Texture Mapping

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

• Introduce Mapping Methods
  - Texture Mapping
  - Environment Mapping
  - Bump Mapping

• Consider basic strategies
  - Forward vs backward mapping
  - Point sampling vs area averaging
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 (1)

- Consider the problem of modeling an orange (the fruit)
- 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

diagram of geometric model and texture mapped image
Bump Mapping
Texture Mapping
Basic Idea

• 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
- Appear to need three functions
  \[ x = x(s,t) \]
  \[ y = y(s,t) \]
  \[ z = z(s,t) \]
- 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

• Simple examples: cylinder, sphere, box
Cylindrical Mapping

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

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 environmental maps
Box Mapping

• Easy to use with simple orthographic projection
• Also used in environment maps
Second Mapping

- Map from intermediate object (e.g., sphere) to actual object
- Three variations:
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate object
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
Applying Textures

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 Objects (1)

• Have OpenGL store your images
  - one image per texture object
  - may be shared by several graphics contexts

• Generate texture names

```c
glGenTextures( n, *texIds );
```
Texture Objects (2)

• Create texture objects with texture data and state

```c
glBindTexture( target, id );
```

• Bind textures before using

```c
glBindTexture( target, id );
```
Specifying a Texture Image

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

target: type of texture, e.g. `GL_TEXTURE_2D`
level: used for mipmapping (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

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, 
    GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```
Mapping a Texture

- Based on parametric texture coordinates
- Coordinates need to be specified at each vertex
Applying Texture to Cube

// add texture coordinate attribute to quad function
quad( int a, int b, int c, int d )
{
    vColors[Index] = colors[a];
    vPositions[Index] = positions[a];
    vTexCoords[Index] = vec2( 0.0, 0.0 );
    Index++;

    vColors[Index] = colors[b];
    vPositions[Index] = positions[b];
    vTexCoords[Index] = vec2( 1.0, 0.0 );
    Index++;

    … // rest of vertices
}

Create/Read a Texture Image

// Create a checkerboard pattern
for ( int i = 0; i < 64; i++ ) {
    for ( int j = 0; j < 64; j++ ) {
        GLubyte c;
        c = ((i & 0x8 == 0) ^ (j & 0x8 == 0)) * 255;
        image[i][j][0] = c;
        image[i][j][1] = c;
        image[i][j][2] = c;
    }
}
**Texture Object**

```c
GLuint textures[1];
glGenTextures( 1, textures );

glActiveTexture( GL_TEXTURE0 );
glBindTexture( GL_TEXTURE_2D, textures[0] );

glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, TextureSize, TextureSize, GL_RGB, GL_UNSIGNED_BYTE, image );

glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
```
Attribute Variables

offset = 0;
GLuint vPosition = glGetAttribLocation( program, "vPosition" );
glEnableVertexAttribArray( vPosition );
glVertexAttribPointer( vPosition, 4, GL_FLOAT, GL_FALSE,
    0,(void*) offset );

offset += sizeof(points);
GLuint vTexCoord = glGetAttribLocation( program, "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2, GL_FLOAT,
    GL_FALSE, 0, (void *) offset );
Vertex Shader

```glsl
in vec4 vPosition;
in vec4 vColor;
in vec2 vTexCoord;

out vec4 color;
out vec2 texCoord;

void main()
{
    color = vColor;
texCoord = vTexCoord;
    gl_Position = vPosition;
}
```
Fragment Shader

```glsl
in vec4 color;
in vec2 texCoord;
out vec4 fColor;
uniform sampler texture;

void main()
{
    fColor = color * texture( texture, texCoord );
}
```
Fragment Shader for Modulating Intensity with Texture

```glsl
in vec4 texCoord;

// Declare the sampler
uniform float intensity;
uniform sampler2D diffuseMaterialTexture;

// Apply the material color
vec3 diffuse = intensity *
    texture(diffuseMaterialTexture, texCoord).rgb;
```
Interpolation

OpenGL uses interpolation (in rasterizer) 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

```c
GLuint texParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);  
GLuint texParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);  
```

![Texture and Wrapping Modes](image)
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

- `glTexParameterIi( target, type, mode )`

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

```c
glTexParameterIi(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 prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition
  
  ```
  glTexImage2D( GL_TEXTURE_*D, level, … )
  ```
Example

<table>
<thead>
<tr>
<th>Point sampling</th>
<th>Linear filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mipmapped point sampling</td>
<td>Mipmapped linear filtering</td>
</tr>
</tbody>
</table>
Texture Functions

• Controls how texture is applied
  • `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
  - `GL(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)`;

• Set blend color with `GL_TEXTURE_ENV_COLOR`
Using Texture Objects

1. specify textures in texture objects
2. set texture filter
3. set texture function
4. set texture wrap mode
5. set optional perspective correction hint
6. bind texture object
7. enable texturing
8. supply texture coordinates for vertex
   - coordinates can also be generated
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 spherical map
  - Alternative is to use a cube map

• Multitexturing
  - Apply a sequence of textures through cascaded texture units
Applying Textures

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

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

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

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

out vec4 color;       //output color to be interpolated
out vec2 texCoord;    //output tex coordinate to be interpolated
Checkerboard Texture

GLubyte image[64][64][3];

// Create a 64 x 64 checkerboard pattern
for ( int i = 0; i < 64; i++ ) {
    for ( int j = 0; j < 64; j++ ) {
        GLubyte c = (((i & 0x8)==0)^((j & 0x8)==0)) * 255;
        image[i][j][0]  = c;
        image[i][j][1]  = c;
        image[i][j][2]  = c;
void quad( int a, int b, int c, int d )
{
    quad_colors[Index] = colors[a];
    points[Index] = vertices[a];
    tex_coords[Index] = vec2( 0.0, 0.0 );
    Index++;
    quad_colors[Index] = colors[a];
    points[Index] = vertices[b];
    tex_coords[Index] = vec2( 0.0, 1.0 );
    Index++;

    // other vertices
}

Adding Texture Coordinates
Texture Object

```c
GLuint textures[1];
    glGenTextures( 1, textures );

    glBindTexture( GL_TEXTURE_2D, textures[0] );
    glTexImage2D(  GL_TEXTURE_2D, 0, GL_RGB, TextureSize,
                    TextureSize, 0, GL_RGB, GL_UNSIGNED_BYTE, image );
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glActiveTexture(GL_TEXTURE0);
```
Linking with Shaders

GLuint vTexCoord = glGetAttribLocation(program, "vTexCoord");
glEnableVertexAttribArray(vTexCoord);
glVertexAttribPointer(vTexCoord, 2, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(offset));

// 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
// glActiveTexture().
glUniform1i(glGetUniformLocation(program, "texture"), 0);