Texture Mapping II
Step 1: Reserve names for texture

`glGenTextures(GLSizei n, GLUint *textures);`

- requests `n` unique ids
- stores them in *textures*
Step 2: Bind Textures & Create Texture Objects

`glBindTexture(Glenum target, Gluint texture)`

- **target**: `GL_TEXTURE_2D`
- **texture**: id returned from Gen Textures

Texture object is created
Binds it to the currently active texture unit
All subsequent texture calls apply to this active texture
**Texture Units**

Bind texture to context (OGL State) via texture units

```gl
GL_TEXTURE0..GL_TEXTUREi
```

Once bound, it can be accessed in shaders via sampler variables

Current cards must support up to at least 80 texture units
Step 3: Specifying a Texture Image

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

```c
GLubyte my_texels[512][512];
```

Can be....

- Scanned image
- Generate by application code
Step 4: Specify 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
Wrapping Mode

```c
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE )
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT )
```

open.gl/textures
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

http://www.glprogramming.com/red/chapter09.html
Filter Modes

`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`

`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);`

`open.gl/textures`
Mipmapped Textures

Mipmapping allows for prefILTERED texture maps of decreasing resolutions
Reduces aliasing
Lessens interpolation errors for smaller textured objects

glGenerateMipmap(GL_TEXTURE_2D);
Example

- point sampling
- mipmapped point sampling
- linear filtering
- mipmapped linear filtering
Step 3: Load the Texture

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

- `target`: type of texture, e.g. `GL_TEXTURE_2D`
- `level`: used for mipmapping
- `components`: elements per texel
- `w, h`: width and height of `texels` in pixels
- `border`: no longer used so **must set to 0**
- `format` and `type`: describe `texels`
- `texels`: pointer to texel array

```c
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```
Step 4: Texture Coordinates

1. Add to the VBO as another SubBuffer component
2. Link to the shader

Glfloat tex_coord[N][2];
...
glBufferSubData(GL_ARRAY_BUFFER, offset,
    sizeof(tex_coords), tex_coord);
...
    /* Tie to shader */
Gluint loc3;
loc3 = glGetAttribLocation(program, “texCoord”);
glEnableVertexAttribArray(loc3);
 glVertexAttribPointer(loc3, 2, GL_FLOAT, GL_FALSE, 0, tex_coord);
...
Vertex Shader - Texturing

Simply pass along the texture coordinates so they can be interpolated

```glsl
in vec2 texcoord;
out vec2 st
st = texcoord
```
Fragment Shader - Textures

Sampler : a variable that provides access to the texture object

In the app, link texture object to sampler, “texMap”

```c
GLuint tex_loc;
tex_loc = glGetUniformLocation(program, "texMap");
glUniform1i(tex_loc, 0); //use default texture unit 0
```
In the FragmentShader, use the sampler to access the texture object

```cpp
in vec2 st;       //from the vertex shader..interpolated
in vec4 color;    //from the vertex shader, interpolated
uniform sampler2D texMap;

void Main()
{
  gl_FragColor = color * texture(texMap, st);
}
```
Texture Functions – Deprecated

Controls how texture is applied

\[
\text{glTexEnv}\{fi\}[v](\text{GL\_TEXTURE\_ENV}, \text{prop}, \text{param})
\]

\text{GL\_TEXTURE\_ENV\_MODE} modes

- \text{GL\_MODULATE}: modulates with computed shade
- \text{GL\_BLEND}: blends with an environmental color
- \text{GL\_REPLACE}: use only texture color

\text{GL(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE)};

We now have access to texture and fragment in the Fragshader, so we can implement all of these ourselves! (and more!)
GL_TEXTURE_ENV_MODE: Replace

GL_REPLACE: use only texture color (ie. Soup can label)

\[ Cv = Ct \]
GL_TEXTURE_ENV_MODE: Decal

GL_DECAL: blends with fragment color using Alpha of the texture as the blend parameter

put an alpha channeled texture like a logo onto an object

\[ C_v = (1 - A_t) C_f + A_t C_t \]

\[ A_v = A_f \]
GL_TEXTURE_ENV_MODE: Modulate

**GL_MODULATE**: modulates or scales the fragment color by the texture color (or luminance, alpha, etc. depending on chosen internal format)

Good for combining texturing and shading. Use white specular polygons to render lit textured objects where the texture provides the diffuse color) – mult texture by shade

\[
C_v = C_t \cdot C_f \\
A_v = A_t \cdot A_f
\]