Texture Mapping in WebGL

From: Ed Angel
University of New Mexico

Objectives

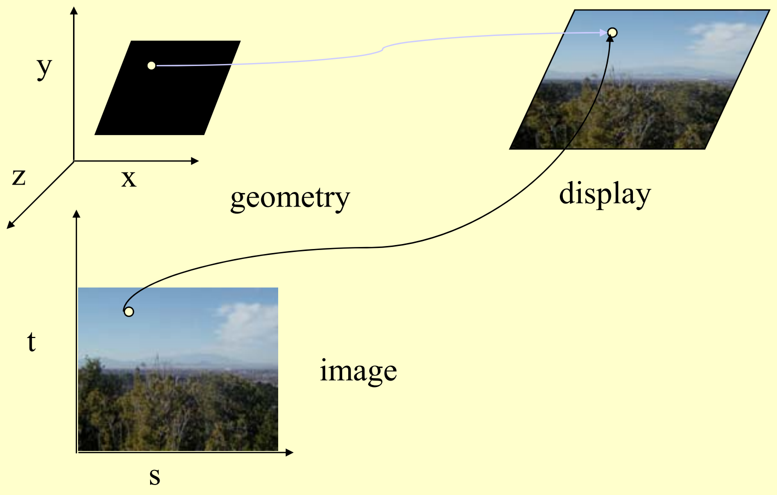
- Introduce WebGL texture mapping
  - two-dimensional texture maps
  - assigning texture coordinates
  - forming texture images
- Introduce the WebGL texture functions and options
  - texture objects
  - texture parameters
  - example code

Basic Stragegy

Three steps to applying a texture

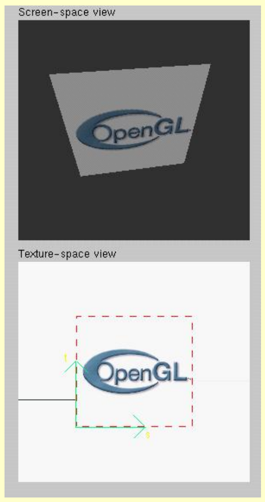
1. specify the texture
   - read or generate image
   - assign to texture
2. specify texture parameters
   - wrapping, filtering
3. assign texture coordinates to vertices
   - Proper mapping function is left to application

Texture Mapping



Texture Example (Nate's tutorial)

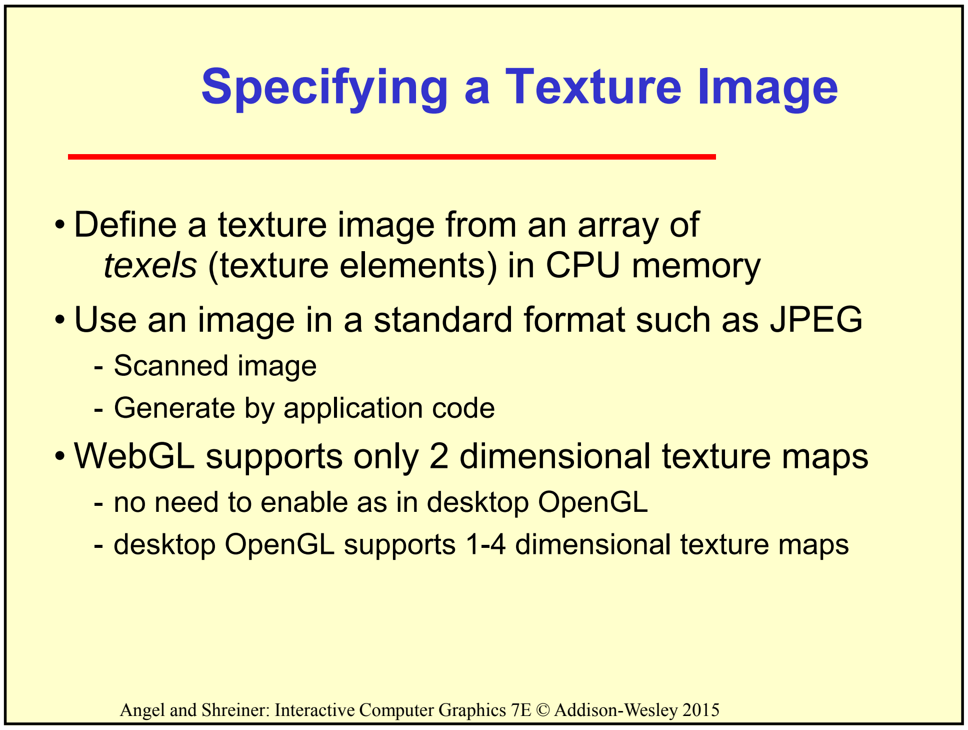
- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



Texture Mapping and the WebGL Pipeline

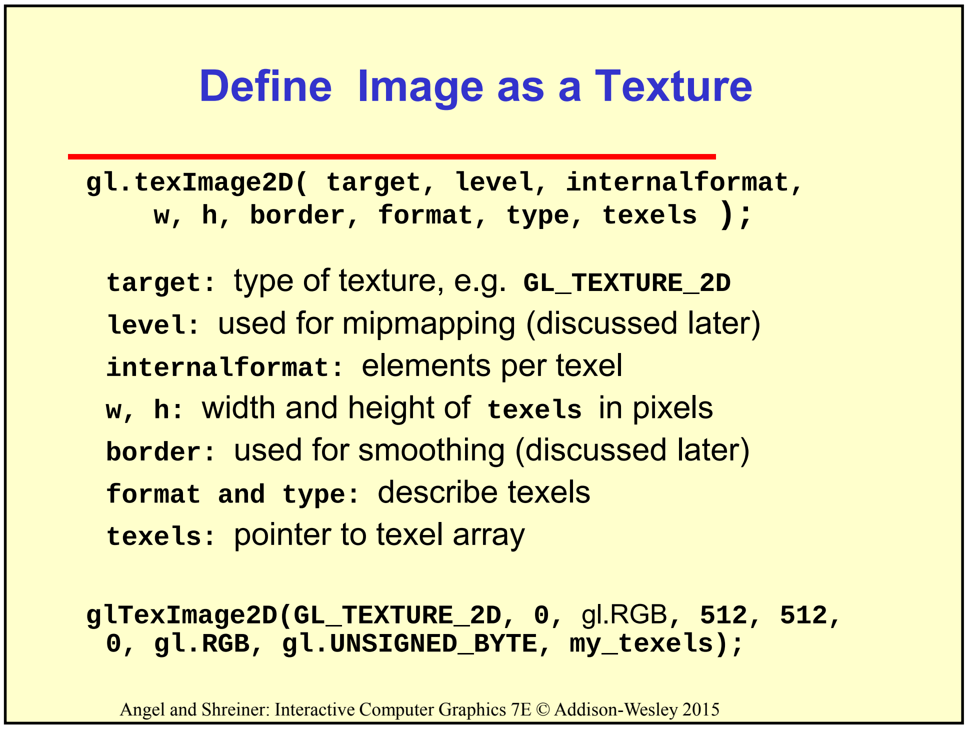
- Images and geometry flow through separate pipelines that join during fragment processing
  - "complex" textures do not affect geometric complexity

$$\begin{tikzcd}
\text{vertices} \arrow[r] & \boxed{\text{geometry pipeline}} \arrow[dr] & \\
& & \boxed{\text{fragment processor}} \\
\text{image} \arrow[r] & \boxed{\text{texel pipeline}} \arrow[ur] &
\end{tikzcd}$$



Specifying a Texture Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory
- Use an image in a standard format such as JPEG
  - Scanned image
  - Generate by application code
- WebGL supports only 2 dimensional texture maps
  - no need to enable as in desktop OpenGL
  - desktop OpenGL supports 1-4 dimensional texture maps



Define Image as a Texture

```
gl.texImage2D( target, level, internalformat,
    w, h, border, format, type, texels );
```

`target:` type of texture, e.g. **`GL_TEXTURE_2D`**
`level:` used for mipmapping (discussed later)
`internalformat:` 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, gl.RGB, 512, 512,
 0, gl.RGB, gl.UNSIGNED_BYTE, my_texels);
```

Define Image as a Texture

```
gl.texImage2D( target, level, internalformat,
    w, h, border, format, type, texels );
```

target: type of texture, e.g. **gl.TEXTURE_2D**
level: used for mipmapping (discussed later)
internalformat: 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, gl.RGB, 512, 512,
 0, gl.RGB, gl.UNSIGNED_BYTE, my_texels);
```

A Checkerboard Image

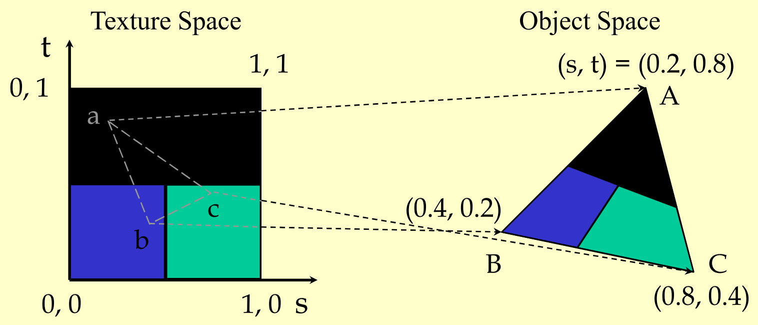
```
var image1 = new Uint8Array(4*texSize*texSize);
   for ( var i = 0; i < texSize; i++ ) {
      for ( var j = 0; j <texSize; j++ ) {
         var patchx = Math.floor(i/(texSize/numChecks));
         var patchy = Math.floor(j/(texSize/numChecks));
         if(patchx%2 ^ patchy%2) c = 255;
         else c = 0;
         //c = 255*(((i & 0x8) == 0) ^ ((j & 0x8)  == 0))
         image1[4*i*texSize+4*j] = c;
         image1[4*i*texSize+4*j+1] = c;
         image1[4*i*texSize+4*j+2] = c;
         image1[4*i*texSize+4*j+3] = 255;
      }
   }
```

Using a GIF image (textureCube1)

```
// specify image in JS file

var image = new Image();
    image.onload = function() {
        configureTexture( image );
    }
    image.src = "SA2011_black.gif”

// or specify image in HTML file with <img> tag

// <img id = "texImage" src = "SA2011_black.gif"></img>

var image = document.getElementById("texImage”)     window.onload =
configureTexture( image );
```

Mapping a Texture

- Based on parametric texture coordinates
- Specify as a 2D vertex attribute



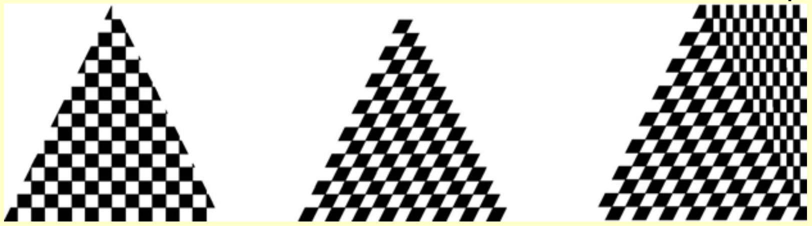
Cube Example

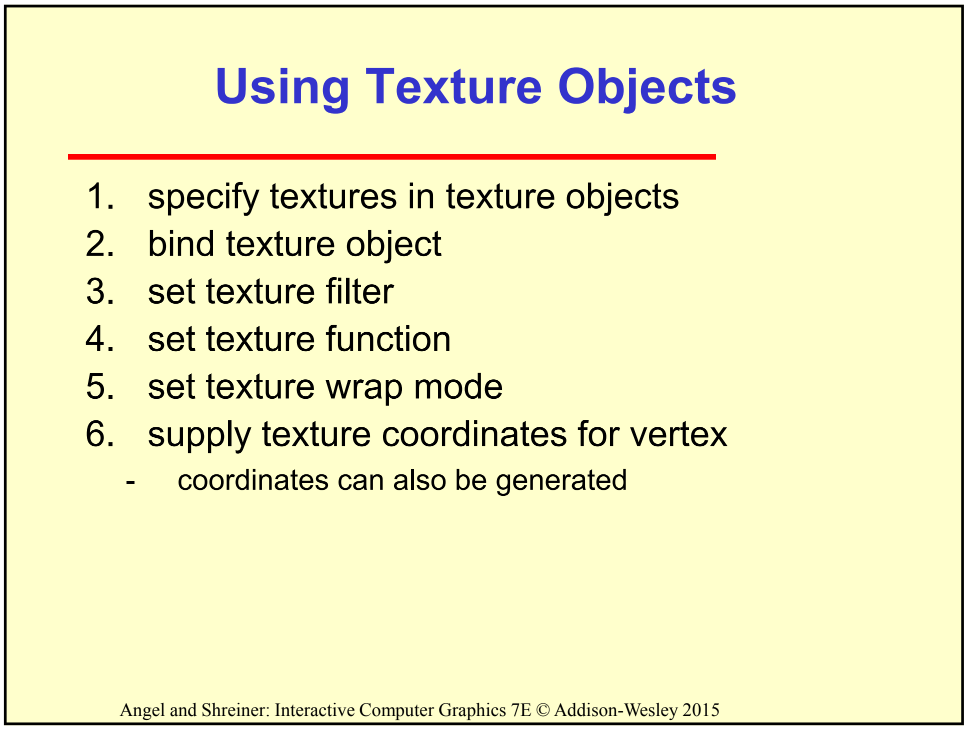
```
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];

function quad(a, b, c, d) {
     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[b]);
      colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[1]);
// etc
```

Interpolation

WebGL uses interpolation to find proper texels from specified texture coordinates

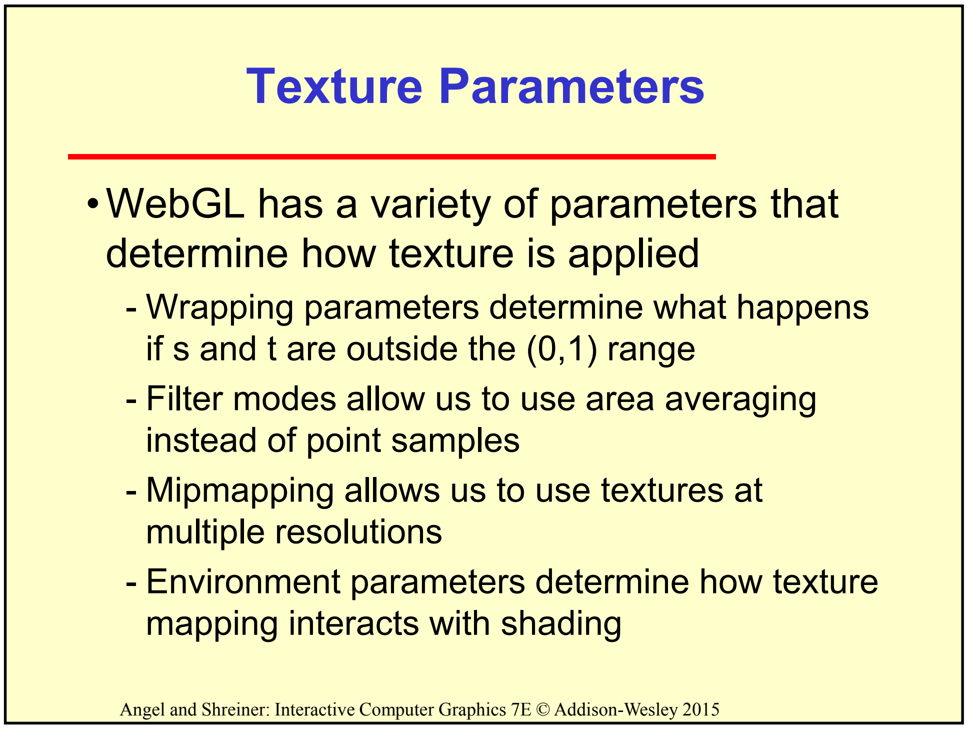
Can be distortions


good selection of tex coordinates
poor selection of tex coordinates
texture stretched over trapezoid showing effects of bilinear interpolation



Using Texture Objects

1. specify textures in texture objects
2. bind texture object
3. set texture filter
4. set texture function
5. set texture wrap mode
6. supply texture coordinates for vertex
   - coordinates can also be generated



Texture Parameters

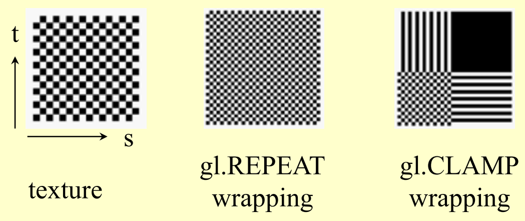
- WebGL has a variety of parameters that determine how texture is applied
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
  -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

Wrapping Mode (Nate's tutorial, textureCube2.js)

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

Wrapping: use s,t modulo 1

```
gl.texParameteri(gl.TEXTURE_2D,
     gl.TEXTURE_WRAP_S, gl.CLAMP )
gl.texParameteri( gl.TEXTURE_2D,
     gl.TEXTURE_WRAP_T, gl.REPEAT )
```

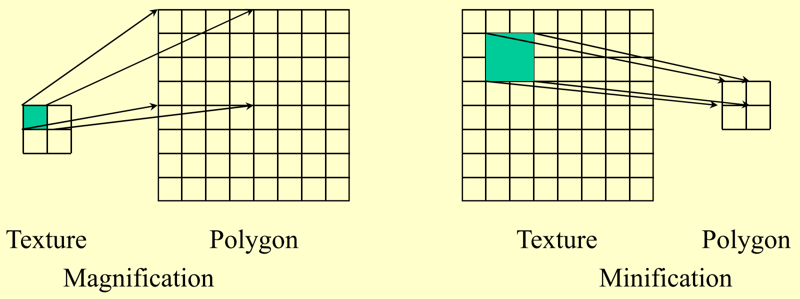


Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015

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



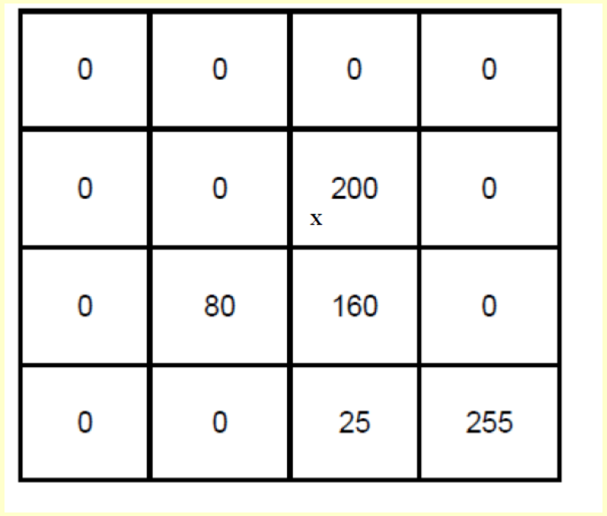
Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015

Filter Modes (Nate's tutorial)

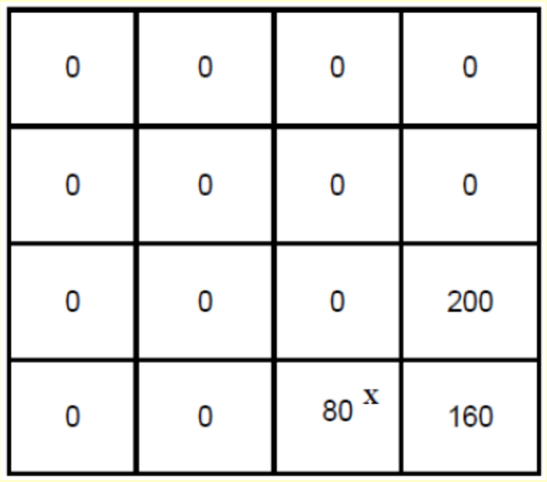
Modes determined by

```
gl.texParameteri( target, type, mode )
```

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MAG_FILTER,
                 gl.NEAREST);

gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MIN_FILTER,
                gl.LINEAR);
```



0	0	0	0
0	0	200	0
0	80	160	0
0	0	25	255



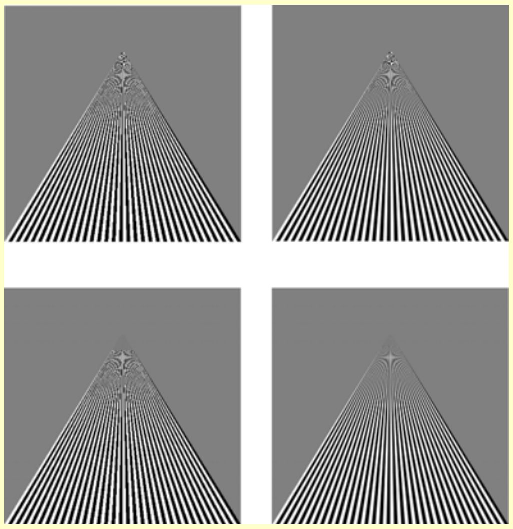
0	0	0	0
0	0	0	0
0	0	0	200
0	0	80 x	160

Mipmapped Textures

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

```
gl.texImage2D(gl.TEXTURE_*D, level, … )
```

Example (textureSquare.html)



Applying Textures (textureCube2.html)

- Texture can be applied in many ways
  - texture fully determines color
  - modulated with a computed color
  - alpha determines non-textured regions (decal)
  - blended with and environmental color
- Fixed function pipeline has a function glTexEnv to set mode
  - deprecated
  - can get all desired functionality via fragment shader
- Can also use multiple texture units

Applying Textures

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

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

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

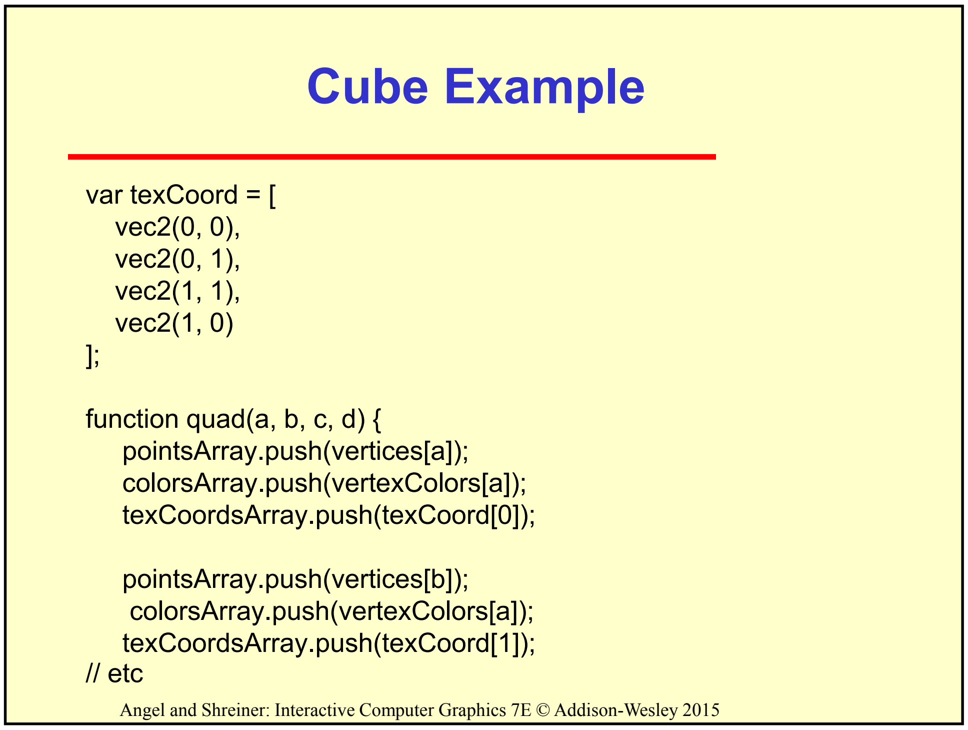
Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015

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

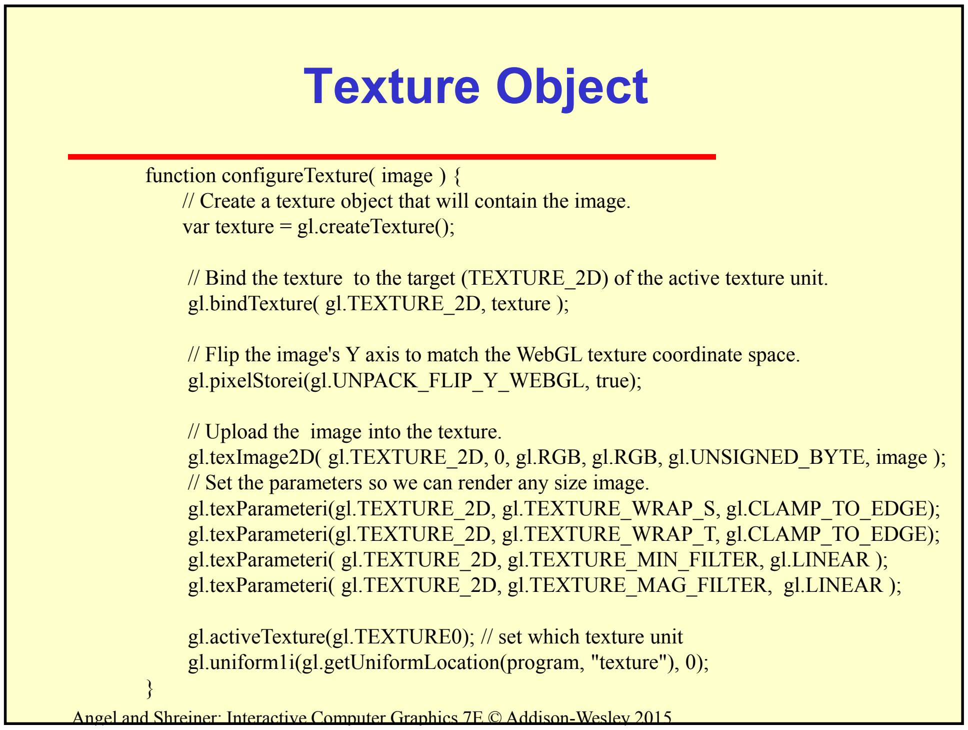
```
attribute vec4 vPosition; //vertex position in object coordinates
attribute vec4 vColor;  //vertex color from application
attribute vec2 vTexCoord; //texture coordinate from application

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

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015



Cube Example

```
var texCoord = [
   vec2(0, 0),
   vec2(0, 1),
   vec2(1, 1),
   vec2(1, 0)
];

function quad(a, b, c, d) {
    pointsArray.push(vertices[a]);
    colorsArray.push(vertexColors[a]);
    texCoordsArray.push(texCoord[0]);

    pointsArray.push(vertices[b]);
     colorsArray.push(vertexColors[a]);
    texCoordsArray.push(texCoord[1]);
// etc
```



Texture Object

```
function configureTexture( image ) {
    // Create a texture object that will contain the image.
    var texture = gl.createTexture();

     // Bind the texture  to the target (TEXTURE_2D) of the active texture unit.
     gl.bindTexture( gl.TEXTURE_2D, texture );

     // Flip the image's Y axis to match the WebGL texture coordinate space.
     gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);

     // Upload the  image into the texture.
     gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image );
     // Set the parameters so we can render any size image.
     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
     gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR );
     gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,  gl.LINEAR );

     gl.activeTexture(gl.TEXTURE0); // set which texture unit
     gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
}
```

Linking with Shaders

```
var vTexCoord = gl.getAttribLocation( program, "vTexCoord" );
gl.enableVertexAttribArray( vTexCoord );
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0);

// 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
//   gl.activeTexture().

gl.uniform1i( glGetUniformLocation(program, "texture"), 0 );
```

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 cube map
  - Alternative is to use a spherical map
- Multitexturing
  - Apply a sequence of textures through cascaded texture units