Goal: Complexity

- How to make something complex?
- Given what we have: lots of small triangles
  - To now, Gouraud shading – color per vertex
- Why not?
  - Hard to model / author / design
  - Hard to draw fast
  - Hard to sample (triangles get smaller than a pixel)
  - Hard to maintain the models
  - Hard to store the models

Alternative Approach to Complexity:
“Texture” Mapping (and its variants)

- Use simple geometry (big polygons)
- Vary color (and other things) over its surface
- Analogy: paint a picture on something
- Basic case: change color at each point
  - Advanced cases later

Why just paint objects?

- Why paint rather than model?
  - Easier (can use 2D tools, photographs)
  - Less to store
  - Less to model
  - Easier to draw (*)
  - Easier to sample
- Why not?
  - Things really aren’t flat
  - Parallax / self-shadowing / illumination effects
  - More advanced “texturing” to get these later

Faster to draw requires special hardware!
Only recently has this become common!

Texture Mapping

- For every point on the object, have a “map” (function) to color
  - Later extend to other properties
- Big pieces here:
  - Need ways to “name” points on object
    - Texture Coordinates
  - Need ways to describe the mappings
    - Procedural
    - Use images

How to assign points to objects

- Use world space positions?
  - No – properties usually move with objects
  - Might be OK for things like lights that effect objects
- Use local 3D positions?
  - 3D Textures
  - Problem: harder to define functions that give colors for all points in a volume
  - Don’t care about points off the surface anyway
  - Use 3D textures when it’s easy to make 3D functions
    - Procedural wood, stone, …
2D Texture Mapping

- So common, it's almost synonymous with Texture
- For every point, give a 2D coordinate
  - Texture coordinate
  - U,V for every vertex
- Interpolate across triangles
  - (same as across quads)

Interpolating Coordinates

\[ x = \frac{x_0 (1 - \alpha - \beta) + x_1 \alpha + x_2 \beta}{\text{Area}(x_0, x_1, x_2)} \]

Barycentric Coordinates

- An alternate way of describing points in triangles
- These can be used to interpolate texture coordinates
  - Gives the same result as previous slide
  - Method in textbook (Shirley)

\[ x = \alpha x_0 + \beta x_1 + \gamma x_2 \]
\[ \frac{\text{Area}(x, x_1, x_2)}{\text{Area}(x_0, x_1, x_2)} = \alpha \]
\[ \frac{\text{Area}(x, x_1, x_2)}{\text{Area}(x_0, x_1, x_2)} = \beta \]
\[ \delta = 1 - \alpha - \beta \]

How to represent the function

- \( C(u,v) \)
  - Write code (needs programmable graphics system)
    - Programmable shaders (later in course)
    - Use an image and sample
  - Sampling is an issue even for procedural texture
    - It's just harder!
  - One pixel can be a large part of a triangle

Image Based Texture Maps

- So common it's synonymous
- U,V coords at vertices
- Specify where in texture to get colors

Perspective Correction

- Linear interpolation wrong if polygon isn't screen aligned
- Stuff farther away needs to be smaller
- Need to interpolate in world space, then do perspective
- Need to interpolate w, and divide (per-pixel)
- Divide per pixel used to be expensive
Perspective Correct Texture Mapping

- Don’t worry – the graphics hardware does it
- 1/Z (or 1/W) is linear in screen space
  - This is a little tricky to prove

To do perspective correct

- Interpolate 1/Z (or 1/W)
- Compute Z (from 1/Z) – requires divide
- Compute fraction of way from begin to end in Z
- Use this fraction to get how far in U/V
- Can combine steps

- Big picture – need to do a divide for every conversion (pixel)
- See Shirley for details

Sampling

- Have U, V for the pixel – what color is it?
- Look it up in the texture map
- Point sample
- Bilinear interpolation (if between pixels)
  - Always will be between pixels
- Filtering – pixel maps to a region of texture

Fast Sampling

- Screen pixel is funny shape in Texture Space
- Perspective transform of circle (skewed ellipse)
- Use a simpler shape for sampling

Average over rectangular regions

- Sum over region in constant time / precomputed
- table was above and to the left
- A+B-C-D+E
- 2x2 lookup, but need table – overflow case

Square Region Centered at Point

- Pretend pixels are squares
- If region is 1 pixel big, this is easy!
  - Use bilinear interpolation to get position right
- If the region is bigger, halve both region and image
  - 2x2 region – halve the image (each pixel is average of a 2x2 block)
  - 4x4 region – halve the image twice
MIP Map

- Repeatedly halve the image to make a "pyramid"
  - Until there’s 1 pixel (which is average of whole)
- Given a position and square size
  - Use square size to pick pyramid level
  - Use bilinear interpolation to get position
- But only have pyramid for 1,2,4,8… pixel squares
  - Linear interpolate between levels!
  - E.g. 5 = ¼ way between 4 and 8, so compute 4 and 8 and interpolate
  - Tri-Linear Interpolation! - looks at 8 texels (4 per level)

Making Textures Work

- Need to load textures into FAST memory
  - Multiple lookups per pixel
- Need to build MipMaps
- Need to give triangles UV values
- Need to decide how to filter
- How is texture color used
  - Replace existing color?
  - Blend with it?
  - Before or after specular highlight?
- Need to decide what happens to "out of bounds" texture coordinates
  - Clamp, repeat, border

More stuff with textures

- Lots of extensions and uses!
- Multi-Texturing (combine several textures)
- Bump Mapping – lookup normal values
- Displacement Mapping
- Textures for lighting and shadows
- Can fake many complex effects by using texturing in interesting ways
  - Draw many times – each with another texture