Alpha Blending
Double Buffering

Wolfgang Heidrich

Course News

Assignment 2
• Due Monday!

Quiz 2 MOVED!
• Friday, March 13 (instead of Wed, March 11)
• Office hours on Wednesday, Thursday (Mar 11/12)
• Out of town Mon, Mar 9
  – Office hour canceled
  – Lecture will take place

Reading
• No new reading this week
The Rendering Pipeline

Geometry Database → Model/View Transform → Lighting → Perspective Transform → Clipping

Geometry Processing

Scan Conversion → Texturing → Depth Test → Blending → Framebuffer

Rasterization → Fragment Processing

Creating BSP Trees: Objects
Splitting Objects

No bunnies were harmed in previous example
But what if a splitting plane passes through an object?
• Split the object; give half to each node

Traversing BSP Trees

Tree creation independent of viewpoint
• Preprocessing step

Tree traversal uses viewpoint
• Runtime, happens for many different viewpoints

Each plane divides world into near and far
• For given viewpoint, decide which side is near and which is far
  • Check which side of plane viewpoint is on independently for each tree vertex
  • Tree traversal differs depending on viewpoint!
• Recursive algorithm
  • Recurse on far side
  • Draw object
  • Recurse on near side
BSP Trees: Viewpoint A

- decide independently at each tree vertex
- not just left or right child!
BSP Trees: Viewpoint A

BSP Trees: Viewpoint B
BSP Trees: Viewpoint B

BSP Tree Traversal: Polygons

- Split along the plane defined by any polygon from scene
- Classify all polygons into positive or negative half-space of the plane
  - *If a polygon intersects plane, split polygon into two and classify them both*
- Recurse down the negative half-space
- Recurse down the positive half-space
BSP Demo

Useful demo:
http://symbolcraft.com/graphics/bsp

Summary: BSP Trees

Pros:
- Simple, elegant scheme
- Correct version of painter’s algorithm back-to-front rendering approach
- Still very popular for video games (but getting less so)

Cons:
- Slow(ish) to construct tree: $O(n \log n)$ to split, sort
- Splitting increases polygon count: $O(n^2)$ worst-case
- Computationally intense preprocessing stage restricts algorithm to static scenes
The Rendering Pipeline

Geometry Processing

Geometry Database → Model/View Transform. → Lighting → Perspective Transform → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer

Rasterization → Fragment Processing

Blending

*How might you combine multiple elements?*

- New color A, old color B

[Images of blending results: A over B, A in B, A out B, A atop B, A xor B]
**Alpha Blending (OpenGL)**

**Parameters:**
- \( s = \) source color
- \( d = \) destination color
- \( b = \) source blend factor
- \( c = \) dest blend factor
- \( d' = bs + cd \)

**Where**
- "Source" means "color/alpha of currently rendered primitive"
- "Destination" means framebuffer value

**Over operator**
- \( d' = \alpha_s s + (1-\alpha_s)d \)
- Examples: \( \alpha_A = 1 \), \( \alpha_B = 0.4 \)
Over operator

- \[ d' = \alpha_s s + (1-\alpha_s)d \]
- Examples: \( \alpha_A = 0.4 \), \( \alpha_B = 1.0 \)

Comparison from previous

\begin{align*}
\text{A over B: } & d' = 0.4*C_A \\
& + (0.6)*C_B \\
\text{B over A: } & d' = 1*C_B \\
& + (0)*C_A
\end{align*}
OpenGL Blending

**In OpenGL:**

- Enable blending
  - `glEnable( GL_BLEND )`
- Specify alpha channel for colors
  - `glColor4f( r, g, b, alpha )`
- Specify blending function
  - *E.g:* `glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA )`
  - $C = \alpha_{\text{new}} C_{\text{new}} + (1-\alpha_{\text{new}})C_{\text{old}}$

**Caveats:**

- Note: alpha blending is an order-dependent operation!
  - *It matters which object is drawn first AND Which surface is in front*
- For 3D scenes, this makes it necessary to keep track of rendering order explicitly
  - *Possibly also viewpoint-dependent!*
    - *E.g.* always draw “back” surface first
- Also note: interaction with z-buffer
Double Buffering

**Framebuffer:**
- Piece of memory where the final image is written
- Problem:
  - *The display needs to read the contents, cyclically, while the GPU is already working on the next frame*
  - *Could result in display of partially rendered images on screen*
- Solution:
  - *Have TWO buffers*
    - Currently displayed (front buffer)
    - Render target for the next frame (back buffer)
Double Buffering

**Front/back buffer:**
- Each buffer has both color channels and a depth channel
  - Important for advanced rendering algorithms
  - Doubles memory requirements!

**Switching buffers:**
- At end of rendering one frame, simply exchange the pointers to the front and back buffer
- GLUT toolkit: glutSwapBuffers() function
  - Different functions under windows/X11 if not using GLUT

Triple Buffering

**Used by some game consoles**
- Why?
Coming Up:

*Friday / next week*
- Texture mapping