Texture Mapping
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 (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 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

- **Environmental Mapping**
  - Uses a picture of the environment for texture maps

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

geometric model

texture mapped
Environment Mapping

- Mapping environmental information as textures to the geometry
Bump Mapping

• Perturbing normals for vertices and then apply shading (Phong) model
Where does mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons pass down the geometric pipeline
Is it simple?

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

2D image

3D surface
Coordinate Systems

- Parametric coordinates
  - May be used to model curved surfaces
- Texture coordinates
  - Used to identify points in the image to be mapped
- World Coordinates
  - Conceptually, where the mapping takes place
- Screen Coordinates
  - Where the final image is really produced
Texture Mapping

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
• 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: “parameterization” problem
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
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.
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environmental maps
Second Mapping

• Map from intermediate object to actual object
• Three possible strategies
  – Normals from intermediate to actual
  – Normals from actual to intermediate
  – Vectors from center of intermediate
Cube Mapping Texturing

- Take 6 photos surrounding you, attach them to a cube
Cube Mapping

• Texture mapping that uses a 3D direction vector to index into a texture that is six square 2D textures arranged like the faces of a cube
• Use texture mapping technique to simplify ray tracing process
Cube Mapping Example

- hardware-accelerated cube map texturing, run in real-time!
- a physics model to distort the bubble's shape in real-time
- The undulating bubble's surface results in a dynamic reflection of the patio environment
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

texture mapping involves mapping a 2D image onto a 3D geometry. The image is projected onto the screen, with coordinates s and t corresponding to the image, and coordinates x, y, and z corresponding to the geometry.
Texture Example

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

vertices → geometry pipeline → rasterizer

image → pixel pipeline

• The OpenGL pipeline includes two separate pipelines: geometry and pixel pipelines, which join at the rasterizer.
• "Complex" textures do not affect geometric complexity.
Specify Texture Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory
  
  ```glubyte my_texels[512][512];```

- Define as any other pixel map
  - Scan images
  - Via 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 mipmapping
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
  • 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
Mapping a Texture

- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex
Typical Code

```c
glBegin(GL_POLYGON);
    glColor3f(r0, g0, b0);
    glNormal3f(u0, v0, w0);
    glTexCoord2f(s0, t0);
    glVertex3f(x0, y0, z0);
    glColor3f(r1, g1, b1);
    glNormal3f(u1, v1, w1);
    glTexCoord2f(s1, t1);
    glVertex3f(x1, y1, z1);
    ...;
    glVertex3f(x1, y1, z1);
    glEnd();
```

Note that we can use vertex arrays to increase efficiency
Interpolation

OpenGL uses bilinear interpolation to find proper texels from specified texture coordinates

Can have distortions

- good selection of tex coordinates
- poor selection of tex coordinates
- textures stretched over trapezoid
Texture Parameters

- OpenGL: a variety of parameter 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

Repeat in s and t

Clamp in s and t

Clamp in s, Repeat in t
Magnification and Minification

Texel smaller than pixel – more than one texel can cover a pixel (*minification*)
Texel larger than pixel – more than one pixel can cover a texel (*magnification*)
Can use point sampling (nearest texel - fastest) or linear filtering (2 x 2 filter – less aliasing) to obtain texture values
Filter Modes

Modes determined by

- glTexParameteri( target, type, mode )

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

```c
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 prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

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

point sampling

mipmapped point sampling

linear filtering

mipmapped linear filtering