Creating shaders in OpenGL

- Sequence of OpenGL API calls to load, compile, link, activate shaders
- Input is a string that contains shader program
  - String usually read from file
  - Separate files for fragment and vertex shaders

Per-pixel diffuse lighting

```c
// Vertex shader, stored in file diffuse.vert
varying vec3 normal, lightDir;
void main()
{
    lightDir = normalize(vec3(gl_LightSource[0].position));
    normal = normalize(gl_NormalMatrix * gl_Normal);
    gl_Position = ftransform();
}
```

```c
// Pixel shader, stored in file diffuse.frag
varying vec3 normal, lightDir;
void main()
{
    gl_FragColor = gl_LightSource[0].diffuse * 
    max(dot(normalize(normal), normalize(lightDir)), 0.0) * 
    gl_FrontMaterial.diffuse;
}
```

Today

Texturing
- Basics
- Texture coordinate assignment
- Antialiasing
- Fancy textures

Large triangles

Pros
- Often ok for simple geometry
- Fast to render

Cons
- Per vertex colors look bad
- Need more interesting surfaces
  - Detailed color variation, small scale bumps, roughness
- Ideas?
Texture mapping

- Glue textures (images) onto surfaces
- Same triangles, much more interesting appearance
- Think of colors as reflectance coefficients

Textures

- Photographs
- Paint directly on surfaces using modeling program
- Stored as image files

Images by Paul Debevec

Texture painting in Maya

Texture mapping

- Goal: assign locations in texture to locations on 3D geometry
- Introduce texture space
  - Texture pixels (texels) have texture coordinates \((u,v)\)
  - Convention
    - Bottom left corner of texture is \((u,v) = (0,0)\)
    - Top right corner is \((u,v) = (1,1)\)

Texture mapping

- Each point on triangle has barycentric \(\alpha, \beta, \gamma\) coordinates
- Use barycentric coordinates to interpolate texture coordinates
**Rendering**
- Given
  - Texture coordinates at each vertex
  - Texture image
- At each pixel, use barycentric coordinates to interpolate texture coordinates
- Look up corresponding texel
- Paint current pixel with texel color
- All computations on GPU

**Texture mapping**
- Includes texture mapping

**Perspective correct interpolation**
- Point in image space with barycentric coordinates \( \alpha, \beta, \gamma \)
- Triangle vertices with homogeneous coordinates \( a_w, b_w, c_w \)
  1. \( \frac{1}{w} = \alpha \frac{1}{a_w} + \beta \frac{1}{b_w} + \gamma \frac{1}{c_w} \)
  2. \( \frac{u}{w} = \alpha \frac{u}{a_w} + \beta \frac{u}{b_w} + \gamma \frac{u}{c_w} \)
  3. \( u = \frac{u}{w} \frac{1}{w} \)
- Same for \( v \) texture coordinate

**Texture look-up**
- Given interpolated texture coordinates \((u, v)\) at current pixel
- Closest four texels in texture space are at \((u_0, v_0), (u_1, v_0), (u_1, v_1), (u_0, v_1)\)
- How to compute color of pixel?

**Nearest-neighbor interpolation**
- Use color of closest texel
- Simple, but low quality
Bilinear interpolation

1. Linear interpolation horizontally

\[ w_u = \frac{u - u_0}{u_1 - u_0} \]
\[ c_0 = tex(u_0, v_0)(1 - w_u) + tex(u_1, v_0)w_u \]
\[ c_1 = tex(u_0, v_1)(1 - w_u) + tex(u_1, v_1)w_u \]

2. Linear interpolation vertically

\[ w_v = \frac{v - v_0}{v_1 - v_0} \]
\[ c_0 = tex(u_0, v_0)(1 - w_v) + tex(u_0, v_1)w_v \]
\[ c_1 = tex(u_1, v_0)(1 - w_v) + tex(u_1, v_1)w_v \]

Questions?

Basic shaders for texturing

```cpp
// Need to initialize texture using OpenGL API calls.
// See base code.

// Vertex shader
void main()
{
    gl_Position = ftransform();
}

// Fragment shader
uniform sampler2D tex;
void main()
{
    gl_FragColor = texture2D(tex, gl_TexCoord[0].st);
}
```

Tiling

- Image exists from [0,0]x[1,1] in texture space
- (u,v) texture coordinates may go outside that range
- **Tiling or wrapping** rules for out-of-range coordinates

Tiling

- Repeat the texture
  - Seams, unless the texture lines up on the edges

![Texture Space]
Clamping

• Use the edge value everywhere outside the data
• Or, ignore the texture outside 0-1

Mirroring

• Flip left-to-right and top-to-bottom
  • All the edges line up

Today

Texturing
• Basics
• Texture coordinate assignment
• Antialiasing
• Fancy textures

Texture coordinate assignment

Surface parameterization
• Mapping between 3D positions on surface and 2D texture coordinates
  • In practice, defined by texture coordinates of triangle vertices
• Various options to establish a parameterization
  • Parametric mapping
  • Orthographic mapping
  • Projective mapping
  • Spherical mapping
  • Cylindrical mapping
  • Skin mapping

Parametric mapping

• Surface given by parametric functions
  \[ x = f(u,v) \quad y = f(u,v) \quad z = f(u,v) \]
• Very common in CAD
• Use \((u,v)\) parameters as texture coordinates

Orthographic mapping

• Use linear transformation of object’s xyz coordinates
• For example
\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  u \\
  v \\
  1 \\
  1
\end{bmatrix}
\]
### Projective mapping
- Use perspective projection of xyz coordinates
  - OpenGL provides GL_TEXTURE matrix to apply perspective projection on texture coordinates
- Useful to achieve “fake” lighting effects

### Spherical mapping
- Use, e.g., spherical coordinates for sphere
- Place object in sphere
- “shrink-wrap” sphere to object

### Cylindrical mapping
- Similar as spherical mapping, but with cylinder
- Useful for faces

### Skin mapping
- Fancy techniques to unfold surface onto plane
- Preserve area, angle
- Sophisticated math

### Today
- Texturing
  - Basics
  - Texture coordinate assignment
  - Antialiasing
  - Fancy textures

### What is going on here?
Antialiasing

Sufficiently sampled, no aliasing

Insufficiently sampled, aliasing

High frequencies in the input appear as low frequencies in the sampled signal.

Antialiasing: intuition

- Pixel may cover large area on triangle in camera space
- Corresponds to many texels in texture space
- Need to compute average

Antialiasing: the math

- Pixels are samples, not little squares
- Use frequency analysis to explain sampling artifacts
  - Fourier transforms
- Antialiasing is achieved through low-pass filtering
  - If you are interested
    - Glassner, Principles of digital image synthesis

Questions?

Antialiasing using mipmaps

- Averaging over texels is expensive
  - Many texels as objects get smaller
  - Large memory access and compuation cost
- Precompute filtered (averaged) textures
  - Mipmaps
- Practical solution to aliasing problem
  - Fast and simple
  - Available in OpenGL, implemented in GPUs
  - Reasonable quality
Mipmaps

• MIP stands for *multum in parvo* (Williams 1983)

**Before rendering**

• Precompute and store scaled down versions of textures
  - Reduce resolution by factor 2 successively
  - Use high quality filtering (averaging) scheme
• Increases memory cost by 1/3
  - $1/3 = \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + ...$
• Width, height of texture needs to be power of two

Mipmaps

• 1 texel in level 4 is an average of $4^4 = 256$ texels in level 0

Rendering with mipmaps

• Mipmapping
• Interpolate texture coordinate of each pixel as before
• Compute approximate size of pixel in texture space
• Look-up color in nearest mipmap
  - E.g., if pixel corresponds to 10x10 texels use mip-map level 3
  - Use nearest neighbor or bilinear interpolation as before

Mipmapping

• Given by partial derivatives of mapping $u(x, y), v(x, y)$

Size of a pixel in texture space

• Given by partial derivatives of mapping $u(x, y), v(x, y)$
<table>
<thead>
<tr>
<th>Nearest mipmap, nearest neighbor</th>
<th>Nearest mipmap, bilinear</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Visible transition between mipmap levels</td>
<td>• Visible transition between mipmap levels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trilinear mipmapping</th>
<th>Trilinear mipmapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Use two nearest mipmap levels</td>
<td>• Smooth transition between mipmap levels</td>
</tr>
<tr>
<td>- E.g., if pixel corresponds to 10x10 texels, use mipmap level 3 and 4</td>
<td></td>
</tr>
<tr>
<td>• Perform bilinear interpolation in both mipmap maps</td>
<td></td>
</tr>
<tr>
<td>• Linearly blend between the results</td>
<td></td>
</tr>
<tr>
<td>• Requires access to 8 texels for each pixel</td>
<td></td>
</tr>
<tr>
<td>• Standard method, supported by hardware with no performance penalty</td>
<td></td>
</tr>
</tbody>
</table>

**Note on OpenGL**

- Distinguishes between minification and magnification
  - Minification: a texel is smaller than a pixel
  - Magnification: a texel is larger than a pixel
  - Minification, magnification may vary across pixels of individual triangles
- OpenGL allows you to specify different interpolation techniques separately

**Questions?**
### Are we satisfied?

| Trilinear mipmapming |

### Mipmapping limitations

- Mipmap texels always represent square areas
- Pixel area is not always square in texture space
- Mipmapping makes trade-off between aliasing and blurriness

| A circular pixel is back-projected to an ellipse |

### Anisotropic texture filtering

- Average texture over elliptical area
  - Higher quality than trilinear mipmap-mapping
  - More expensive
- Anisotropic filtering in hardware
  - Take several bilinear probes approximating the ellipse
  - Reduces rendering performance on current GPUs

| Bilinear probe | Texture space | Ellipse of back-projected pixel |

### Comparison

- Animation

### Today

**Texturing**
- Basics
- Texture coordinate assignment
- Antialiasing
- Fancy textures

### Fancy textures

- Textures most commonly used to modulate ambient and diffuse reflection
  - OpenGL `glTexEnvf(…, GL_MODULATE)`
- Other applications?
**Bump mapping**
- Texture map contains normal perturbations
- No modification of geometry
  - Visible mostly at silhouettes
- Render using per-pixel shading, fragment shader (later in course)

**Displacement mapping**
- Texture map contains local height field
- Modifies geometry
  - Correct silhouettes, shadows
- Requires complicated fragment shader

**Other effects**
**Multi-texturing**
- Several layers of textures for different effects
  - Scratches, dents, rust, ...
  - Illumination textures
**Animated textures**
- Raindrops
- A TV screen, projector in a 3D scene

**Next time**
- Scene management and hierarchies