Texture Mapping

- A way of adding surface details

- Two ways can achieve the goal:
  - Model the surface with more polygons
    - Slows down rendering speed
    - Hard to model fine features
  - Map a texture to the surface
    - This lecture
    - Image complexity does not affect complexity of processing

- Efficiently supported in hardware

Trompe L’Oeil (“Deceive the Eye”)

- Windows and columns in the dome are painted, not a real 3D object
- Similar idea with texture mapping:
  Rather than modeling the intricate 3D geometry, replace it with an image!

The texture

- Texture is a bitmap image
  - Can use an image library to load image into memory
  - Or can create images yourself within the program

- 2D array:
  unsigned char texture[height][width][4]

- Or unrolled into 1D array:
  unsigned char texture[4*height*width]

- Pixels of the texture are called texels
- Texel coordinates (s, t) scaled to [0, 1] range
Texture map

Texture coordinates

The “st” coordinate system

Specifying texture coordinates in OpenGL (core profile)

Specifying texture coordinates in OpenGL (compatibility profile)
What if texture coordinates are outside of [0,1]?

Solution 1: Repeat texture

\[
\begin{array}{c}
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)} \\
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)}
\end{array}
\]

Solution 2: Clamp to [0,1]

\[
\begin{array}{c}
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)} \\
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE)}
\end{array}
\]

Combining texture mapping and shading

Outline

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL

Texture interpolation

\[
\begin{array}{c}
\text{5 x 5 texture} \\
(0,0), (0.25,0), (0.5,0), (0.75,0), (1,0)
\end{array}
\]
Texture interpolation

- (s,t) coordinates typically not directly at pixel in the texture, but in between
- Solutions:
  - Use the nearest neighbor to determine color
    - Faster, but worse quality
    - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);`
  - Linear interpolation
    - Incorporate colors of several neighbors to determine color
    - Slower, better quality
    - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)`

Filtering

- Texture image is shrunk in distant parts of the image
- This leads to aliasing
- Can be fixed with filtering
  - bilinear in space
  - trilinear in space and level of detail (mipmapping)

Mipmapping

- Pre-calculate how the texture should look at various distances, then use the appropriate texture at each distance
- Reduces / fixes the aliasing problem

Mipmapping in OpenGL

- Generate mipmaps automatically (for the currently bound texture):
  - Core profile: `glGenerateMipmap(GL_TEXTURE_2D);`
  - Compatibility profile: `gluBuild2DMipmaps(GL_TEXTURE_2D, components, width, height, format, type, data)`
- Must also instruct OpenGL to use mipmaps:
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)`

Outline

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL
Textures do not have to represent color

- Specularity (patches of shininess)
- Transparency (patches of clearness)
- Normal vector changes (bump maps)
- Reflected light (environment maps)
- Shadows
- Changes in surface height (displacement maps)

Bump mapping

- How do you make a surface look rough?
  - Option 1: model the surface with many small polygons
  - Option 2: perturb the normal vectors before the shading calculation
    » Fakes small displacements above or below the true surface
    » The surface doesn’t actually change, but shading makes it look like there are irregularities!
    » A texture stores information about the “fake” height of the surface

Bump mapping

- We can perturb the normal vector without having to make any actual change to the shape.
- This illusion can be seen through—how?

Outline

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL

OpenGL Texture Mapping (Core Profile)

- During initialization:
  1. Read texture image from file into an array in memory, or generate the image using your program
  2. Initialize the texture (glTexImage2D)
  3. Specify texture mapping parameters: Repeat/clamp, filtering, mipmapping, etc.
  4. Make VBO for the texture coordinates
  5. Create VAO
- In display():
  1. Bind VAO
  2. Select the texture unit, and texture (using glBindTexture)
  3. Render (e.g., glDrawArrays)
Read texture image from file into an array in memory

- Can use our ImageIO library

  ```cpp
  ImageIO * imageIO = new ImageIO();
  if (imageIO->loadJPEG(imageFilename) != ImageIO::OK)
    { cout << "Error reading image " << imageFilename << "." << endl;
      exit(EXIT_FAILURE);
    }
  
  See starter code for hw2
  ```

Initializing the texture

- Do once during initialization, for each texture image in the scene, by calling glTexImage2D

  ```cpp
  Texture Initialization
  Global variable:
  GLuint texHandle;

  During initialization:
  // create an integer handle for the texture
  glGenTextures(1, &texHandle);
  int code = initTexture("sky.jpg", texHandle);
  if (code != 0)
    { printf("Error loading the texture image.\n");
      exit(EXIT_FAILURE);
    }

  Function initTexture() is given in the starter code for hw2.
  ```

Texture Shader: Vertex Program

```cpp
// version 150

in vec3 position;  
in vec2 texCoord;  

out vec4 tc;  

void main()
{
  // compute the transformed and projected vertex position (into gl_Position)
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0f);
  // pass-through the texture coordinate
  tc = texCoord;
}
```
Texture Shader: Fragment Program

```glsl
#version 150
in vec2 tc; // input tex coordinates (computed by the interpolator)
out vec4 c; // output color (the final fragment color)
uniform sampler2D textureImage; // the texture image

void main()
{
  // compute the final fragment color,
  // by looking up into the texture map
  c = texture(textureImage, tc);
}
```

Multitexturing

- The ability to use multiple textures simultaneously in a shader
- Useful for bump mapping, displacement mapping, etc.
- The different texture units are denoted by GL_TEXTURE0, GL_TEXTURE1, GL_TEXTURE2, etc.
- In simple applications (our homework), we only need one unit

```c
void setTextureUnit(GLint unit)
{
  glActiveTexture(unit); // select the active texture unit
  // get a handle to the "textureImage" shader variable
  GLuint h_textureImage = glGetUniformLocation(program, "textureImage");
  // deem the shader variable "textureImage" to read from texture unit "unit"
  glUniform1i(h_textureImage, unit - GL_TEXTURE0);
}
```

The display function

```c
void display()
{
  // put all the usual code here (clear screen, set up camera, upload
  // the modelview matrix and projection matrix to GPU, etc.)
  // ...
  setTextureUnit(GL_TEXTURE0); // it is safe to always use GL_TEXTURE0
  // select the texture to use ("texHandle" was generated by glGenTextures)
  glBindTexture(GL_TEXTURE_2D, texHandle);
  // here, bind the VAO and render the object using the VAO (as usual)
  // ...
  glutSwapBuffers();
}
```

Texture mapping in OpenGL (Compatibility Profile)

- During your initialization:
  1. Read texture image from file into an array in memory,
     or generate the image using your program
  2. Specify texture mapping parameters
     » Wrapping, filtering, etc.
  3. Initialize and activate the texture
- In display():
  1. Enable OpenGL texture mapping
  2. Draw objects: Assign texture coordinates to vertices
  3. Disable OpenGL texture mapping

Enable/disable texture mode (Compatibility Profile)

- Must be done before rendering any primitives that are to be texture-mapped
  » glEnable(GL_TEXTURE_2D)
  » glDisable(GL_TEXTURE_2D)
- Successively enable/disable texture mode to switch between drawing textured/non-textured polygons
- Changing textures:
  - Only one texture is active at any given time (with OpenGL extensions, more than one can be used simultaneously; this is called multitexturing)
  - Use glBindTexture to select the active texture
Rendering (compatibility profile)

```c
void display()
{
    ...
    // no modulation of texture color with lighting; use texture color directly
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

    // turn on texture mapping (this disables standard OpenGL lighting, unless in GL_MODULATE mode)
    glEnable(GL_TEXTURE_2D);

    (continues on next page)
```

Rendering (compatibility profile) (part 2)

```c
    glBegin(GL_QUADS); // draw a textured quad
    glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
    glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
    glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
    glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
    glEnd();

    // turn off texture mapping
    glEnable(GL_TEXTURE_2D);

    // draw some non-texture mapped objects (standard OpenGL lighting will be used if it is enabled)
    ...
    // switch back to texture mode, etc.
    ...
} // end display()
```

Summary

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL