Texture Mapping

CSE452 - 18

[ Slides courtesy of Eric Shaffer ]
The Limits of Geometric Modeling

- Graphics cards can render over 10 million polygons per second
- That number is still insufficient for many visual phenomena
- Consider rendering a herd of bumpy-skinned dinosaurs
Or consider an orange

- Start with an orange-colored sphere
  - Too simple
- Replace sphere with a more complex shape
  - Requires too many polygons to model all the dimples
Modeling an Orange

- Take a picture of a real orange; “paste” onto simple geometric model
  - This process is known as texture mapping
- Still might be problematic… resulting surface will be smooth
  - Need to change local shading... use bump mapping
Three Types of Mapping

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

- **Environment Mapping**
  - Uses a picture of the environment for texture maps
  - Allows simulation of mirror-like surfaces

- **Bump mapping**
  - Alters normal vectors during the rendering process
  - Generates
Texture Mapping

geometric model

texture mapped
Bump Mapping
Environment Mapping
The Idea is Simple

- Map an image to a surface
- Each fragment samples a color from the image

2D image

3D surface
Texture Mapping

texture coordinates

world coordinates

window coordinates
Cylindrical Mapping

parametric cylinder

\[ x = r \cos 2\pi s \]
\[ y = r \sin 2\pi s \]
\[ z = t/h \]

maps rectangle in t,s space to cylinder of radius r and height h in world coordinates
Basic Steps

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

geometry → display

image
Filtering using Nearest Neighbor

- A \((u,v)\) parametric coordinate mapped to texel coordinates
- e.g. \((0.78, 0.22) \rightarrow (7.8, 2.2)\)
- ...but we have only integer texel coordinates

- Nearest Neighbor picks the closest texel in Manhattan Distance
- e.g. \((0.78, 0.22) \rightarrow (7.8, 2.2) \rightarrow (8,2)\)
Filtering using Bilinear Interpolation

- Bilinear Filtering blends the 4 closest texels
  - weighted by distance
- Example... let $T(s,t)$ be the color at texel $s,t$
- For $(7.8, 2.2)$ we blend $T(8,3), T(8,2), T(7,3), T(7,2)$
Bilinear Interpolation
Bilinear Interpolation

\[ (u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor) \]

\[ b(p_u, p_v) = (1 - u')(1 - v')t(x_l, y_b) \]

\[ + u'(1 - v')t(x_r, y_b) \]

\[ + (1 - u')v't(x_l, y_t) \]

\[ + u'v't(x_r, y_t) \]
Examples

Nearest neighbor filtering  Bilinear Interpolation
Mapping a Texture

- Based on parametric texture coordinates
- Specify as a 2D vertex attribute
A Texture Fetch Simplified

- Seems pretty simple...
- Given
  1. An image
  2. A position
- Return the color of image at position

Fetch at \((u,v) = (0.6, 0.25)\)

RGBA Result is 0.95, 0.4, 0.24, 1.0
Filtering Textures

- Magnification occurs when we have more fragments than texels
- What are two filters we can use to map texels to fragments?
- If we are magnifying a texture, what is the maximum number of texels that must be fetched per fragment?
Filtering Textures

- Magnification occurs when we have more fragments than texels.
- What are two filters we can use to map texels to fragments?
  - Nearest Neighbor
  - Bilinear Filtering
- If we are magnifying a texture, what is the maximum number of texels that must be fetched per fragment?
  - Four for bilinear filtering.
Filtering Textures

- Minification occurs when we have more texels than fragments
- Using NN or Bilinear Filtering can lead to aliasing
- Why?
- What would a better strategy be?
- What is the maximum number of texels fetched per fragment?
Filtering Textures

- Minification occurs when we have more texels than fragments.
- Using NN or Bilinear Filtering can lead to aliasing.
- Why?
  - Sparse sampling will cause us to miss features.
  - E.g., a checkerboard pattern could be turned into solid color.
- What would a better strategy be?
  - Average all of the texels that map into a fragment.
- What is the maximum number of texels fetched per fragment?
  - The entire texture.
Mipmapping is a method of pre-filtering a texture for minification.

- History: 1983 Lance Williams introduced the word “mipmap” in his paper “Pyramidal Parametrics”
- $mip = “multum in parvo”.... latin: many things in small place(?)$

We generate a pyramid of textures:
- Bottom-level is the original texture
- Each subsequent level reduces the resolution by $\frac{1}{4}$ (by $\frac{1}{2}$ along s and t)
Pre-filtered Image Versions

- Base texture image is say 256x256
- Then down-sample 128x128, 64x64, 32x32, all the way down to 1x1

**Trick:** When sampling the texture, pixel the mipmap level with the closest mapping of pixel to texel size.

**Why?** Hardware wants to sample just a small (1 to 8) number of samples for every fetch—and want constant time access.
Mipmap Level-of-detail Selection

- Hardware uses 2x2 pixel entities
  - Typically called quad-pixels or just quad
  - Finite difference with neighbors to get change in \( u \) and \( v \) with respect to window space
    - Approximation to \( \partial u / \partial x, \partial u / \partial y, \partial v / \partial x, \partial v / \partial y \)
    - Means 4 subtractions per quad (1 per pixel)

- Now compute approximation to gradient length
  - \( p = \max(\sqrt{(\partial u / \partial x)^2 + (\partial u / \partial y)^2}), \sqrt{(\partial v / \partial x)^2 + (\partial v / \partial y)^2}) \)
Level-of-detail Bias and Clamping

- Convert p length to level-of-detail and apply LOD bias
  - $\lambda = \log_2(p) + \text{lodBias}$

- Now clamp $\lambda$ to valid LOD range
  - $\lambda' = \max(\min\text{LOD}, \min(\max\text{LOD}, \lambda))$
Determine Mipmap Levels

- Determine lower and upper mipmap levels
  - \( b = \text{floor}(\lambda') \) is bottom mipmap level
  - \( t = \text{floor}(\lambda' + 1) \) is top mipmap level

- Determine filter weight between levels
  - \( w = \text{frac}(\lambda') \) is filter weight
Mipmap texture example

```java
int vaoID = createVAO();
bindIndicesBuffer(indices);
storeDataInAttributeList(0,3,positions);
storeDataInAttributeList(1,2,textureCoords);
storeDataInAttributeList(2,3, normals);
unbindVAO();
return new RawModel(vaoID, indices.length);

public int loadTexture(String fileName) {
    Texture texture = null;
    try {
        texture = TextureLoader.getTexture("PNG",
            new FileInputStream("res/" + fileName + ".png");
        GL30.glGenerateMipmap(target, int pname, int param)
        GL11.glTexParameteri(GL11.GL_TEXTURE_2D, GL11.GL_TEXTURE_MIN_FILTER, param)
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println("Tried to load texture " + fileName + ".png, didn't work");
        System.exit(-1);
    }
    textures.add(texture.getTextureID());
    return texture.getTextureID();

public void cleanUp(){
    for(int vao:vaos){
        GL30.glDeleteVertexArrays(vao);
    }
```
Mipmap texture example
Anisotropic Texture Filtering

- Standard (isotropic) mipmap LOD selection
  - Uses magnitude of texture coordinate gradient (not direction)
  - Tends to spread blurring at shallow viewing angles
- Anisotropic texture filtering considers gradients direction
  - Minimizes blurring

[Images of isotropic and anisotropic texture filtering]
Bump mapping
How to make it more 3D

- Why does it look 3D?
How to make it more 3D

- How could I make a "photo cube" of friends and family to show to people that aren’t really interested.
- Assume you can only use one texture.
How to make it more 3D

- How would you add in shading?
Bump Mapping:
Perturbing mesh normals to create the appearance of geometric detail

Normal Mapping:
A way of implementing bump mapping
**Shading**

<table>
<thead>
<tr>
<th>Flat shading</th>
<th>Gouraud shading</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Flat shading diagram" /></td>
<td><img src="image2" alt="Gouraud shading diagram" /></td>
</tr>
<tr>
<td>Only the first normal of the triangle is used to compute lighting in the entire triangle.</td>
<td>The light intensity is computed at each vertex and interpolated across the surface.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phong shading</th>
<th>Bump mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Phong shading diagram" /></td>
<td><img src="image4" alt="Bump mapping diagram" /></td>
</tr>
<tr>
<td>Normals are interpolated across the surface, and the light is computed at each fragment.</td>
<td>Normals are stored in a bumpmap texture, and used instead of Phong normals.</td>
</tr>
</tbody>
</table>
Normal Map

- Normal vector encoded as rgb
  - \([-1,1]^3 \rightarrow [0,1]^3\): rgb = n*0.5 + 0.5

- RGB decoding in fragment shaders
  - vec3 n = texture2D(NormalMap, texcoord.st).xyz * 2.0 – 1.0

- In tangent space, the frame normal points in the +z direction.
  - Hence the RGB color for the straight up normal is (0.5, 0.5, 1.0).
  - This is why normal maps are a blueish color

- Normals are then used for shading computation
  - Diffuse: n•l
  - Specular: \((n\bullet h)^{\text{shininess}}\)
  - Computations done in tangent space
original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh and normal mapping
500 triangles
Bump mapping
Environment mapping
Types of Environment Maps

a) Sphere around object (sphere map)

b) Cube around object (cube map)
Forming a Cube Map

- Use 6 cameras directions from scene center
- each with a 90 degree angle of view
Reflection Mapping

- Need to compute reflection vector, r
Indexing into Cube Map

- Compute \( R = 2(N \cdot V)N - V \)
- Object at origin

- Use largest magnitude component of \( R \) to determine face of cube

- Other two components give texture coordinates
Sphere Mapping

- Original environmental mapping technique proposed by Blinn and Newell based in using lines of longitude and latitude to map parametric variables to texture coordinates
- OpenGL supports sphere mapping which requires a circular texture map equivalent to an image taken with a fisheye lens

Sphere map (texture)  Sphere map applied on torus
Refraction

- Can also use cube map for refraction (transparent)
Refraction

Reflection

Refraction
Need to Compute Refraction Vector

\[ I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + I_{tran} \]
Snell's Law

- Transmitted direction obeys Snell's law
- Snell's law: relationship holds in diagram below

\[
\frac{\sin(\theta_2)}{c_2} = \frac{\sin(\theta_1)}{c_1}
\]

c_1, c_2 are speeds of light in medium 1 and 2
Environmental mapping
Environmental mapping
And something more advanced (light mapping)
Projection mapping