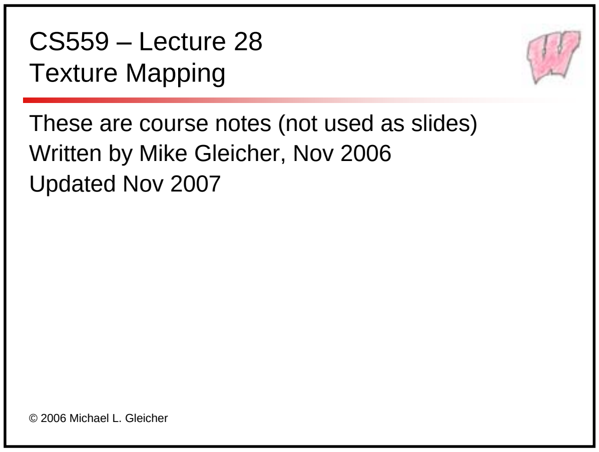
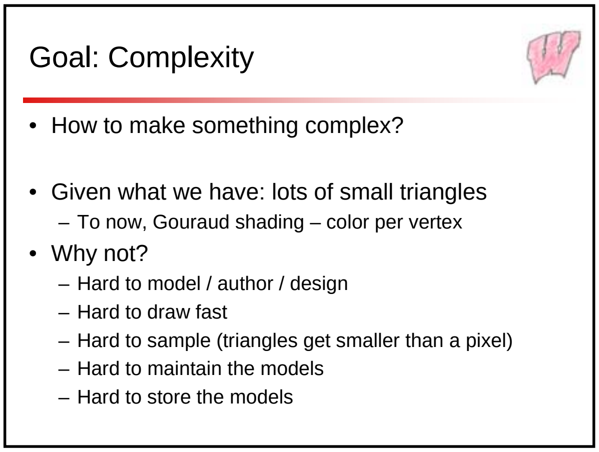


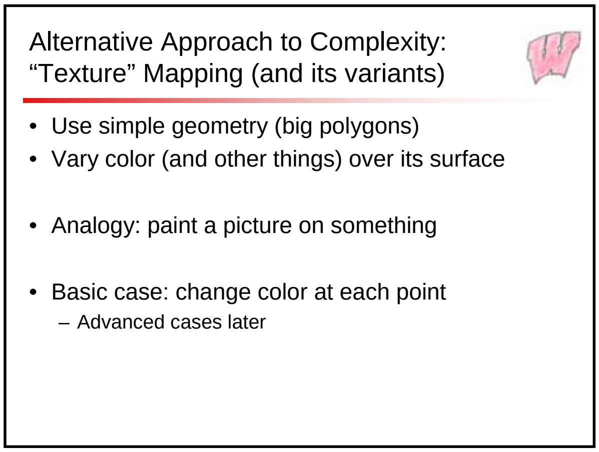
CS559 – Lecture 28
Texture Mapping

These are course notes (not used as slides)
Written by Mike Gleicher, Nov 2006
Updated Nov 2007



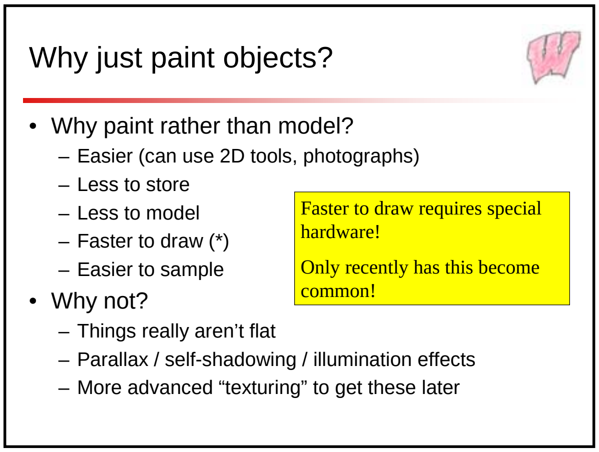
Goal: Complexity

- 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

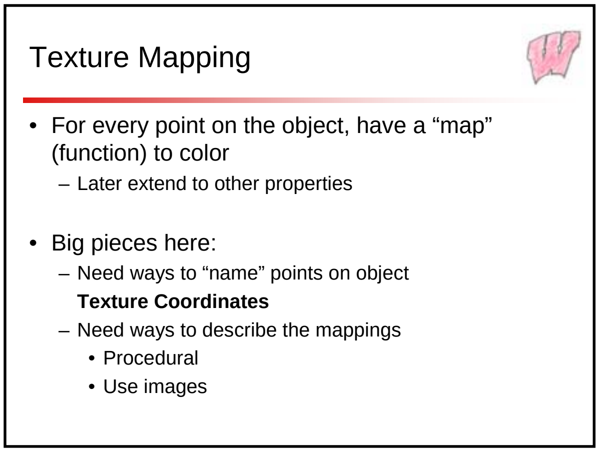


Why just paint objects?

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

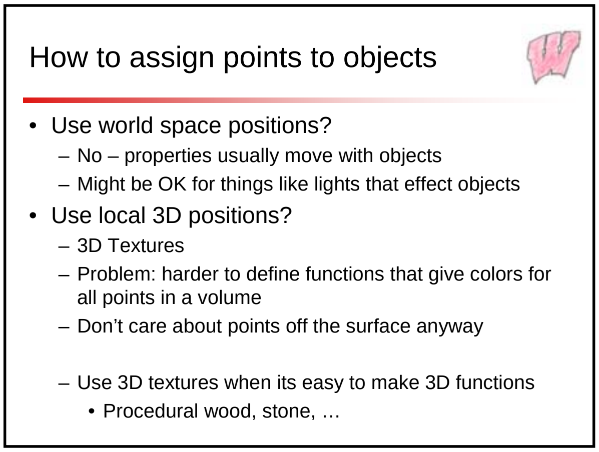
Faster to draw requires special hardware!

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



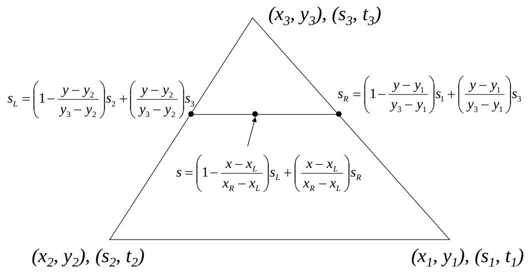
How to assign points to objects

- 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 its easy to make 3D functions
    - Procedural wood, stone, …

2D Texture Mapping

- So common, its 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



$$s_L = \left(1 - \frac{y - y_2}{y_3 - y_2}\right) s_2 + \left(\frac{y - y_2}{y_3 - y_2}\right) s_3$$

$$s_R = \left(1 - \frac{y - y_1}{y_3 - y_1}\right) s_1 + \left(\frac{y - y_1}{y_3 - y_1}\right) s_3$$

$$s = \left(1 - \frac{x - x_L}{x_R - x_L}\right) s_L + \left(\frac{x - x_L}{x_R - x_L}\right) s_R$$

Barycentric Coordinates

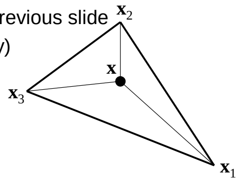
- An alternate way of describing points in triangles
- These can be used to interpolate texture coordinates
  - Gives the same result as previous slide
  - Method in textbook (Shirley)

$$\mathbf{x} = \alpha \mathbf{x}_1 + \beta \mathbf{x}_2 + \gamma \mathbf{x}_3$$

$$\alpha = \frac{\text{Area}(\mathbf{x}, \mathbf{x}_2, \mathbf{x}_3)}{\text{Area}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)}$$

$$\beta = \frac{\text{Area}(\mathbf{x}_1, \mathbf{x}, \mathbf{x}_3)}{\text{Area}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)}$$

$$\delta = 1 - \alpha - \beta$$

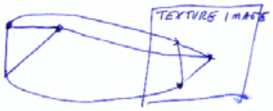


How to represent the function

- 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
  - Its just harder!
- One pixel can be a large part of a triangle

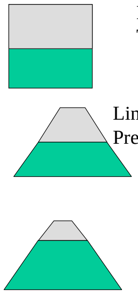
Image Based Texture Maps

- So common its 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



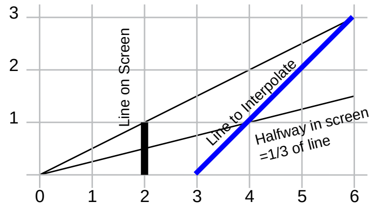
Equal size,
Tip it back

Linear
Preserve size ratio

Perspective
Correct

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



1/2 in screen line

Interpolate 1/3->1/6
halfway = 1.5 / 6

Z of halfway point=4

1/3 of line

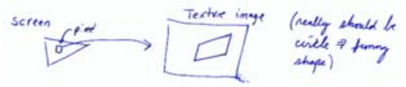
To do perspective correct

- 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

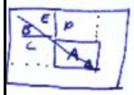
- Have U,V for the pixel – what color is it?
- Look it up in the texure map
- Point sample
- Bilinear interpolation (if between pixels)
  - Always will be between pixels
- Filtering – pixel maps to a region of texture

Fast Sampling



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

Average over rectangular regions



sum over region in constant
time w/ precomputed

table – area above and to the left

$A = B - C - D + E$

↳ 4 lookups, but need
table – overflow issue

need to know rectangle

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

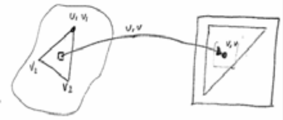
- Need to load textures into FAST memory
  - Multiple lookups per pixel
- Need to build MipMaps
- Need to give triangles UV values
- Need to decide how to filter
- How is texture color used
  - Replace existing color?
  - Blend with it?
  - Before or after specular highlight?
- Need to decide what happens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

RECAP

- Object / Triangle
- Texture Coordinates
- Interpolation
  - Linear in space
  - Perspective on screen
- For each pixel lookup color
- Bilinear interpolation
- Tri-Linear interpolation
  - MIPMAP



Small Gotcha

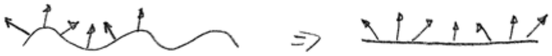
- Lighting computed at Vertex
- Color (texture) at each pixel
- Do per-pixel lighting (write a shader)
- Do Gouraud Shading on “Base Color”
  - Texture modulates base color
  - Color = Ct * Cl (color from lighting)
  - Make objects white, mult “over” color
  - Special tricks for dealing with specular highlights

Color Modulation

- Rather than multiply…
  - Add, subtract, …
- Combining multiple color sources
- Use lighting + colors
- Use multiple textures simultaneously
  - Basic color, plus surface detail
  - Use textures for lighting effects

But my object still look flat

- Simple method – BUMP mapping
- Use texture to change NORMAL
- Object is still flat, but reflects as if bumpy
- Normal map = displacement of “real” normal vector
  - N’ = N + a U + b V (U,V=tangents, N=original normal)



Bump Mapping is limited

- Only changes lighting
- No self-shadowing
- Doesn’t change silhouette
- But can be done with clever combinations of basic texturing
- Improved versions hack some of the benefits

Displacement Mapping

- Actually move points
  - Moving points changes normal
- Map stores positional offsets
  - Usually relative to surface direction
- Hard to do – since a pixel might get moved into another pixel

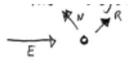
Multi-Texturing

- Use multiple textures
  - Combine together
- Many uses
  - Different textures based on viewpoint (or light direction)
  - Different “layers” of texture (scratches in woodgrain)
  - Light effects “painted on”
    - Complex highlights, reflections, shadows, …
- How to do?
  - Texture combiners
  - Multiple texture access in shaders (but limits…)
  - Multi-pass rendering (talk about later)

Environment Mapping

- Make mirror reflections
- Draw a picture of the world onto a map
  - Must know what will be reflected
  - Typically make a sphere or cube
- Assume object is an infinitessimal sphere

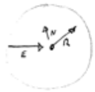
Environment map details



R depends on E, N

assume E is constant

Use R to Look up into Map of "Environment"



← spherical environment map

cubic environment map

Cylindrical environment map

Lighting with Texture

- Paint lighting onto objects
- Volumetric textures (things get lit around source)
- Environment map
  - Allows for positioning of many lights
  - Allows for capture of real lights
  - Mainly for specular highlights
    - But sampling (mipmapping) can give fuzzy highlights for things in-between specular and diffuse
- Slide projector mapping

Shadow Mapping

- Not to be confused with painting dark spots
  - Which is like slide-projector mapping
- Shadow map – can light be seen
  - Render scene from light's point of view
  - Visible objects are lit, others are shadowed
  - Keep the Z-buffer (the shadow map) to know which object

Hack Shadows / Spotlights

- Draw black or white splotches
- Draw semi-transparent
  - How to avoid overdraw?

shouldn't be twice as dark, but if using blend w/ black it will be

- Stencil buffer
  - A buffer you can write any value you want to
  - Write values when drawing
  - Test values when drawing
- Useful for many things in multi-pass rendering

Hack Shadows with Stencil Buffer

- Clear stencil buffer to zero
- Draw the ground plane with stencil buffer on
- Draw the shadows
  - Only draw with the stencil buffer bit set
  - Set to zero when drawn
- Notice:
  - No drawing off of the ground plane
  - No overdraw of the shadows!