Textures I

Week 8, Mon Feb 28

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005
News

- face to face p2 grading this week
  - you can check your time slot from scans posted to course page
    - (student numbers blocked out)
  - Mon 1-5, Tue 10-1, 3-5, Wed 1-5

- midterm scaling announced
Midterm 1 Raw Scores

- range 26-98, avg 66
Midterm 1 Scaled Scores

- range 33-98, avg 69
News

- Homework 2 Q1-3 correction
  - point A is (2,0,0)
  - point B is (3,0,0)
  - point C is (4,0,0)
Review: Warnock’s Algorithm

- start with root viewport and list of all objects
- recursion:
  - clip objects to viewport
  - if only 0 or 1 objects
    - done
  - else
    - subdivide to new smaller viewports
    - distribute objects to new viewpoints
    - recurse
Review: Warnock’s Algorithm

- termination
  - viewport is single pixel
  - explicitly check for object occlusion
- single-pixel case common in high depth complexity scenes
Review: Z-Buffer Algorithm

- augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel
  - at frame beginning, initialize all pixel depths to $\infty$
  - when rasterizing, interpolate depth (Z) across polygon
- check Z-buffer before storing pixel color in framebuffer and storing depth in Z-buffer
- don’t write pixel if its Z value is more distant than the Z value already stored there
Z-Buffer Demo
Review: Object vs. Image Space

- **object space**
  - determine visibility on object or polygon level
  - resolution independent, VCS / NDC coords
  - early in pipeline
  - requires depth sorting objects/polylgons

- **image space**
  - determine visibility at viewport or pixel level
  - resolution dependent, screen coords
  - late in pipeline
Texturing
Reading (whole week)

- FCG Chapter 10
- Red Book Chapter Texture Mapping
Rendering Pipeline
Texture Mapping

- real life objects nonuniform in terms of color & normal
- to generate realistic objects - reproduce coloring & normal variations = Texture
- can often replace complex geometric details
Texture Mapping

- introduced to increase realism
  - lighting/shading models not enough
- hide geometric simplicity
  - images convey illusion of geometry
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface
- associate 2D information with 3D surface
  - point on surface corresponds to a point in texture
  - “paint” image onto polygon
Color Texture Mapping

- define color (RGB) for each point on object surface
- two approaches
  - surface texture map
  - volumetric texture
Surface Texture

- define texture pattern over \((u,v)\) domain (Image)
  - image – 2D array of “texels”
- assign \((u,v)\) coordinates to each point on object surface
- for free-form – use inverse of surface function
- for polygons (triangle)
  - inside – use barycentric coordinates
  - for vertices need mapping function
Texture Mapping
Mapping for Triangular Meshes

- mapping defined by
  - vertices (3D) mapped to specified (u,v) locations in 2D
  - each interior point mapped to 2D using barycentric coordinates
Texture Mapping

- texture map is an image, two-dimensional array of color values (**texels**)
- texels are specified by texture’s (u,v) space
- at each screen pixel, texel can be used to substitute a polygon’s surface property (color)
- we must map (u,v) space to polygon’s (s, t) space
Example Texture Map

Applied to tilted polygon
Texture Mapping

- (u,v) to (s,t) mapping can be explicitly set at vertices by storing texture coordinates with each vertex
- OpenGL
  - generation at vertices
    - specified by programmer or artist
      
      ```
      glTexCoord2f(s, t)
      glVertex3f(x, y, z)
      ```
    - ...
Texture Coordinates

- every polygon has object coordinates and texture coordinates
  - object coordinates describe where polygon vertices are on the screen
  - texture coordinates describe texel coordinates of each vertex
  - texture coordinates are interpolated along vertex-vertex edges
- `glTexCoord{1234}{sifd}(TYPE coords)`
Example Texture Map

![Diagram of texture mapping with vertices and texture coordinates]

- `glVertex3d(s, s, s)`
- `glTexCoord2d(1,1);`
- `glVertex3d(-s, -s, -s)`
- `glTexCoord2d(0,0);`

**Texture**  
(0, 0)  
(0, 1)  

**Object**  
(-s, -s, s)  

**Mapped Texture**  

```
```
Example Texture Map

```c
void ExampleTextureMap()
{
    glVertex3d(s, s, s);
    glTexCoord2d(5, 5);
    glVertex3d(s, s, s);
    glTexCoord2d(1, 1);
}
```
Texture Lookup

**issue:**
- what happens to fragments with $s$ or $t$ outside the interval $[0...1]$?

**multiple choices:**
- take only fractional part of texture coordinates
  - cyclic repetition of texture to tile whole surface
    
    \[
    \text{glTexParameteri( ..., GL_TEXTURE_WRAP_S, GL_REPEAT )}
    \]

- clamp every component to range $[0...1]$
  - re-use color values from border of texture image
    
    \[
    \text{glTexParameteri( ..., GL_TEXTURE_WRAP_S, GL_CLAMP )}
    \]
Texture Coordinate Transformation

- Motivation:
  - Change scale, orientation of texture on an object

- Approach:
  - *texture matrix stack*
  - 4x4 matrix stack
  - transforms specified (or generated) tex coords
  
```c
glMatrixMode( GL_TEXTURE );
glLoadIdentity();
...
```
Texture Coordinate Transformation

Example:

```
glScalef(4.0, 4.0, ?);
```
Texture Functions

- once have value from the texture map, can:
  - directly use as surface color `GL_REPLACE`
  - modulate surface color `GL_MODULATE`
  - blend surface and texture colors `GL_DECAL`
  - blend surface color with another `GL_BLEND`

- specific action depends on internal texture format
  - only existing channels used

- specify with `glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode)`
Demo: Robbins Tutor
Texture Pipeline

1. Compute object space location
2. Use projector function to find \((u, v)\)
3. Use corresponder function to find texels
4. Apply value transform function (e.g., scale, bias)
5. Modify illumination equation value
Texture Pipeline

Object position: \((-2.3, 7.1, 17.7)\)

Parameter space: \((u, v)\) = \((0.32, 0.29)\)

Image space: \((81, 74)\)

Texture:
- Texel color: \((0.9, 0.8, 0.7)\)
- Image space: \((81, 74)\)

\((x, y, z)\) in object space maps to \((u, v)\) in parameter space, which then maps to texture coordinates in image space.
Texture Mapping

(s, t) parameterization in OpenGL
Texture Mapping

- texture coordinates
  - generation at vertices
    - specified by programmer or artist
      \[ \text{glTexCoord2f}(s, t) \]
      \[ \text{glVertexf}(x, y, z) \]
    - generate as a function of vertex coords
      \[ \text{glTexGeni}(), \text{glTexGenfv}() \]
      \[ s = a^*x + b^*y + c^*z + d^*h \]
  - interpolated across triangle (like R,G,B,Z)
    - …well not quite!
Texture Mapping

- texture coordinate interpolation
  - perspective foreshortening problem
  - also problematic for color interpolation, etc.
Attribute Interpolation

Bilinear Interpolation
Incorrect

Perspective correct
Correct
Texture Coordinate Interpolation

- perspective correct interpolation
  - \( \alpha, \beta, \gamma : \)
    - barycentric coordinates of a point \( P \) in a triangle
  - \( s0, s1, s2 : \)
    - texture coordinates of vertices
  - \( w0, w1, w2 : \)
    - homogeneous coordinates of vertices

\[
S = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}
\]
Volumetric Texture

- define texture pattern over 3D domain - 3D space containing the object
  - texture function can be digitized or procedural
  - for each point on object compute texture from point location in space
- common for natural material/irregular textures (stone, wood, etc… )
Volumetric Texture Principles

- 3D function \( \rho \)
  - \( \rho = \rho(x,y,z) \)
- texture space – 3D space that holds the texture (discrete or continuous)
- rendering: for each rendered point \( P(x,y,z) \) compute \( \rho(x,y,z) \)
- volumetric texture mapping function/space transformed with objects
Texture Effects: Simple Marble

- boring marble

  function boring_marble(point)
    
    x = point.x;
    
    return marble_color(sin(x));
    
    // marble_color maps scalars to colors
Texture Effects: Bombing

- bombing
  - randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - for point P search table and determine if inside shape
    - if so, color by shape
    - otherwise, color by objects color