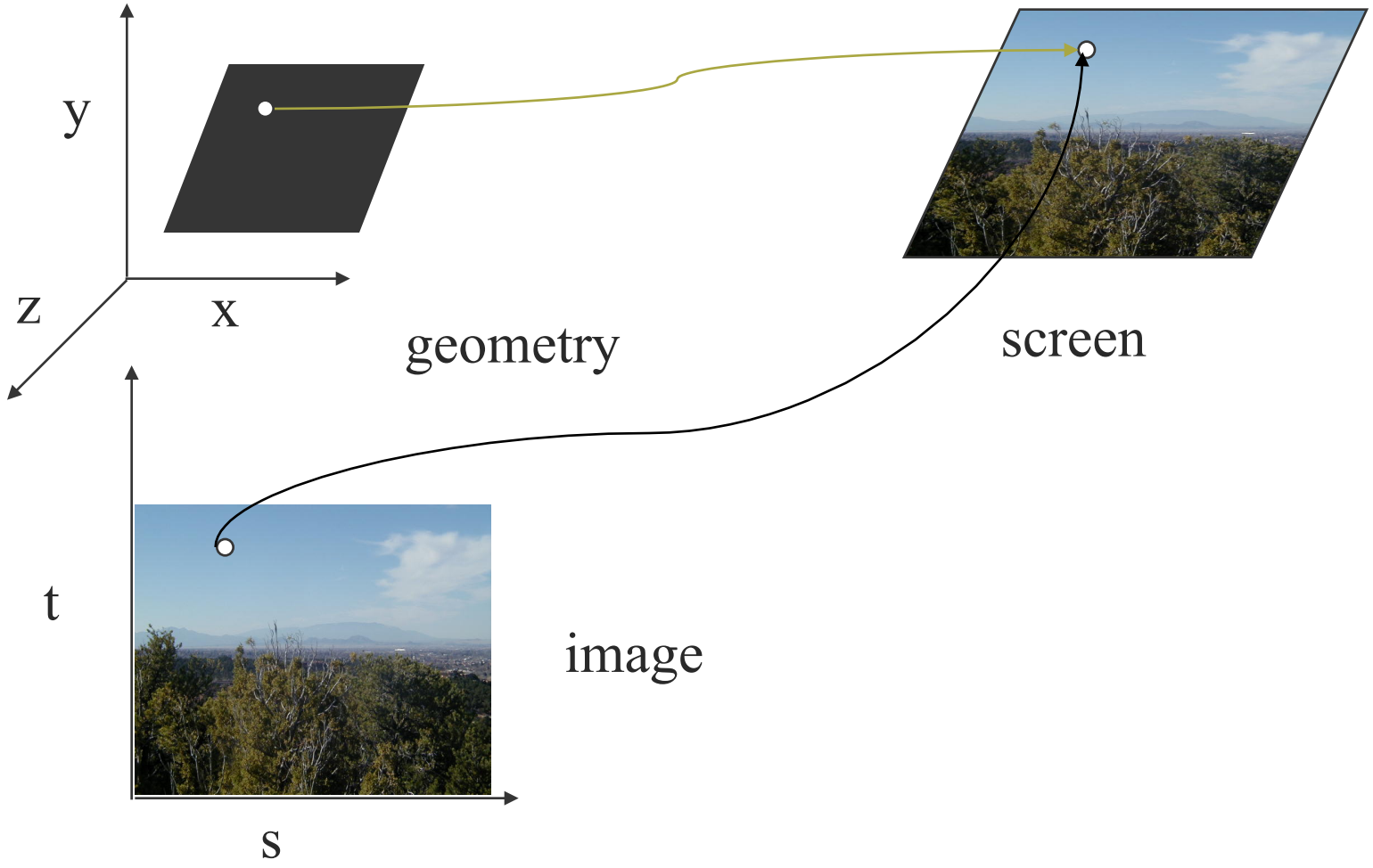
OpenGL
Texture Mapping and Related

Basic Strategy

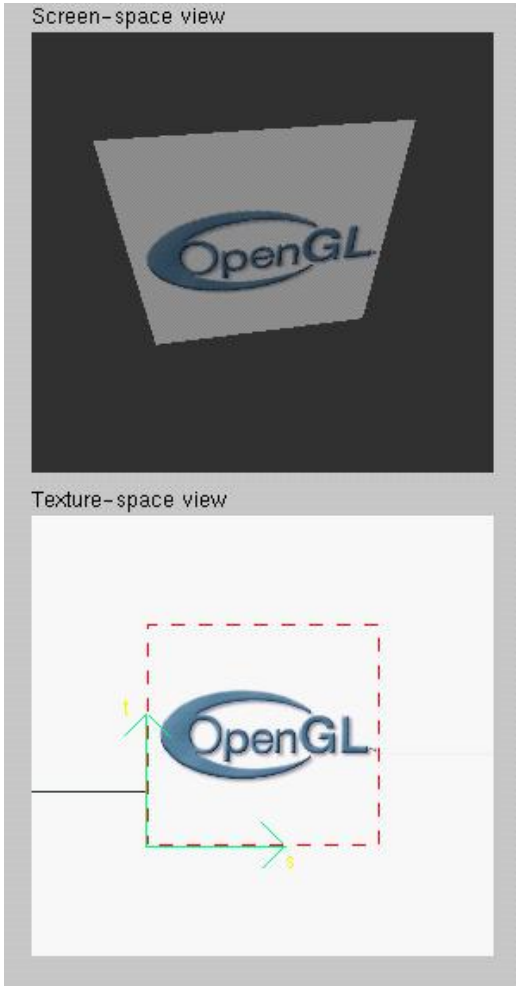
- Three steps to applying a texture
  1. specify the texture
     - read or generate image
     - assign to texture
     - enable texturing
  2. assign texture coordinates to vertices
     - Proper mapping function is left to application
  3. specify texture parameters
     - wrapping, filtering

Texture Mapping



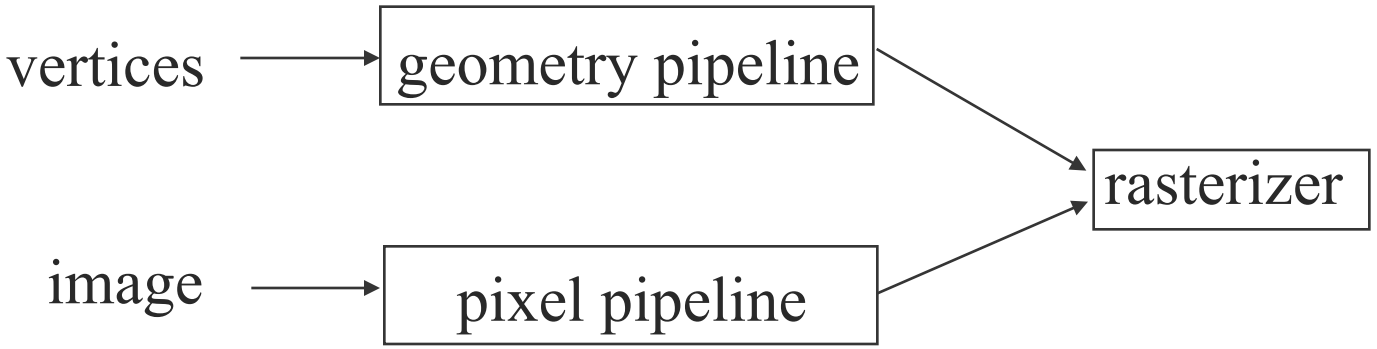
Texture Example

- 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 at the rasterizer
  - "complex" textures do not affect geometric complexity



Specify 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
  - Scan
  - Via application code
- Enable texture mapping
  - `glEnable(GL_TEXTURE_2D)`
  - OpenGL supports 1-4 dimensional texture maps

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 (use 0, more later)

`components:` number of color components 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

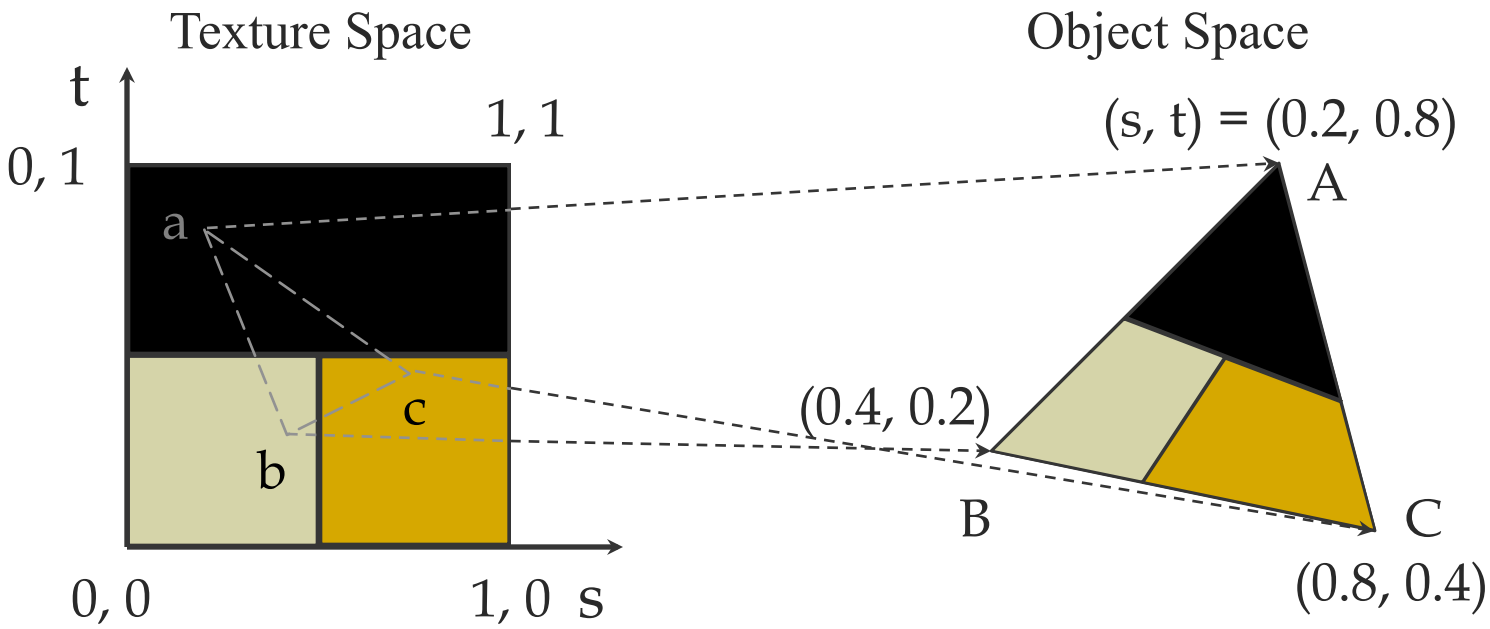
```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB,
   GL_UNSIGNED_BYTE, my_texels);
```

Converting A Texture Image

- OpenGL requires texture dimensions to be powers of 2
- If dimensions of image are not powers of 2
  - `gluScaleImage( format, w_in, h_in, type_in, *data_in, w_out, h_out, type_out, *data_out );`
    - `data_in` is source image
    - `data_out` is for destination image
- Image interpolated and filtered during scaling

Mapping a Texture

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



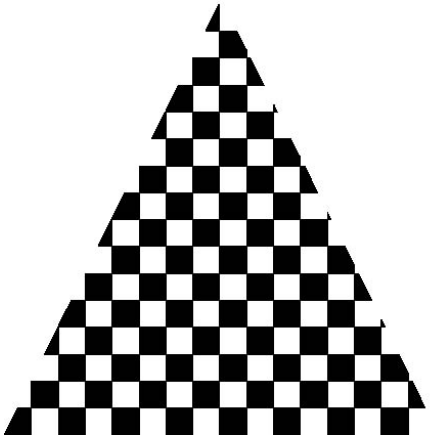
Typical Code

```
glBegin(GL_POLYGON);
 glNormal3f(u0, v0, w0);
 glTexCoord2f(s0, t0);
 glVertex3f(x0, y0, z0);
 glNormal3f(u1, v1, w1);
 glTexCoord2f(s1, t1);
 glVertex3f(x1, y1, z1);
      .
      .
glEnd();
```

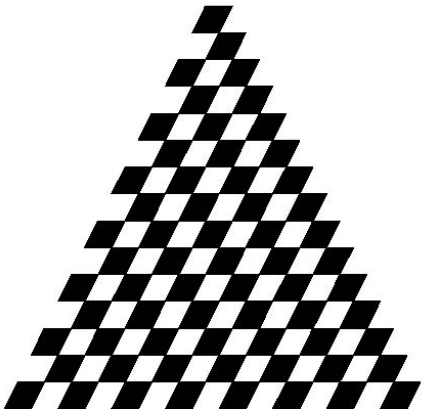
Interpolation

- OpenGL uses bilinear interpolation to find proper values from specified texture coordinates
- Can have distortions

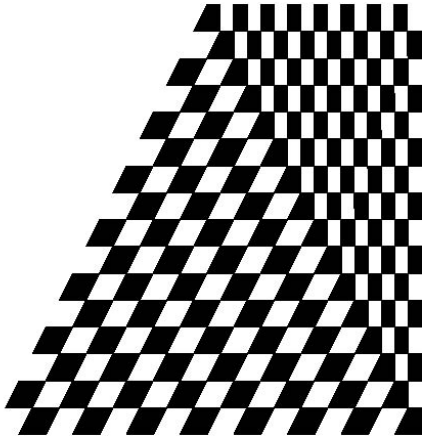
good selection
of tex coordinates



poor selection
of tex coordinates



texture stretched
over trapezoid
showing effects of
bilinear interpolation



Texture Parameters

- OpenGL uses a variety of parameters that determine how texture is applied
  - Wrapping parameters – determine what happens of s and t outside of 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

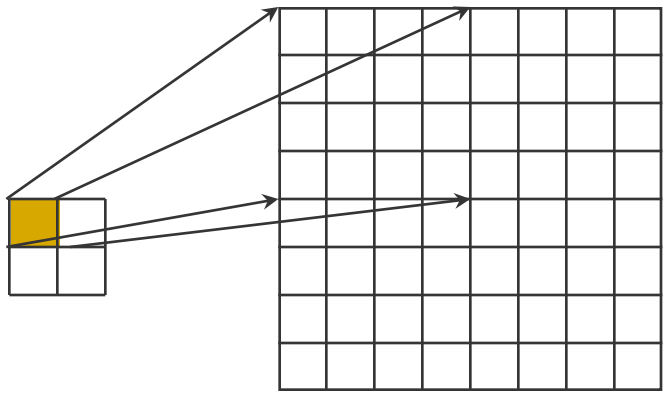
Wrapping: use s,t modulo 1

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

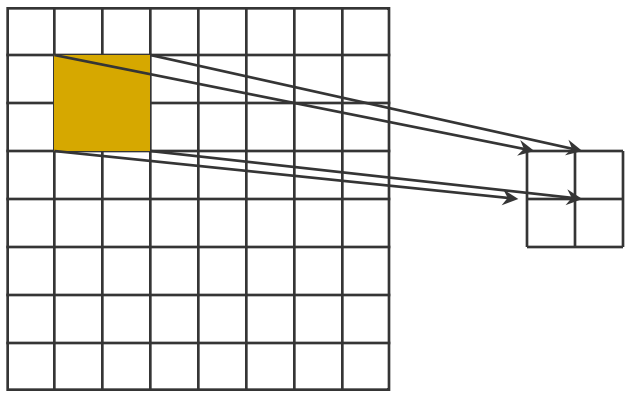
Magnification and Minification

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_2D, level, … )
```

- GLU mipmap builder routines will build textures of all power of 2 resolutions from a given image

```
gluBuild2DMipmaps(GL_TEXTURE_2D,3,64,64,
GL_RGB,GL_UNSIGNED_BYTE,my_texels);
```

Mipmap Modes

- point sampling with best mipmap

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

- linear filtering with best mipmap

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

- point sampling with linear filtering btw maps

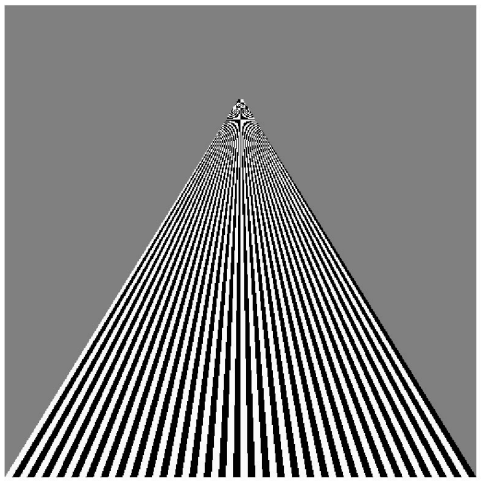
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_NEAREST_MIPMAP_LINEAR);
```

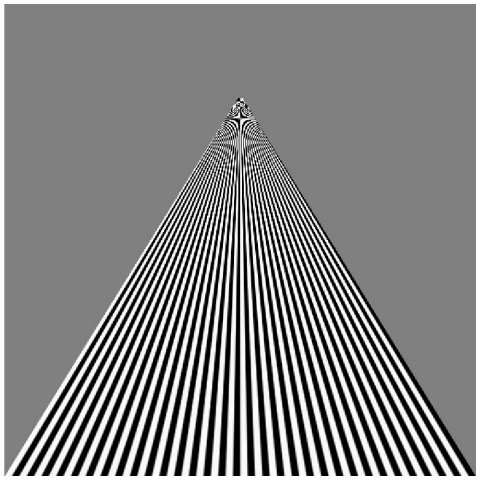
- both linear

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

Example

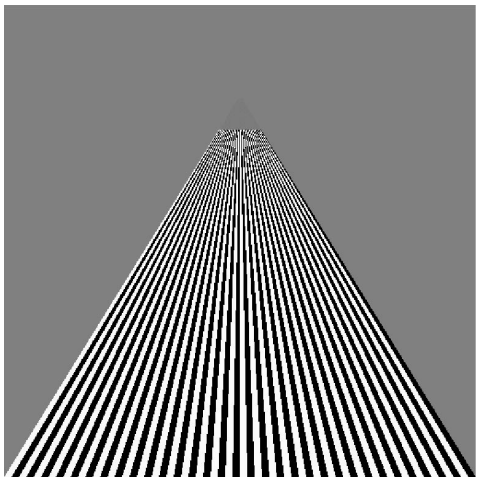
point sampling

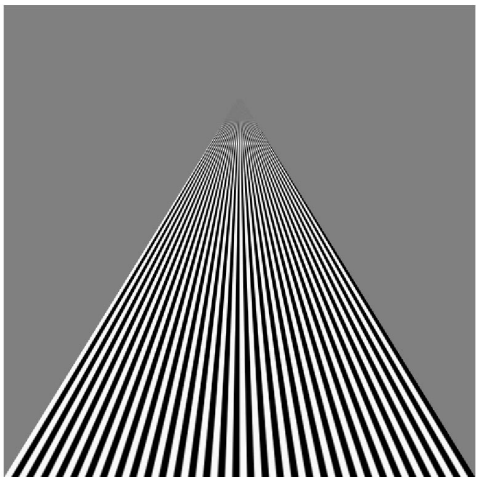




linear filtering

mipmapped point sampling





mipmapped linear filtering

Texture and Shading

- Controls how texture is applied with regard to shading

```
glTexEnv{fi}[v]( GL_TEXTURE_ENV,
    GL_TEXTURE_ENV_MODE, 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**

Perspective Correction Hint

- Proper texture mapping also depends on projection
  - linear map works fine for ortho
  - perspective needs correction
- Noticeable for polygons “on edge”
  - `glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )`

  where `hint` is one of
    - `GL_DONT_CARE`
    - `GL_NICEST`
    - `GL_FASTEST`

Generating Texture Coordinates

- GL can generate texture coordinates automatically

  `glTexGen{ifd}[v]()`

- specify a plane
  - generate texture coordinates based upon distance from the plane
- generation modes
  - **`GL_OBJECT_LINEAR`**
  - **`GL_EYE_LINEAR`**
  - **`GL_SPHERE_MAP`** **(used for environmental maps)**

Environment Maps

- Use automatic texture coordinate generation

```
glTexGen[if](GL_S,GL_TEXTURE_GEN_MODE,
   GL_SPHERE_MAP);
glTexGen[if](GL_T,GL_TEXTURE_GEN_MODE,
   GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

Texture Objects

- Texture is part of the OpenGL state
  - If we have different textures for different objects, OpenGL will be moving large amounts data from processor memory to texture memory
- Recent versions of OpenGL have *texture objects*
  - one image per texture object
  - Texture memory can hold multiple texture objects

Applying Textures II

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