CS 4204 Computer Graphics

Texture Mapping
Adapted from notes by Yong Cao
Virginia Tech
## The Limits of Geometric Modeling

Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena:

- Clouds
- Grass
- Terrain
- Skin
Modeling an Orange

Consider the problem of modeling an orange
Start with an orange-colored sphere
  • Too simple

Replace sphere with a more complex shape
  • Does not capture surface characteristics (small dimples)
  • Takes too many polygons to model all the dimples
Modeling an Orange (2)

Take a picture of a real orange, scan it, and “paste” onto simple geometric model

- This process is known as texture mapping

Still might not be sufficient because resulting surface will be smooth

- Need to change local shape
- Bump mapping
Three Types of Mapping

**Texture Mapping**
- Uses images to fill inside of polygons

**Environment (reflection mapping)**
- Uses a picture of the environment for texture maps
- Allows simulation of highly specular surfaces

**Bump mapping**
- Emulates altering normal vectors during the rendering process
Texture Mapping

geometric model

texture mapped
Environment Mapping
Bump Mapping
Where does mapping take place?

**Mapping techniques are implemented at the end of the rendering pipeline**

- Very efficient because few polygons make it past the clipper

![Diagram of rendering pipeline](image-url)
Is it simple?

*Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved*
## Coordinate Systems

**Parametric coordinates**
- May be used to model curves and surfaces

**Texture coordinates**
- Used to identify points in the image to be mapped

**Object or World Coordinates**
- Conceptually, where the mapping takes place

**Window Coordinates**
- Where the final image is really produced
Texture Mapping

parametric coordinates

texture coordinates

world coordinates

window coordinates
Mapping Functions

*Basic problem is how to find the maps*

Consider mapping from texture coordinates to a point a surface

*Appears to need three functions*

\[
\begin{align*}
x &= x(s,t) \\
y &= y(s,t) \\
z &= z(s,t)
\end{align*}
\]

*But we really want to go the other way*
Backward Mapping

We really want to go backwards

- Given a pixel, we want to know to which point on an object it corresponds
- Given a point on an object, we want to know to which point in the texture it corresponds

Need a map of the form

\[ s = s(x, y, z) \]
\[ t = t(x, y, z) \]

Such functions are difficult to find in general
Flat mapping

*Easiest case is to map a texture to a flat surface*

*Simply assign appropriate texture coordinates to vertices of the face*

*The texture coordinates specify where the color value should be taken from texture space*

*Interpolation of texture coordinates is done between vertices*
### Flat mapping example

<table>
<thead>
<tr>
<th>Texture Coordinates</th>
<th>A: (0, 0)</th>
<th>B: (1, 0)</th>
<th>C: (1, 1)</th>
<th>D: (0, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: (1, 0)</td>
<td>B: (0, 0)</td>
<td>C: (0, 1)</td>
<td>D: (1, 1)</td>
<td></td>
</tr>
</tbody>
</table>

![Flat mapping example diagram](image)
Two-part mapping

One solution to the mapping problem is to first map the texture to a simple intermediate surface.

Example: map to cylinder
**Cylindrical Mapping**

parametric cylinder

\[
x = r \cos 2\pi u \\
y = r \sin 2\pi u \\
z = v/h
\]

maps rectangle in \( u,v \) space to cylinder of radius \( r \) and height \( h \) in world coordinates

\[
s = u \\
t = v
\]

maps from texture space
Spherical Map

*We can use a parametric sphere*

\[
\begin{align*}
x &= r \cos 2\pi u \\
y &= r \sin 2\pi u \cos 2\pi v \\
z &= r \sin 2\pi u \sin 2\pi v
\end{align*}
\]

in a similar manner to the cylinder but have to decide where to put the distortion

Spheres are used in environment maps
Box Mapping

*Easy to use with simple orthographic projection*

*Also used in environment maps*
Second Mapping

Map from intermediate object to actual object

- Normals from intermediate to actual
- Normals from actual to intermediate
- Vectors from center of intermediate to actual
Aliasing

Point sampling of the texture can lead to aliasing errors

miss blue stripes

point samples in texture space

point samples in u,v (or x,y,z) space
Area Averaging

A better but slower option is to use area averaging

Note that *preimage* of pixel is curved
Texture mapping in openGL

*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

image

geometry

display

s

t

x

y

z
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 a Texture Image

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

```c
GLuint my_texels[512][512];
```

Define as any other pixel map
- Scanned image
- Generate by application code

Enable texture mapping
- `glEnable(GL_TEXTURE_2D)`
- OpenGL supports 1-4 dimensional texture maps
Define Image as a Texture

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

- **target**: type of texture, e.g. `GL_TEXTURE_2D`
- **level**: used for mipmaping (discussed later)
- **components**: 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

```c
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

```c
    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*

If
**Mapping a Texture**

*Based on parametric texture coordinates*

```c
glTexCoord*(()) `specified at each vertex`
```

![Diagram showing texture mapping](image)

- **Textures Space**: (0, 0) to (1, 1)
- **Object Space**: (0.2, 0.8) and (0.8, 0.4)
- Vertex A maps to texture coordinates (0.4, 0.2)
- Vertex B maps to texture coordinates (0.8, 0.4)
- Vertex C maps to texture coordinates (0.8, 0.4)
Typical Code

```c
glBegin(GL_POLYGON);
  glColor3f(r0, g0, b0); // if no shading used
  glNormal3f(u0, v0, w0); // if shading used
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  glEnd();
```

Note that we can use vertex arrays to increase efficiency
Interpolation

*OpenGL 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
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
• 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

\[
\text{glTexParameteri( } \text{GL_TEXTURE}_2D, \\
\text{GL_TEXTURE_WRAP}_S, \text{GL_CLAMP }) \\
\text{glTexParameteri( } \text{GL_TEXTURE}_2D, \\
\text{GL_TEXTURE_WRAP}_T, \text{GL_REPEAT } )
\]
Wrapping Mode

```c
// Wrapping Mode

glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 3.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(3.0, 3.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(3.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
glEnd();

// Wrapping Mode

glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 3.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(3.0, 3.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(3.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
glEnd();
```

- Wrapping Mode
- GL_REPEAT
- GL_REPEAT
Wrapping Mode

GL_CLAMP
in both directions

GL_REPEAT in one direction
GL_CLAMP in the other
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.
Filter Modes

Modes determined by

- `glTexParameteri(target, type, mode)`

  ```
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_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 pre-filtered texture maps of decreasing resolutions

Lessens interpolation errors for smaller textured objects

Declare mipmap level during texture definition

```c
glTexImage2D( GL_TEXTURE_*D, level, ... )
```

GLU mipmap builder routines will build all the textures from a given image

```c
gluBuild*DMipmaps( ... )
```
Example

point sampling  mipmapped point sampling

mipmapped linear filtering  linear filtering
Texture Functions

Controls how texture is applied

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

GL\_TEXTURE\_ENV\_MODE modes

- GL\_MODULATE: modulates with computed shade
- GL\_BLEND: blends with an environment color
- GL\_REPLACE: use only texture color

\[ \text{glTexEnvi}(\text{GL\_TEXTURE\_ENV}, \text{GL\_TEXTURE\_ENV\_MODE}, \text{GL\_MODULATE}) \]

Set blend color with GL\_TEXTURE\_ENV\_COLOR
Perspective Correction Hint

Texture coordinate and color interpolation

- either linearly in screen space
- or using depth/perspective values (slower)

Noticeable for polygons “on edge”

```c
glHint (GL_PERSPECTIVE_CORRECTION_HINT, hint)
```

where hint is one of

- `GL_DONT_CARE`
- `GL_NICEST`
- `GL_FASTEST`
Generating Texture Coordinates

*OpenGL can generate texture coordinates automatically*

\[
glTexGen\{idf\}[v]()
\]

**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 environment maps)
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
<table>
<thead>
<tr>
<th>Applying Textures II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>specify textures in texture objects</strong></td>
</tr>
<tr>
<td><strong>set texture filter</strong></td>
</tr>
<tr>
<td><strong>set texture function</strong></td>
</tr>
<tr>
<td><strong>set texture wrap mode</strong></td>
</tr>
<tr>
<td><strong>set optional perspective correction hint</strong></td>
</tr>
<tr>
<td><strong>bind texture object</strong></td>
</tr>
<tr>
<td><strong>enable texturing</strong></td>
</tr>
<tr>
<td><strong>supply texture coordinates for vertex</strong></td>
</tr>
<tr>
<td>• coordinates can also be generated</td>
</tr>
</tbody>
</table>