Comp 410/510
Computer Graphics
Spring 2017

Texture Mapping
The Limits of Geometric Modeling

• Although graphics cards can now render millions of polygons in a second, that number is still insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin
Modeling an Orange

- Consider the problem of modeling an orange (the fruit)
- Start with an orange-colored sphere
  - Too simple
- Difficult to capture surface characteristics (small dimples)
  - Takes too many polygons to model all the dimples
Modeling an Orange

• 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 too smooth for lighting
  - Need to change local shape
  - Use bump mapping
Three Types of Mapping

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

• **Bump mapping**
  - Emulates altering normal vectors during the rendering process (randomly or following a pattern)

• **Environmental (reflection mapping)**
  - Uses a picture of the environment as texture image
  - Allows simulation of highly specular surfaces
Texture mapping: 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** \((u,v)\)
  - Can be used to model curved surfaces
- **Texture coordinates** \((s,t)\)
  - Used to identify points in the image to be mapped
- **Object or World Coordinates** \((x,y,z)\)
  - Conceptually, where the mapping takes place
- **Screen or Window Coordinates** \((x_s,y_s)\)
  - Where the final image is really produced
Coordinate Systems

- parametric coordinates
- texture coordinates
- world coordinates
- screen coordinates
Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point on a surface
- Need three functions
  \[ x = x(s,t) \]
  \[ y = y(s,t) \]
  \[ z = z(s,t) \]
- But we actually want to go the other way

![Diagram illustrating mapping functions from texture coordinates to a point on a surface.]
Backward Mapping

- We actually want to go backwards
  - Given a pixel, we may want to know to which point on an object it corresponds (forward)
  - Given a point on an object, we may want to know to which point in the texture it corresponds (backward)
- Need a map of the form
  \[ s = s(x,y,z) \]
  \[ t = t(x,y,z) \]
- Such functions are difficult to find in general if not given explicitly
Texture Mapping - Example

In this example, mapping function, i.e., texture coordinates \((s,t)\) associated with each vertex, are given explicitly.
How to find mapping automatically if not given explicitly?

- One solution is to use the method, referred to as **two-part mapping**
- First map the texture onto a simple intermediate surface, and then to the actual object surface
- Example: map first to a cylinder
Cylindrical Mapping

Parametric cylinder

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

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

\[
\begin{align*}
s &= u \\
t &= v
\end{align*}
\]

maps from texture space
We could 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 we have to decide where to put the distortion (at the poles of the sphere for instance).

Spheres are used especially in environmental mapping.
Box Mapping

- Can also be used in environmental maps
Second Mapping

Mapping from intermediate object to actual object
  - Use normals from intermediate to actual
  - Use normals from actual to intermediate
  - Use vectors from the center of intermediate
Texture Mapping
Basic Strategy (OpenGL)

Texture mapping requires interaction among the application program, the vertex shader, and the fragment shader.

Three steps to applying a texture
1. Specify the texture
   • read or generate texture image
   • place it as texture object on GPU
2. Assign texture coordinates to vertices (which are then interpolated to fragments)
   • Proper mapping function is left to application
3. Specify texture parameters
   • wrapping, filtering, etc
Texture Example

- The texture (below) is a 256 x 256 image mapped to a rectangular polygon which is viewed in perspective.
Example

Wireframe → Shading → Texture mapping
Based on parametric texture coordinates \((s, t)\)
- Specified at each vertex as a vertex attribute

OpenGL uses bilinear interpolation to find the texture coordinates for the interior points of a polygon.
Where does texture mapping take place?

- Texture mapping is carried out at the end of the rendering pipeline as part of fragment processing
  - More efficient because few polygons make it past the clipper

Image pixels and geometry flow through separate pipelines that join at fragment processor
Texture Object

• First create two-dimensional texture object(s): Generate ids and bind them

```c
GLuint mytex[2];
glGenTextures(2, mytex);
glBindTexture(GL_TEXTURE_2D, mytex[0]);
```

• Subsequent texture functions specify the texture image and its parameters, which become part of the texture object.

• Another call to `glBindTexture` either starts a new texture object, or switches to another existing texture object.

• OpenGL supports 1-3 dimensional texture maps
Texture Image (2D)

• Define a texture image from an array of texels (texture elements)
  
  ```glubyte my_texels[512][512][3];```

• Specify that this array is to be used as a two-dimensional texture after a call to `glBindTexture`
  
  ```glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, my_texels);```

Function prototype:

```glTexImage2D(GLenum target, GLint level, GLint iformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, GLvoid *tarray)```
How to Specify an Image as Texture?

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

target: type of texture, e.g., GL_TEXTURE_2D
level: used for mipmapping (will be discussed soon)
components: elements per texel
w, h: width and height of texels in pixels
border: must be zero (no longer used)
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);
Interpolation

Bilinear interpolation may cause distortions when texture is mapped onto a triangle:

Texture image  good selection of texture coordinates  poor selection of texture coordinates
Texture Parameters

- OpenGL has a variety of parameters that determine how texture is applied
  - **Wrapping** parameters determine what happens when texture coordinates $s$ and $t$ are outside the $[0,1]$ range
  - **Filtering** modes allow us to use area averaging instead of point samples
  - **Mipmapping** allows us to use textures at multiple resolutions
Wrapping Mode

Determines what to do when texture coordinates are out of the range \([0,1]\):

**Clamping:** if \( s, t > 1 \) use 1, if \( s, t < 0 \) use 0

**Wrapping (repeat):** use \( s, t \) modulo 1

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

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

texture

GL_REPEAT wrapping

GL_CLAMP clamping
Aliasing

Point sampling of the texture can lead to aliasing errors
Area Averaging

A better but slower option is to use area averaging.

Note that the preimage of a pixel is curved.
OpenGL - Texture Aliasing

Let’s see how OpenGL handles this problem....
Magnification and Minification Problems (OpenGL)

More than one texel may cover a pixel (minification) or more than one pixel may cover a texel (magnification)

Can use point sampling (nearest texel) or linear filtering (2 × 2 filter) to obtain texture values

Magnification, e.g., when zooming in

Minification, e.g., when zooming out
Filter Modes

Modes determined by

- `glTexParameter( target, type, mode )`

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

glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```
Mipmapped Textures

- **Mipmapping** allows for prefILTERED texture maps of decreasing resolutions
- Another way to deal with minification problem

- OpenGL can create a series of texture arrays at reduced sizes, and then automatically uses the appropriate size. For a 64 × 64 original array, we can set up 32 × 32, 16 × 16, 8 × 8, 4 × 4, 2 × 2, and 1× 1 arrays for the current texture object by executing the function call:

  ```cpp
  glGenerateMipmap(GL_TEXTURE_2D);
  ```

- **These mipmaps are invoked automatically by**

  ```cpp
  glTexParameterIi(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
  ```

Mipmapped texture images
Mipmapped Textures

Alternatively, we can declare mipmap level during texture definition:

```c
glTexImage2D(GL_TEXTURE_2D, 1, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```
Example

point sampling

mipmapped point sampling

linear filtering

mipmapped linear filtering
Example

point sampling

mipmapped linear filtering
Applying Textures

- Textures are applied during fragment processing by using a sampler function
- Sampler function returns a texture color from a texture object

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

void main() {
    gl_FragColor = texture2D( texture, texCoord );
}
```
Generating Texture Coordinates

- We have assumed so far that texture coordinates are explicitly provided with data. If not, application should generate them. There are various ways of doing that.
- One option: Specify a plane, and then generate texture coordinates based upon distance from the plane.
- A 1D texture mapping example based on plane distance:

1D texture:

See Ch. 7.7.6 in the textbook

- Another option: Use the two-part mapping approach described previously.