Texture Mapping
Texture Mapping

- Adds back the details lost by assuming that the BRDF doesn’t change along an object’s surface
- RGB reflectance is stored as an image (called a texture)
- The image colors are mapped to the object’s surface (one triangle at a time)
Similar to Wrapping Gifts
Texture Coordinates

• The texture 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:
  • 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:

$$\alpha_0 = \frac{z_1z_2\alpha'_0}{z_1z_2\alpha'_0 + z_0z_2\alpha'_1 + z_0z_1\alpha'_2}$$

$$\alpha_1 = \frac{z_0z_2\alpha'_1}{z_1z_2\alpha'_0 + z_0z_2\alpha'_1 + z_0z_1\alpha'_2}$$

$$\alpha_2 = \frac{z_0z_1\alpha'_2}{z_1z_2\alpha'_0 + z_0z_2\alpha'_1 + z_0z_1\alpha'_2}$$
Screen Space vs. World Space Barycentric Weights

- Perspective transformation nonlinearly changes a triangle’s shape
- Interpolating texture coordinates in screen space results in texture distortion
Texture Distortion

- Consider one triangle edge
- 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: \(T = R, G, B, \alpha, \text{etc.}\)
  - First, linearly interpolate in the \(u\) direction; then, in the \(v\) direction (or vice versa)

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

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

• For complex surfaces, 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 intermediate/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 necessarily other edges
Proxy Objects – Step 2

• Next, map the texture coordinates from the intermediate/proxy object to the final object
• Three ways of doing this:
  • Use the intermediate/proxy object’s surface normal
  • Use the target object’s surface normal
  • Use rays emanating from a “center”-point or “center”-line of the target object
Distortion

- It’s 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 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 texture pixels.
- At far distances, a triangle pixel should use several texture pixels.
  - But, interpolation ignores all but the nearest texture pixels (resulting in aliasing).

1 to 1 (optimal)  pixel interpolates from texels  pixel should use multiple texels
MIP Maps

- Multum in Parvo (much in little)
- Precompute 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 one level are averaged to form a single pixel at the next lower level
• Since $1 + \frac{1}{4} + \frac{1}{16} + \cdots = \frac{4}{3}$, can store EVERY coarser resolution using only 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** the higher/lower resolution MIP map images
- Linearly interpolate between the results (with weights based on comparing the screen space resolution to that of the two MIP maps)
RIP Maps

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

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