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

Texture Mapping + Shading
Filtering and Mipmaps
Non-color Texture Maps
[Angel Ch. 7]

Texture Mapping

• 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!

Jesuit Church, Vienna, Austria

Map textures to surfaces

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

(0,0) texture image (1,0)

(0,1) (1,1)

3D polygon

Texture coordinates

For each pixel, lookup into the texture image to obtain color.

The “st” coordinate system

Note: also called “uv” space

Texture mapping: key slide

Specifying texture coordinates in OpenGL (core profile)

• Use VBO

• Either creates a separate VBO for texture coordinates, or put them with vertex positions into one VBO

What if texture coordinates are outside of [0,1]?
Solution 1: Repeat texture

\[
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)}
\]
\[
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)}
\]

Solution 2: Clamp to [0,1]

\[
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)}
\]
\[
\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE)}
\]

Using this color

\((s,t)\)

Outline

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

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
    - \text{g}l\text{TexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST)};
  - Linear interpolation
    - Incorporate colors of several neighbors to determine color
    - Slower, better quality
    - \text{g}l\text{TexParameteri(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 OpenML

- 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 OpenML

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

Light Mapping

- Quake uses light maps in addition to texture maps. Texture maps are used to add detail to surfaces, and light maps are used to store pre-computed illumination. The two are multiplied together at runtime, and cached for efficiency.

Example: Far Cry 4 (low mapping setting)

Example: Far Cry 4 (high mapping setting)
Example: Far Cry 4 (low mapping setting)

Note the low detail on the walls, due to low-resolution displacement mapping.

Example: Far Cry 4 (high mapping setting)

Note the high detail on the walls, due to high-resolution displacement mapping.

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
- 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
- The dimensions of texture images must be a multiple of 4 (Note: they do NOT have to be a power of 2)
- Can load textures dynamically if GPU memory is scarce:
  Delete a texture (if no longer needed) using glDeleteTextures
Texture Initialization

Global variable:
GLint 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

#define 150

in vec3 position;  // input vertex position
in vec2 texCoord;  // input texture coordinates

out vec2 tc;  // output texture coordinates; they will be passed to
              // the fragment program (interpolated by hardware)

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

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

#define 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);
}

VAO code ("texCoord" shader variable)

During initialization:
glBindVertexArray(vao);  // bind the VAO

// bind the VBO “buffer” (must be previously created)
glBindBuffer(GL_ARRAY_BUFFER, buffer);

// get location index of the “texCoord” shader variable
GLuint loc = glGetUniformLocation(program, "texCoord");

// enable the “texCoord” attribute

// set the layout of the “texCoord” attribute data
const void * offset = (const void*) sizeof(positions);
GLsizei stride = 0;
glVertexAttribPointer(loc, 2, GL_FLOAT, GL_FALSE, stride, offset);

VBO Layout: positions, texture coordinates (for 2 vertices)

VBO

pos1 x y z
pos1 x y z
tc1 u

pos2 x y z
pos2 x y z
tc2 u

in vec3 position
in vec2 texCoord

Texture Initialization

Global variable:
GLint 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.
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 texture unit affected by subsequent texture calls
    // get a handle to the "textureImage" shader variable
    GLint h_textureImage = glGetUniformLocation(program, "textureImage");
    // deem the shader variable "textureImage" to read from texture unit "unit"
    glUniform1i(h_textureImage, unit - GL_TEXTURE0);
}
```

• 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

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.)
    // ...

    // select the active texture unit
    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();
}
```

Summary

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