Texture Mapping
Texture Mapping

- Offsets the modeling assumption that the BRDF doesn’t change in u and v coordinates along the object’s surface
- Store a reflectance as an image called a texture
- Map that image onto the object

For each triangle in the model establish a corresponding region in the phototexture
A texture image is defined in a 2D coordinate system: \((u, v)\)

Each triangle vertex is assigned a \((u,v)\) coordinate, so that one knows which part of the texture to stick on the triangle.

The texture is stuck onto the triangle by defining an \((R,G,B)\) texture color for each pixel inside the triangle.

Texture Coordinates

\[
\begin{array}{c}
0 & 1 \\
\hline
0 & 1 & 1 & 1 \\
\end{array}
\]

\((0,0)\) \hspace{1cm} \((1,0)\) \\
\((0,1)\) \hspace{1cm} \((1,1)\)

2 triangles
Texture Coordinates

- Calculate texture coordinates \((u, v)\) for each pixel using barycentric interpolation of the texture coordinates of the triangle’s vertices

\[
(u_0, v_0) \quad (u_1, v_1) \quad (u_2, v_2)
\]

Pixel texture coordinates:

\[
(u, v) = \alpha_0(u_0, v_0) + \alpha_1(u_1, v_1) + \alpha_2(u_2, v_2)
\]
Pixel RGB Color

- Given the pixel’s (u,v) texture coordinates, find the pixel’s color in the texture map - using interpolation!
Bilinear Interpolation

- First linearly interpolate in the u-direction, then linearly interpolate in the v-direction (or vice versa)
Bilinear Interpolation

\[ T(u, v) = (1-a)(1-b) \ T[i, j] + a \ (1-b) \ T[i+1, j] + (1-a)b \ T[i, j+1] + ab \ T[i+1, j+1] \]

- Bilinear interpolation is applied to each of the R, G, B (and alpha) channels, using the same weights
Screen Space vs. World Space

- Triangles change shape nonlinearly via perspective transformation, leading to different barycentric weights before and after the perspective transformation.
- Interpolating in screen space results in texture distortion.
- Interpolating in world space requires projecting all pixel locations backwards from screen space to world space, which is expensive.

![Texture Source](image1)

![What We Get](image2)

![What We Want](image3)
Texture Distortion

- Consider a single edge of a triangle
- Project this from world space to screen space
Texture Distortion

- Uniform increments along the edge in world space do not correspond to uniform increments along the edge in screen space.
- Barycentric interpolation (which is linear) does not account for this nonlinearity.
Mesh Refinement

- Refinement of the triangle mesh improves the result,
  - because a nonlinear function can be approximated as a piecewise linear function if the intervals are small enough
- More and more triangles helps everything mostly, but is not very efficient (or desirable)

- If the resolution varies, T-junctions cause problems too...
Mesh Refinement

- Better-ish, but still looks bad...
Screen Space Barycentric Interpolation

- Find the relationship between the barycentric weights in screen space and those in world space
- Use this relationship to compute the world space barycentric weights from the screen space barycentric weights
- This removes the need to inefficiently go back to world space to get the correct result
Screen Space Barycentric Interpolation

Ignore the y direction coordinate for simplicity

- The world space coordinates of the end points of an edge are

$$p^w_1 = \begin{pmatrix} x_1 \\ z_1 \end{pmatrix}, \quad p^w_2 = \begin{pmatrix} x_2 \\ z_2 \end{pmatrix}$$

- The corresponding screen space end points are

$$p_1 = \frac{hx_1}{z_1}, \quad p_2 = \frac{hx_2}{z_2}$$

- The world space coordinates of a point on the edge are given by

$$p^w(s) = (1 - s) \begin{pmatrix} x_1 \\ z_1 \end{pmatrix} + s \begin{pmatrix} x_2 \\ z_2 \end{pmatrix}$$

- The corresponding screen space location of $p^w(s)$ is

$$SCREEN\_SPACE( p^w(s) ) = h \frac{(1 - s)x_1 + sx_2}{(1 - s)z_1 + sz_2}$$

- This same point corresponds to some value of interpolation in screen space with $t$:

$$\left(1 - t\right) \frac{hx_1}{z_1} + t \frac{hx_2}{z_2} = h \frac{(1-s)x_1+sx_2}{(1-s)z_1+sz_2}$$
Screen Space Barycentric Interpolation

- Solving this last equation for $s$ gives a mapping from barycentric coordinates in screen space to barycentric coordinates in world space:

$$ s = \frac{tz_1}{z_2 + t(z_1 - z_2)} $$

- Thus for each pixel, compute its barycentric weights in screen space to obtain $t$
- Then use the mapping from screen space barycentric weights $t$ to world space barycentric weights $s$ to find the world space barycentric weights
- Use these world space barycentric weights $s$ to interpolate the texture coordinates from the triangle’s vertices

- N.B. this is the formula for a segment, not for a triangle…
Screen Space Barycentric Interpolation

For triangles...

• A pixel inside a screen space triangle has barycentric weights

\[ p(\alpha, \beta) = \alpha p_0 + \beta p_1 + (1 - \alpha - \beta)p_2 \]

• Following all the steps in the previous slides (for triangles) we obtain the world space barycentric weights

\[ \alpha^w = \frac{z_1z_2\alpha}{z_0z_1 + z_1\alpha(z_2 - z_0) + z_0\beta(z_2 - z_1)} \]

\[ \beta^w = \frac{z_0z_2\beta}{z_0z_1 + z_1\alpha(z_2 - z_0) + z_0\beta(z_2 - z_1)} \]

• These world space barycentric weights \( \alpha^w, \beta^w, (1 - \alpha^w - \beta^w) \) are used to interpolate the texture coordinates from the triangle vertices to that pixel

• Then the color is looked up in the texture map as usual...
Screen Space Barycentric Interpolation

- Texture source
- Results without perspective correct interpolation
- Results with perspective correct interpolation
Assigning Texture Coordinates
Assigning Texture Coordinates

- Typically assign coordinates one piece of the object at a time
Assigning Texture Coordinates

- For certain surfaces, the \((u, v)\) texture coordinates can be generated procedurally
  - Example: Cylinder
    - map the \(u\) coordinate from \([0, 1]\) to \([0, 2\ \pi]\) for \(\Phi\)
    - map the \(v\) coordinate from \([0, 1]\) to \([0, h]\) for \(y\)
    - This wraps the image around the cylinder
- For more complex surfaces, \((u, v)\) must be defined per vertex manually or by using proxy objects
Proxy Objects – Step 1

Assign texture coordinates to intermediate/proxy objects:

- Example: Cylinder
  - wrap texture around the outside of the cylinder
  - not the top or bottom, in order to avoid distorting the texture
- Example: Cube
  - unwrap the cube and map texture over the unwrapped cube
  - the texture is seamless across some of the edges, but not necessarily others
Proxy Objects – Step 2

- Map texture coordinates from the intermediate/proxy object to the final object
- Three ways of mapping are generally used:
  - Use the intermediate/proxy object’s surface normal
  - Use the target object’s surface normal
  - Use rays emanating from center of target object
Aliasing
Aliasing

- When textures are viewed from a distance, you may get aliasing

What we get | What we want

(more about aliasing in week 6)
Aliasing

- Texture mapping is point-sampling
- If the pixel sampling frequency is too low compared to the signal frequency (the texture resolution), aliasing occurs
- At an optimal distance, there is a 1 to 1 mapping from triangle pixels to texture pixels (texels)
- At a closer distance, each triangle pixel maps to a small part of a texture pixel, multiple pixels per texel (oversampling is fine)
- At a far distance, each triangle pixel maps to several texture pixels, but interpolation ignores all but the nearest texels – so information is lost (averaging would be better)
MIP Maps

- Multum in Parvo: Much in little, many in small places
- Precomputes the texture maps at multiple resolutions, using averaging as a low pass filter
- When texture mapping, choose the image size that approximately gives a 1 to 1 pixel to texel correspondence
- The averaging “bakes-in” all the nearby pixels that otherwise would not be sampled correctly
MIP Maps

- 4 neighboring pixels of the higher level are averaged to form a single pixel in the lower level.
- Starting at a base resolution, you can store EVERY coarser resolution in powers of 2 using only 1/3 additional space.
- $1 + 1/4 + 1/16 + \ldots = 4/3$
MIP Maps

- To use the MIP maps, look for the two MIP map images just above and below the screen space pixel resolution.
- Use the standard bilinear interpolation on both the higher and lower resolution MIP map images.
- Then use linear interpolation between those two bilinearly interpolated texture values, where the weights come from comparing the screen space resolution to that of the two MIP maps.
MIP Maps

point sampling

using mip-maps
RIP Maps

- Consider texturing a horizontal plane which is at an oblique angle to the camera
- The texel sampling rate will be much smaller vertically than horizontally
- Using an averaged MIP map created to avoid aliasing in the vertical direction also averages in the horizontal direction – causing unwanted blurring
- In MIP mapping, both texture axes are equally averaged
- RIP mapping is an anisotropic improvement to isotropic MIP mapping that coarsens both axes separately
RIP Maps

- RIP maps require 4 times the storage if we store every coarser resolution in each axial direction:
  \[(1 + 1/4 + 1/16 + ...) [1 + 2(1/2 + 1/4 + 1/8 + ...)] = 4\]
Homework & Resources
OpenGL

First load your image

- **Input:** any image source (as long as you know how to parse it)

- **Output:** unsigned byte array

- Bytes can encode data in many different ways, most common is RGB or RGBA (with each component an unsigned int = 4 bytes)

- Total data size for RGBA = width * height * 4

- Data looks like RGBARGBARGBA....
OpenGL

- GLint textureId;  // identifier for our texture
- glGenTextures(1, &textureId);  // make room for our texture
- glBindTexture(GL_TEXTURE_2D, textureId);  // tell GL which texture to edit
- glTexImage2D(GL_TEXTURE_2D, level, components, width, height, border, format, type, data);
  - level - for mip-maps. If you supply only one resolution, level should be 0. If you supply mip-maps with multiple resolutions, the highest resolution level should be 0, then 1, then 2, ...
  - components - how texture is store (r,g,b, how much to allocate for each component)
  - width, height, border
  - format of input data, and the data itself
- glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
OpenGL

... Create Texture ...

```c
glActiveTexture(GL_TEXTUREi);
glBindTexture(GL_TEXTURE_2D, textureId);
glUniform1i("diffuseTexture", i);
```

... Render ...

In GLSL:
```glsl
in vec2 fragmentUV;
uniform sampler2D diffuseTexture;
vec4 textureColor = texture(diffuseTexture, fragmentUV);
```
Online Databases

- You can find a number of useful texture maps online. For example, you can find many different crate textures for boxes by searching for "texture map crate" using Google.

- There are a number of websites containing different kinds of textures for different objects. Here are some examples:
  
  http://www.cgtextures.com/
  http://www.textureking.com/
  http://www.lughertexture.com/
  http://www.mayang.com/textures/
  http://www.marlinstudios.com/samples/samples.htm
  http://www.arroway-textures.com/
  http://www.environment-textures.com/
  http://www.3dttexture.net/