Hidden Surfaces III

Week 9, Wed Mar 17

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010
Review: BSP Trees

- preprocess: create binary tree
  - recursive spatial partition
  - viewpoint independent
Review: BSP Trees

• runtime: correctly traversing this tree enumerates objects from back to front
  • viewpoint dependent: check which side of plane viewpoint is on **at each node**
  • draw far, draw object in question, draw near
Review: The 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
More: Integer Depth Buffer

• reminder from picking discussion
  • depth lies in the NDC z range [0,1]
  • format: multiply by $2^n -1$ then round to nearest int
    • where $n = \text{number of bits in depth buffer}$
• 24 bit depth buffer = $2^{24} = 16,777,216$ possible values
  • small numbers near, large numbers far
• consider depth from VCS: $(1<<N) \times (a + b / z)$
  • $N = \text{number of bits of Z precision}$
  • $a = \frac{z\text{Far}}{(z\text{Far} - z\text{Near})}$
  • $b = \frac{z\text{Far} \times z\text{Near}}{(z\text{Near} - z\text{Far})}$
  • $z = \text{distance from the eye to the object}$
Review: Depth Test Precision

• reminder: perspective transformation maps eye-space (view) $z$ to NDC $z$

\[
\begin{bmatrix}
E & 0 & A & 0 \\
0 & F & B & 0 \\
0 & 0 & C & D \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
=
\begin{bmatrix}
Ex + Az \\
Fy + Bz \\
Cz + D \\
-1 \\
\end{bmatrix} =
\begin{bmatrix}
-\left( \frac{Ex}{z} + Az \right) \\
-\left( \frac{Fy}{z} + Bz \right) \\
-\left( C + \frac{D}{z} \right) \\
1 \\
\end{bmatrix}
\]

• thus: 
\[
z_{NDC} = -\left( C + \frac{D}{z_{eye}} \right)
\]
Review: Depth Test Precision

- therefore, depth-buffer essentially stores $1/z$, rather than $z$!
- issue with integer depth buffers
  - high precision for near objects
  - low precision for far objects

![Diagram showing depth test precision]
Review: Depth Test Precision

- Low precision can lead to **depth fighting** for far objects
  - Two different depths in eye space get mapped to same depth in framebuffer
  - Which object “wins” depends on drawing order and scan-conversion
- Gets worse for larger ratios $f:n$
  - **Rule of thumb**: $f:n < 1000$ for 24 bit depth buffer
- With 16 bits cannot discern millimeter differences in objects at 1 km distance
- Demo:
  sjbaker.org/steve/omniv/love_your_z_buffer.html
Correction: Ortho Camera Projection

- camera’s back plane parallel to lens
- infinite focal length
- no perspective convergence
- just throw away z values
- x and y coordinates do not change with respect to z in this projection

\[
\begin{align*}
\begin{bmatrix}
x_p \\
y_p \\
z_p
\end{bmatrix} &= \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \\
\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\end{align*}
\]
Z-Buffer Algorithm Questions

• how much memory does the Z-buffer use?
• does the image rendered depend on the drawing order?
• does the time to render the image depend on the drawing order?
• how does Z-buffer load scale with visible polygons? with framebuffer resolution?
Z-Buffer Pros

• simple!!!
• easy to implement in hardware
  • hardware support in all graphics cards today
• polygons can be processed in arbitrary order
• easily handles polygon interpenetration
• enables deferred shading
  • rasterize shading parameters (e.g., surface normal) and only shade final visible fragments
Z-Buffer Cons

- poor for scenes with high depth complexity
  - need to render all polygons, even if most are invisible

- shared edges are handled inconsistently
  - *ordering dependent*
Z-Buffer Cons

- requires lots of memory
  - (e.g. 1280x1024x32 bits)
- requires fast memory
  - Read-Modify-Write in inner loop
- hard to simulate translucent polygons
  - we throw away color of polygons behind closest one
  - works if polygons ordered back-to-front
    - extra work throws away much of the speed advantage
Hidden Surface Removal

- two kinds of visibility algorithms
  - object space methods
  - image space methods
Object Space Algorithms

- determine visibility on object or polygon level
  - using camera coordinates
- resolution independent
  - explicitly compute visible portions of polygons
- early in pipeline
  - after clipping
- requires depth-sorting
  - painter’s algorithm
  - BSP trees
Image Space Algorithms

- perform visibility test for in screen coordinates
  - limited to resolution of display
  - Z-buffer: check every pixel independently
- performed late in rendering pipeline
Projective Rendering Pipeline

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

glVertex3f(x,y,z)

modeling transformation

viewing transformation

projection transformation

viewport transformation

alter w

glFrustum(...)

clipping

normalized device

NDCS

DCS

glutInitWindowSize(w,h)
glViewPort(x,y,a,b)

gluLookAt(...)

perspective division

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

glTranslatef(x,y,z)
glRotatef(th,x,y,z)

....
Rendering Pipeline

Geometry Database

Model/View Transform.

Lighting

Perspective Transform.

Clipping

object world viewing

OCS WCS VCS

Clipping

(4D)

normalized device

NDCS

device

DCS (3D)

Scan Conversion

Texturing

Depth Test

Blending

Frame-buffer

screen

SCS

(2D)
Backface Culling
Back-Face Culling

• on the surface of a closed orientable manifold, polygons whose normals point away from the camera are always occluded:

note: backface culling alone doesn’t solve the hidden-surface problem!
Back-Face Culling

• not rendering back-facing polygons improves performance
  • by how much?
    • reduces by about half the number of polygons to be considered for each pixel
  • optimization when appropriate
Back-Face Culling

- most objects in scene are typically “solid”
- rigorously: orientable closed manifolds
  - orientable: must have two distinct sides
    - cannot self-intersect
    - a sphere is orientable since has two sides, 'inside' and 'outside'.
    - a Mobius strip or a Klein bottle is not orientable
  - closed: cannot “walk” from one side to the other
    - sphere is closed manifold
    - plane is not
Back-Face Culling

• examples of non-manifold objects:
  • a single polygon
  • a terrain or height field
  • polyhedron w/ missing face
  • anything with cracks or holes in boundary
  • one-polygon thick lampshade
Back-face Culling: VCS

first idea:
\[ cull \text{ if } N_Z < 0 \]

sometimes misses polygons that should be culled
Back-face Culling: NDCS

works to cull if $N_Z > 0$
Invisible Primitives

- **why might a polygon be invisible?**
  - polygon outside the *field of view / frustum*
    - solved by *clipping*
  - polygon is *backfacing*
    - solved by *backface culling*
  - polygon is *occluded* by object(s) nearer the viewpoint
    - solved by *hidden surface removal*
Blending
Rendering Pipeline

1. Geometry Database
3. Lighting
4. Perspective Transform.
5. Clipping
6. Scan Conversion
7. Texturing
8. Depth Test
9. Blending
10. Frame-buffer
Blending/Compositing

- how might you combine multiple elements?
- foreground color \( A \), background color \( B \)
Premultiplying Colors

- specify opacity with alpha channel: (r,g,b,α)
  - α=1: opaque, α=.5: translucent, α=0: transparent

- A over B
  - C = αA + (1-α)B

- but what if B is also partially transparent?
  - C = αA + (1-α)βB = βB + αA + βB - α βB
  - γ = β + (1-β)α = β + α - αβ
    - 3 multiplies, different equations for alpha vs. RGB

- premultiplying by alpha
  - C’ = γ C, B’ = βB, A’ = αA
  - C’ = B’ + A’ - αB’
  - γ = β + α - αβ
    - 1 multiply to find C, same equations for alpha and RGB
Texturing
Rendering Pipeline

Geometry Processing

Geometry Database -> Model/View Transform. -> Lighting -> Perspective Transform. -> Clipping

Scan Conversion -> Texturing -> Depth Test -> Blending

Rasterization

Fragment Processing

Frame-buffer
Texture Mapping

- real life objects have nonuniform colors, normals
- 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
Texture Coordinates

- texture image: 2D array of color values (texels)
- assigning texture coordinates \((s,t)\) at vertex with object coordinates \((x,y,z,w)\)
  - use interpolated \((s,t)\) for texel lookup at each pixel
  - use value to modify a polygon’s color
    - or other surface property
- specified by programmer or artist
  
\[
glTexCoord2f(s,t) \\
glVertexf(x,y,z,w)
\]

![Texture Coordinates Example](image)
Texture Mapping Example
Example Texture Map

- \( \text{glTexCoord2d}(0,0); \)
  \( \text{glVertex3d} (0, -2, -2); \)

- \( \text{glTexCoord2d}(1,1); \)
  \( \text{glVertex3d} (0, 2, 2); \)

- \( \text{glTexCoord2d}(0,0); \)
  \( \text{glVertex3d} (0, -2, -2); \)
Fractional Texture Coordinates

(texture image)

(0,1) (1,1)
(0,0) (1,0)

(0,.5) (.25,.5)
(0,0) (.25,0)
Texture Lookup: Tiling and Clamping

- what if s or t is outside the interval [0…1]?
- multiple choices
  - use fractional part of texture coordinates
    - cyclic repetition of texture to tile whole surface
      ```
      glTexParameteri( ..., GL_TEXTURE_WRAP_S, GL_REPEAT, GL_TEXTURE_WRAP_T, GL_REPEAT, ... )
      ```
  - clamp every component to range [0…1]
    - re-use color values from texture image border
      ```
      glTexParameteri( ..., GL_TEXTURE_WRAP_S, GL_CLAMP, GL_TEXTURE_WRAP_T, GL_CLAMP, ... )
      ```
Tiled Texture Map

```plaintext
glTexCoord2d(1, 1);
glVertex3d (x, y, z);  

(1,0) + (0,0) = (1,1)

Texture
Object
Mapped Texture

---

glTexCoord2d(4, 4);
glVertex3d (x, y, z);  

(4,0) + (0,0) = (4,4)

Tex
(0,0)
(0,4) Mapped Texture
```
Demo

• Nate Robbins tutors
  • texture
Texture Coordinate Transformation

• motivation
  • change scale, orientation of texture on an object

• approach
  • *texture matrix stack*
  • transforms specified (or generated) tex coords
    ```
    glMatrixMode( GL_TEXTURE );
    glLoadIdentity();
    glRotate();
    ...
    ```
  • more flexible than changing (s,t) coordinates

• [demo]
Texture Functions

- once have value from the texture map, can:
  - directly use as surface color: GL_REPLACE
    - throw away old color, lose lighting effects
  - modulate surface color: GL_MODULATE
    - multiply old color by new value, keep lighting info
    - texturing happens after lighting, not relit
  - use as surface color, modulate alpha: GL_DECAL
    - like replace, but supports texture transparency
  - blend surface color with another: GL_BLEND
    - new value controls which of 2 colors to use
    - indirection, new value not used directly for coloring

- specify with `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, <mode>)`

- [demo]
Texture Pipeline

(x, y, z)
Object position
(-2.3, 7.1, 17.7)

(s, t)
Parameter space
(0.32, 0.29)

(s’, t’)
Transformed parameter space
(0.52, 0.49)

Texel space
(81, 74)

Texel color
(0.9, 0.8, 0.7)

Object color
(0.5, 0.5, 0.5)

Final color
(0.45, 0.4, 0.35)
Texture Objects and Binding

- texture object
  - an OpenGL data type that keeps textures resident in memory and provides identifiers to easily access them
  - provides efficiency gains over having to repeatedly load and reload a texture
  - you can prioritize textures to keep in memory
  - OpenGL uses least recently used (LRU) if no priority is assigned

- texture binding
  - which texture to use right now
  - switch between preloaded textures
Basic OpenGL Texturing

- create a texture object and fill it with texture data:
  - `glGenTextures(num, &indices)` to get identifiers for the objects
  - `glBindTexture(GL_TEXTURE_2D, identifier)` to bind
    - following texture commands refer to the bound texture
  - `glTexParameteri(GL_TEXTURE_2D, …, …)` to specify parameters for use when applying the texture
  - `glTexImage2D(GL_TEXTURE_2D, …)` to specify the texture data (the image itself)
- enable texturing: `glEnable(GL_TEXTURE_2D)`
- state how the texture will be used:
  - `glTexEnvf(…)`
- specify texture coordinates for the polygon:
  - use `glTexCoord2f(s, t)` before each vertex:
    - `glTexCoord2f(0, 0); glVertex3f(x, y, z);`
Low-Level Details

• large range of functions for controlling layout of texture data
  • state how the data in your image is arranged
  • e.g.: `glPixelStorei(GL_UNPACK_ALIGNMENT, 1)` tells OpenGL not to skip bytes at the end of a row
  • you must state how you want the texture to be put in memory: how many bits per “pixel”, which channels,…

• textures must be square and size a power of 2
  • common sizes are 32x32, 64x64, 256x256
  • smaller uses less memory, and there is a finite amount of texture memory on graphics cards

• ok to use texture template sample code for project 4
  • http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=09
Texture Mapping

- texture coordinates
  - specified at vertices
    
    ```
    glTexCoord2f(s,t);
    glVertexf(x,y,z);
    ```
  - interpolated across triangle (like R,G,B,Z)
    - …well not quite!
Texture Mapping

• texture coordinate interpolation
  • perspective foreshortening problem
Interpolation: Screen vs. World Space

- screen space interpolation incorrect
  - problem ignored with shading, but artifacts more visible with texturing
Texture Coordinate Interpolation

- perspective correct interpolation
  - $\alpha$, $\beta$, $\gamma$:
    - barycentric coordinates of a point $P$ in a triangle
  - $s_0$, $s_1$, $s_2$:
    - texture coordinates of vertices
  - $w_0$, $w_1$, $w_2$:
    - homogeneous coordinates of vertices

\[
\begin{align*}
\alpha \cdot \frac{s_0}{w_0} + \beta \cdot \frac{s_1}{w_1} + \gamma \cdot \frac{s_2}{w_2} = \frac{\alpha}{w_0} + \frac{\beta}{w_1} + \frac{\gamma}{w_2}
\end{align*}
\]
Reconstruction

(image courtesy of Kiriakos Kutulakos, U Rochester)
Reconstruction

• how to deal with:
  • pixels that are much larger than texels?
    • apply filtering, “averaging”
  • pixels that are much smaller than texels?
    • interpolate
MIPmapping

use “image pyramid” to precompute averaged versions of the texture

store whole pyramid in single block of memory

Without MIP-mapping

With MIP-mapping
MIPmaps

- *multum in parvo* -- many things in a small place
  - prespecify a series of prefiltred texture maps of decreasing resolutions
  - requires more texture storage
  - avoid shimmering and flashing as objects move
- `gluBuild2DMipmaps`
  - automatically constructs a family of textures from original texture size down to 1x1
MIPmap storage

- only 1/3 more space required
Texture Parameters

- in addition to color can control other material/object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)
Bump Mapping: Normals As Texture

- object surface often not smooth – to recreate correctly need complex geometry model
- can control shape “effect” by locally perturbing surface normal
  - random perturbation
  - directional change over region
Bump Mapping

Original surface

A bump map
Bump Mapping

\[ O'(u) \]
Lengthening or shortening \( O(u) \) using \( B(u) \)

\[ N'(u) \]
The vectors to the ‘new’ surface
Embossing

- at transitions
  - rotate point’s surface normal by $\theta$ or $-\theta$
Displacement Mapping

• bump mapping gets silhouettes wrong
  • shadows wrong too
• change surface geometry instead
  • only recently available with realtime graphics
  • need to subdivide surface