Texturing (theory)

CS559 – Fall 2016
Lecture 17
March 29th 2016
Before fragment shading

Color per triangle
Color per vertex

Interpolation

How to get complexity?
Lots of Triangles?
Why not lots of triangles?

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
The pipeline (1988)
The full fixed-function pipeline (1992)
Per-Pixel Coloring

Have bigger triangles
Change color per pixel (fragment)

Change other stuff
  normals
  positions
Paint Pictures on Triangles

Assuming you have hardware
Easier (can use 2D tools, photographs)
Less to store
Less to model
Faster to draw
Easier to sample
Why not paint on triangles?

Things really aren’t flat

Parallax / self-shadowing / illumination effects

More advanced “texturing” to get these later
Texture Mapping

A function (map) of pixels to colors (later other properties)

Two big pieces:
1. Need to “name” the pixels
   Texture Coordinates
2. Need to describe the functions
What coordinates to use

Screen Coordinates?
World Coordinates?
Local Coordinates?
Triangle Coordinates?

User-defined, convenient coordinates?
Checkerboard example

Screen space checkers
World-space checkers
Triangle-space checkers

Texture Coordinates
Texture Coordinates

Assigned to a vertex
Independent of position

Can be 1D, 2D, 3D, 4D

just a number/vector to use in function
For a triangle...

Specify texture coordinates at vertices
Interpolate within the triangle

(caveat about interpolation coming)
Interpolating Coordinates

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

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

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

\[
x = \alpha x_1 + \beta x_2 + \gamma x_3
\]

\[
\alpha = \frac{\text{Area}(x, x_2, x_3)}{\text{Area}(x_1, x_2, x_3)}
\]

\[
\beta = \frac{\text{Area}(x_1, x, x_3)}{\text{Area}(x_1, x_2, x_3)}
\]

\[
\delta = 1 - \alpha - \beta
\]
A Detail on Interpolation
(of historical importance)

Need to interpolate before the divide by w
“Perspective Correct” interpolation

Or interpolate 1/w

(don’t worry, modern hardware does it right)
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

\[ \frac{1/3}{2} \text{ in screen line} \]

Interpolate $1/3 \rightarrow 1/6$

halfway = $1.5 / 6$

$Z$ of halfway point = 4

$1/3$ of line
The usual case

Triangles
2D texture coordinates

Draw the picture – 2D map
How to define the function

\[ f(u,v) \rightarrow r,g,b \]

Procedurally – write a program

easy now that we have shaders

Image-based – use a picture
look up the colors
Image-Based Textures

Texture Mapping often means this but it is a specific case.

Very convenient – since we can paint, photograph, pre-compute, …
Tricky Part

A (triangle) Pixel might map to...
- Much smaller than a (picture) pixel
- Much bigger than a (picture) pixel

(diagram)

Minification and Magnification
Sampling / Aliasing Issue
Point Sampling

Look up U,V in the grid

Nearest Neighbor
Interpolation

Minification Case
Magnification Case (diagrams)
Is this only a problem with images?

Aliasing in procedural textures

Checkerboard(u,v) =
(int(u) + int(v)) % 2 ? Black : White

Zoom out? (400% - every 4\textsuperscript{th} pixel)
Zoom in?
Anti-Aliasing Procedural Textures

It’s Hard.
Back to the image case...

Each pixel in the result (square? circle?) comes from
Some shape in the source image stretched by texture coordinates distorted by perspective

Filtering – average over this region
Ways to make sampling fast

Rectangular region
  width * height pixels

Amortize the cost
  pre-compute to make fast

Summed-Area Tables
Average over rectangular regions

\[ A = B - C - D + E \]

sum over region in constant time w/ precomputed table - area above and to the left

\[ \uparrow \text{4 lookups, but need table - overflow issue} \]

need to know rectangle
• 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
Programming This in GL

A lot of steps, lots can go wrong

Need to load in an image
   and get to graphics card memory
Need to build the MIP MAP
Getting Your Object Ready

Need texture coordinates
your shader can use anything

How does lighting combine with texture?
Accessing the texture map from a shader

Concept of a **Sampler**

Special variable type that encapsulates lookup into an image
Lots of parameters to set up

What Image
What kind of filter for minification
What kinds of filter for magnification
How to deal with texture boundaries
   Wrap? Mirror? Clamp?
Texture Unit (?)
   connect sampler to hardware
RECAP

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

\[ R \text{ depends on } E, N \]

assume \( E \) is constant

use \( R \) to look up into Map of "Environment"

\leftrightarrow \text{spherical environment map}

\rightarrow \text{cubic environment map}

\rightarrow \text{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?

• 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!