Texture Mapping
Texture Mapping

• Adds back the details lost by assuming that the BRDF doesn’t vary along an object’s surface
• These RGB reflectance modifications are stored as an image (called a texture)
• The image colors are mapped to the object’s surface (one triangle at a time)
Similar to Putting on Stickers
Texture Coordinates

• A texture image is defined in a 2D coordinate system: \((u, v)\)
• Texture mapping assigns a \((u, v)\) coordinate to each triangle vertex
• Then, the texture is “stuck” onto the triangle (potentially, with distortion):
  • Let \(p\) be a point inside the triangle, with barycentric weights \(\alpha_0, \alpha_1, \alpha_2\)
  • The color assigned to \(p\) is the texture color at \((u_p, v_p) = \alpha_0 (u_0, v_0) + \alpha_1 (u_1, v_1) + \alpha_2 (u_2, v_2)\)
  • That is, texture coordinates are barycentrically interpolated
Recall: Screen Space vs. World Space Barycentric Weights

- Express the pixel \( p' \) terms of its screen space barycentric weights: \( \alpha_0', \alpha_1', \alpha_2' \)
- Express the point \( p \) that projects to \( p' \) in terms of unknown world space barycentric weights: \( \alpha_0, \alpha_1, \alpha_2 \)
- Project \( p \) into screen space and set the result equal to \( p' \)
- Solve for \( \alpha_0, \alpha_1, \alpha_2 \) to obtain:

\[
\alpha_0 = \frac{z_1 z_2 \alpha_0'}{z_1 z_2 \alpha_0' + z_0 z_2 \alpha_1' + z_0 z_1 \alpha_2'}
\]

\[
\alpha_1 = \frac{z_0 z_2 \alpha_1'}{z_1 z_2 \alpha_0' + z_0 z_2 \alpha_1' + z_0 z_1 \alpha_2'}
\]

\[
\alpha_2 = \frac{z_0 z_1 \alpha_2'}{z_1 z_2 \alpha_0' + z_0 z_2 \alpha_1' + z_0 z_1 \alpha_2'}
\]
Screen Space vs. World Space Barycentric Weights

- Perspective transformation (nonlinearly) changes triangle shape
- Interpolating texture coordinates in screen space (nonlinearly) distorts textures

Mesh refinement helps (less $z$ variance per triangle)
Texture Distortion

- Consider a single edge of one triangle
- Uniform increments (along the edge) in screen space do not correspond to uniform increments in world space
Interpolating from the Texture Image

• \((u_p, v_p)\) is surrounded by 4 pixels in the texture image
• Use bilinear interpolation to interpolate values for: \(R, G, B, \alpha, \) etc.
  • First, linearly interpolate in the \(u\) direction; then, in the \(v\) direction (or vice versa)

\[
T(u_p, v_P) = (1 - a)(1 - b)T_{i,j} + a(1 - b)T_{i+1,j} + (1 - a)bT_{i,j+1} + abT_{i+1,j+1}
\]
Assigning Texture Coordinates

- Assign texture coordinates on complex objects one part/component at a time
Assigning Texture Coordinates

• Manually assigning \((u, v)\) one vertex at a time can be tedious

• For some surfaces, the \((u, v)\) texture coordinates can be generated procedurally
  • E.g. Cylinder (wrap the image around the outside)
    • map the \([0,1]\) values of the \(u\) coordinate to \([0,2\pi]\) for \(\phi\)
    • map the \([0,1]\) values of the \(v\) coordinate to \([0, h]\) for \(y\)
Proxy Objects – Step 1

• Assign texture coordinates to proxy objects:
  • Example: Cylinder
    • wrap texture coordinates around the outside of the cylinder
    • not the top or bottom (to avoid distorting the texture)
  • Example: Cube
    • unwrap cube, and map texture coordinates over the unwrapped cube
    • texture is seamless across some of the edges, but not other edges
Proxy Objects – Step 2

- Transfer texture coordinates from the proxy object to the final object
- Various ways of doing this:
  - Use the proxy object’s surface normal
  - Use the target object’s surface normal
  - Use rays emanating from a “center”-point/line of the target or proxy object
Distortion

- Difficult to find low-distortion mappings (back and forth) from a 2D plane to 3D surfaces
DEBUG with checkerboard textures
Aliasing

• Textures often alias when viewed from a distance
Aliasing

- Aliasing occurs when the sampling frequency is too low compared to the texture resolution (which is the signal frequency)

- At an optimal distance, there is a 1 to 1 mapping from triangle pixels to texture pixels (texels)
- At closer distances, triangle pixels (correctly) interpolate from texels
- At far distances, a triangle pixel should average together several texels
  - But, interpolation ignores all but the neighboring texels (resulting in aliasing)
MIP Maps

• Multum in Parvo (much in little)
• Precompute texture images at multiple resolutions, using averaging as a low pass filter
• Averaging “bakes-in” all the nearby texels that are otherwise interpolated incorrectly
• When texture mapping, choose the image size that (approximately) gives a 1 to 1 pixel to texel correspondence
MIP Maps

- 4 neighboring texels of one level are averaged to form a single texel at the next level
- Since $1 + \frac{1}{4} + \frac{1}{16} + \cdots = \frac{4}{3}$, can store all coarser resolutions with $1/3$ additional space
Using MIP Maps

• Find the MIP map image *just above* and *just below* the screen space pixel resolution
• Use bilinear interpolation on both MIP map images
• Linearly interpolate between the two results (with weights based on comparing the screen space resolution to that of the two MIP map images)
RIP Maps

• A triangle tilted away from the camera has different texel sampling rates in the horizontal and vertical directions.
• MIP map images can only match one of the two sampling rates.
• Anisotropic RIP maps are designed to account for this.
• RIP maps require 4 times the storage:

\[
\left(1 + \frac{1}{4} + \frac{1}{16} + \cdots \right) \left[1 + 2 \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \right)\right] = 4
\]