Texture Mapping II
Specify a Texture Image

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

\[
\text{Glubyte my\_texels[512][512];}
\]

Can be....

- Scanned image
- Generate by application code procedurally
- Rendered into a texture
Step 1: Reserve names for texture

```
int 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 started
All subsequent texture calls apply to this active texture and form the Texture Object
Step 3: 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

- **Mipmapming** allows us to use textures at multiple resolutions
Wrapping Mode

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

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

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

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** (bilinear interpolation) to obtain texture values.

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

```c
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

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

![GL_NEAREST](open.gl/textures)

![GL_LINEAR](open.gl/textures)
Mipmapped Textures for Minification

*Mipmapping* allows for prefiltered texture maps of decreasing resolutions

Reduces aliasing

Lessens interpolation errors for smaller textured objects

```gl
glGenerateMipmap(GL_TEXTURE_2D);
```
Example: Minification 4 ways

GL_NEAREST

GL_LINEAR

GL_NEAREST_MIPMAP_NEAREST

GL_LINEAR_MIPMAP_LINEAR
Step 3: Attach Texture Image to Texture Object

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

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

```c
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```
Applying a Texture to Geometry

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

```c
GLfloat tex_coord[N][2];
...
glBufferSubData(GL_ARRAY_BUFFER, sizeof(verts)+sizeof(normals),
                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 – Alterned for Texturing

Simply pass along the texture coordinates so they can be interpolated

```glsl
in vec2 texcoord;
out vec2 st
st = texcoord
```
In the Fragment Shader, use the built-in type, `sampler`, and built-in function, `texture`, to access the texture object.

```glsl
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);
}
```
Drawing

```c
glBindTexture(GL_TEXTURE_2D, texture);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);
```

Automatically assigns the texture to the sampler
We can assign a location to a Sampler with glUniform1i by linking *texture object* to *sampler*, “texMap”

```c
Gluint tex_loc;
tex_loc = glGetUniformLocation(program, “texMap”);
glUniform1i(tex_loc, 0); //use default texture unit 0
```
Texture Units

We can assign a location to a Sampler with `glUniform1i`

```c
Gluint tex_loc;
tex_loc = glGetUniformLocation(program, "texMap");
glUniform1i(tex_loc, 0); //use default texture unit 0
```

This location is called the **Texture Unit**

Allows us to use multiple textures in the Fragment Shader
Using Multiple Textures

Activate the texture Unit
Bind the Texture

```c
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture1);
    glUniform1i(glGetUniformLocation(ourShader.Program, "mySampler1"), 0);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, texture2);
    glUniform1i(glGetUniformLocation(ourShader.Program, "mySampler2"), 1);

    glBindVertexArray(VAO);
    glDrawElements(...);
    glBindVertexArray(0);
```
Texture Functions – Deprecated

Controls how texture is applied

```c
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

```c
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

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

void Main()
{
    gl_FragColor = texture(texMap, st);
}
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

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

\[
A_v = A_f
\]

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

void Main()
{
vec4 tcolor = texture(texMap, st);


gl_FragColor = (1-tcolor[3]) * color + tcolor[3] * tcolor);
}
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

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

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

void Main()
{
  vec4 tcolor = texture(texMap, st);
  gl_FragColor = tcolor * color; //do each component
}