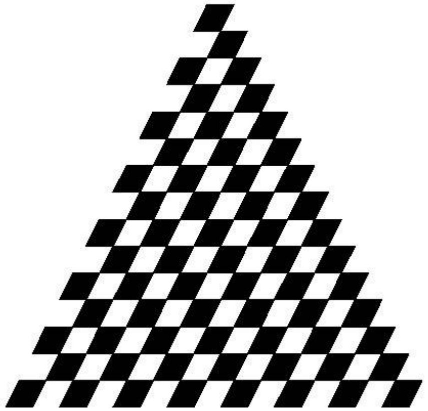
OpenGL uses interpolation to find proper texels from specified texture coordinates

Can be distortions

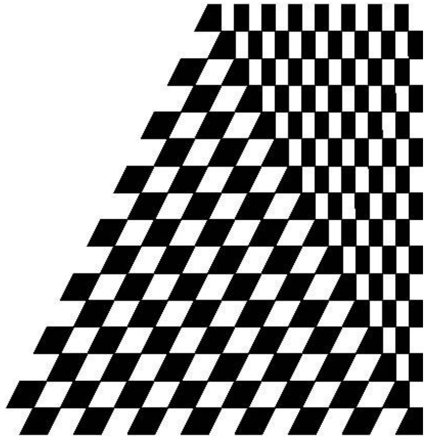
good selection of tex coordinates



poor selection of tex coordinates



texture stretched over trapezoid showing effects of bilinear interpolation



Texture Parameters

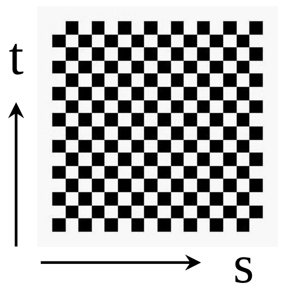
- OpenGL has a variety of parameters that determine how texture is applied
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
  - Environment parameters determine how texture mapping interacts with shading

Wrapping Mode

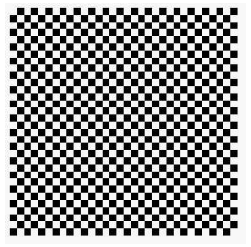
Clamping: if s,t > 1 use 1, if s,t <0 use 0

Wrapping: use s,t modulo 1

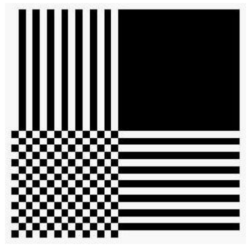
```
glTexParameteri( GL_TEXTURE_2D,
        GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D,
        GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL_REPEAT wrapping

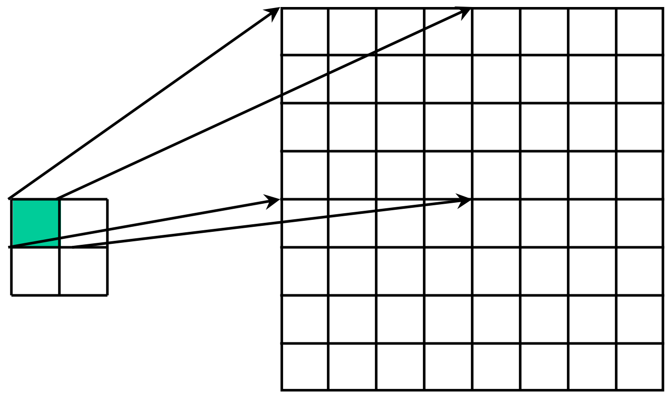


GL_CLAMP wrapping

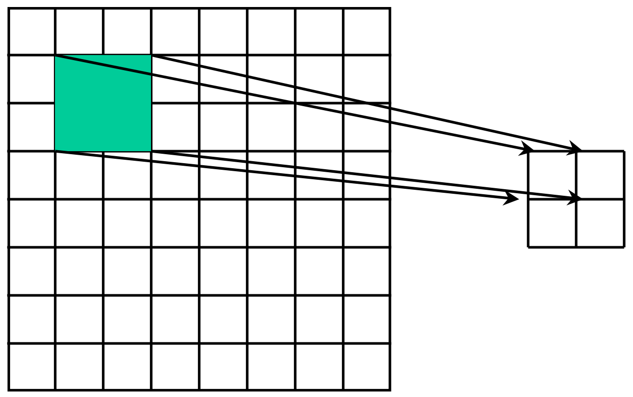
Magnification and Minification

More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Magnification



Minification

Filter Modes

Modes determined by

- `glTexParameteri( target, type, mode )`

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
                GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_LINEAR);
```

Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1)

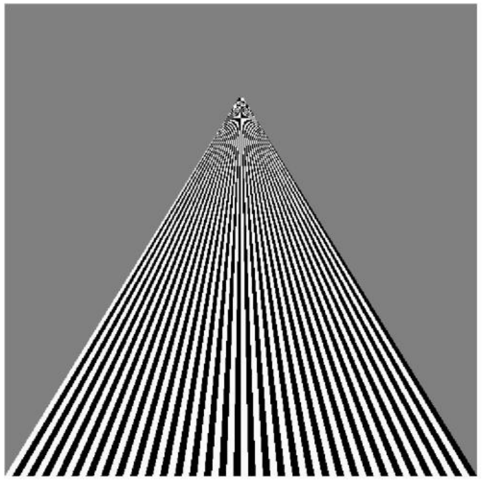
Mipmapped Textures

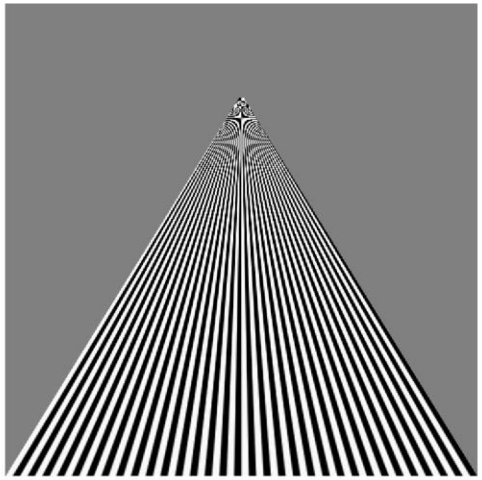
- *Mipmapping* allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

```
glTexImage2D( GL_TEXTURE_*D, level, … )
```

Example

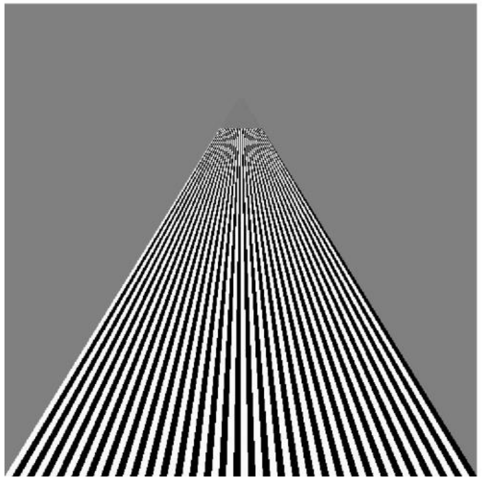
point sampling

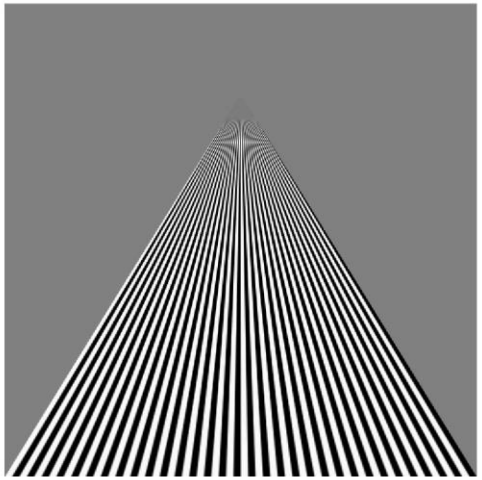




linear filtering

mipmapped point sampling





mipmapped linear filtering

The University of New Mexico

Texture Functions

- Controls how texture is applied
  - `glTexEnv{fi}[v]( GL_TEXTURE_ENV, prop, param )`
- `GL_TEXTURE_ENV_MODE` modes
  - `GL_MODULATE:` modulates with computed shade
  - `GL_BLEND:` blends with an environmental color
  - `GL_REPLACE:` use only texture color
  - `GL(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);`
- Set blend color with `GL_TEXTURE_ENV_COLOR`

Using Texture Objects

1. specify textures in texture objects
2. set texture filter
3. set texture function
4. set texture wrap mode
5. set optional perspective correction hint
6. bind texture object
7. enable texturing
8. supply texture coordinates for vertex
   - coordinates can also be generated

Other Texture Features

- Environment Maps
  - Start with image of environment through a wide angle lens
    - Can be either a real scanned image or an image created in OpenGL
  - Use this texture to generate a spherical map
  - Alternative is to use a cube map
- Multitexturing
  - Apply a sequence of textures through cascaded texture units

Applying Textures

- Textures are applied during fragments shading by a **sampler**
- Samplers return a texture color from a texture object

```
in vec4 color;  //color from rasterizer
in vec2 texCoord; //texure coordinate from rasterizer
uniform sampler2D texture; //texture object from application

void main()  {
    gl_FragColor = color * texture2D( texture, texCoord );
}
```

The University of New Mexico

Vertex Shader

- Usually vertex shader will output texture coordinates to be rasterized
- Must do all other standard tasks too
  - Compute vertex position
  - Compute vertex color if needed

```
in vec4 vPosition; //vertex position in object coordinates
in vec4 vColor;  //vertex color from application
in vec2 vTexCoord; //texture coordinate from application

out vec4 color; //output color to be interpolated
out vec2 texCoord; //output tex coordinate to be interpolated
```

Checkerboard Texture

```
GLubyte image[64][64][3];

// Create a  64 x 64 checkerboard pattern
   for ( int i = 0; i < 64; i++ ) {
       for ( int j = 0; j < 64; j++ ) {
           GLubyte c = (((i & 0x8) == 0) ^ ((j & 0x8)  == 0)) * 255;
           image[i][j][0]  = c;
           image[i][j][1]  = c;
           image[i][j][2]  = c;
```

Adding Texture Coordinates

```
void quad( int a, int b, int c, int d )
{
    quad_colors[Index] = colors[a];
    points[Index] = vertices[a];
    tex_coords[Index] = vec2( 0.0, 0.0 );
    index++;
   quad_colors[Index] = colors[a];
    points[Index] = vertices[b];
    tex_coords[Index] = vec2( 0.0, 1.0 );
    Index++;

// other vertices
}
```

Texture Object

```
GLuint textures[1];
glGenTextures( 1, textures );

glBindTexture( GL_TEXTURE_2D, textures[0] );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, TextureSize,
   TextureSize, 0, GL_RGB, GL_UNSIGNED_BYTE, image );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
    GL_REPEAT );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
     GL_REPEAT );
glTexParameterf( GL_TEXTURE_2D,
      GL_TEXTURE_MAG_FILTER, GL_NEAREST );
glTexParameterf( GL_TEXTURE_2D,
      GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glActiveTexture( GL_TEXTURE0 );
```

Linking with Shaders

```
GLuint vTexCoord = glGetAttribLocation( program, "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2, GL_FLOAT, GL_FALSE, 0,
                       BUFFER_OFFSET(offset) );

// Set the value of the fragment shader texture sampler variable
//   ("texture") to the the appropriate texture unit. In this case,
//   zero, for GL_TEXTURE0 which was previously set by calling
//   glActiveTexture().
glUniform1i( glGetUniformLocation(program, "texture"), 0 );
```