Misc. Rendering Pipeline Topics
Blending, Double Buffer, Picking

CPSC 314

The Rendering Pipeline

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

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

Geometry Processing

Rasterization

Fragment Processing
Depth Test / Hidden Surface Removal

• For most interesting scenes, some polygons overlap

  ![Diagram showing overlapping polygons]

• To render the correct image, we need to determine which polygons occlude which

Hidden Surface Removal

**Object Space Methods:**

• Work in 3D before scan conversion
  – *E.g. Painter’s algorithm*
• Usually independent of resolution
  – *Important to maintain independence of output device (screen/printer etc.)*

**Image Space Methods:**

• Work on per-pixel/per fragment basis after scan conversion
• Z-Buffer/Depth Buffer
• Much faster, but resolution dependent
The Z-Buffer Algorithm

Augment color framebuffer with Z-buffer

- Also called depth buffer
- Stores z value at each pixel
- At frame beginning, initialize all pixel depths to $\infty$
- When scan converting: 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

Store (r,g,b,z) for each pixel

- typically 8+8+8+24 bits, can be more
  for all i,j {
    Depth[i,j] = MAX_DEPTH
    Image[i,j] = BACKGROUND_COLOUR
  }
  for all polygons P {
    for all pixels in P {
      if (Z_pixel < Depth[i,j]) {
        Image[i,j] = C_pixel
        Depth[i,j] = Z_pixel
      }
    }
  }
Depth Test Precision

• Reminder: projective transformation maps eye-space z to generic z-range (NDC)
• Simple example:

\[
T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

• Thus:

\[
z_{NDC} = \frac{a \cdot z_{eye} + b}{z_{eye}} = a + \frac{b}{z_{eye}}
\]

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
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

Object Space Visibility Algorithms

What is the minimum worst-case cost of computing the fragments for a scene composed of \( n \) polygons?

**Answer:** \( O(n^2) \)
Binary Space Partition Trees (1979)

**BSP Tree: partition space with binary tree of planes**

- Idea: divide space recursively into half-spaces by choosing splitting planes that separate objects in scene
- Preprocessing: create binary tree of planes
- Runtime: correctly traversing this tree enumerates objects from back to front

Creating BSP Trees: Objects
Creating BSP Trees: Objects
BSP Trees: Viewpoint A

© Wolfgang Heidrich
BSP Trees: Viewpoint A

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

© Wolfgang Heidrich
BSP Trees: Viewpoint A

The Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping → Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer

Geometry Processing

Rasterization → Fragment Processing
How might you combine multiple elements?

- New color A, old color B
Premultiplying Colors

Specify opacity with alpha channel: \((r,g,b,\alpha)\)
- \(\alpha=1\): opaque, \(\alpha=0.5\): translucent, \(\alpha=0\): transparent

A over B
- \(C = \alpha A + (1-\alpha)B\)

But what if B is also partially transparent?
- \(C = \alpha A + (1-\alpha)B = \alpha A + \gamma B - \alpha \beta B\)
- \(\gamma = \beta + (1-\beta)\alpha = \beta + \alpha - \alpha\beta\)
- 3 multiplies, different equations for alpha vs. RGB

Premultiplying by alpha
- \(C' = \gamma C, B' = \beta B, A' = \alpha A\)

- \(C' = B' + A' - \alpha B'\)
- \(\gamma = \beta + \alpha - \alpha\beta\)
- 1 multiply to find \(C\), same equations for alpha and RGB

OpenGL Blending

In OpenGL:
- Enable blending
  - \texttt{glEnable( GL\_BLEND )}
- Specify alpha channel for colors
  - \texttt{glColor4f( r, g, b, alpha )}
- Specify blending function
  - E.g: \texttt{glBlendFunc( GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA )}
  - \(C = \text{alpha\_new}\ast C\text{new} + (1\text{-alpha\_new})\ast \text{Cold}\)
OpenGL Blending

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 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
    - One is currently displayed (front buffer)
    - One is rendered into for the next frame (back buffer)

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
Picking/Object Selection

Interactive Object Selection

*Move cursor over object, click*
- How to decide what is below?

*Ambiguity*
- Many 3D world objects map to same 2D point

*Common approaches*
- Manual ray intersection
- Bounding extents
- Selection region with hit list (OpenGL support)
Manual Ray Intersection

*Do all computation at application level*
- Map selection point to a ray
- Intersect ray with all objects in scene.

**Advantages**
- No library dependence

---

**Disadvantages**
- Difficult to program
- Slow: work to do depends on total number and complexity of objects in scene
Bounding Extents

*Keep track of axis-aligned bounding rectangles*

### Advantages
- Conceptually simple
- Easy to keep track of boxes in world space

### Disadvantages
- Low precision
- Must keep track of object-rectangle relationship

### Extensions
- Do more sophisticated bound bookkeeping
  - *First level: box check. second level: object check*
OpenGL Picking

“Render” image in picking mode

- Pixels are never written to framebuffer
- Only store IDs of objects that would have been drawn

Procedure

- Set unique ID for each pickable object
- Call the regular sequence of glBegin/Vertex/.glEnd commands
  - If possible, skip glColor, glNormal, glTexCoord etc. for performance

Select/Hit

OpenGL support

- Use small region around cursor for viewport
- Assign per-object integer keys (names)
- Redraw in special mode
- Store hit list of objects in region
- Examine hit list
**Viewport**

**Small rectangle around cursor**
- Change coord sys so fills viewport

![Small rectangle around cursor](image)

**Why rectangle instead of point?**
- People aren’t great at positioning mouse
  - Fitts’s Law: time to acquire a target is function of the distance to and size of the target
- Allow several pixels of slop

**Viewport**

**Tricky to compute**
- Invert viewport matrix, set up new orthogonal projection

**Simple utility command**
- `gluPickMatrix(x,y,w,h,viewport)`
  - `x,y`: cursor point
  - `w,h`: sensitivity/slop (in pixels)
- Push old setup first, so can pop it later
Render Modes

`glRenderMode(mode)`

- GL_RENDER: normal color buffer
  - default
- GL_SELECT: selection mode for picking
- (GL_FEEDBACK: report objects drawn)

Name Stack

- “names” are just integers
  `glInitNames()`
- flat list
  `glLoadName(name)`
- or hierarchy supported by stack
  `glPushName(name), glPopName`
  - Can have multiple names per object
  - Helpful for identifying objects in a hierarchy
Hierarchical Names Example

```java
for(int i = 0; i < 2; i++) {
    glPushName(i);
    for(int j = 0; j < 2; j++) {
        glPushMatrix();
        glPushName(j);
        glTranslate(i*10.0,j*10.0);
        glPushName(HEAD);
        glCallList(snowManHeadDL);
        glLoadName(BODY);
        glCallList(snowManBodyDL);
        glPopName();
        glPopName();
        glPopMatrix();
    }
    glPopName();
}
```

http://www.lighthouse3d.com/opengl/picking/

Hit List

- `glSelectBuffer(int buffersize, GLuint *buffer)`
  - *Where to store hit list data*
- If object overlaps with pick region, create **hit record**
- Hit record
  - *Number of names on stack*
  - *Minimum and minimum depth of object vertices*
    - Depth lies in the z-buffer range [0,1]
    - Multiplied by $2^{32}$ -1 then rounded to nearest int
  - *Contents of name stack (bottom entry first)*
Using OpenGL Picking

**Example code:**

```c
int numHitEntries;
GLuint buffer[1000];
glSelectBuffer( 1000, buffer );
glRenderMode( GL_SELECT );
drawStuff(); // includes name stack calls
numHitEntries= glRenderMode( GL_RENDER );
// now analyze numHitEntries different hit records
// in the selection buffer
...
```

Integrated vs. Separate Pick Function

**Integrate: use same function to draw and pick**
- Simpler to code
- Name stack commands ignored in render mode

**Separate: customize functions for each**
- Potentially more efficient
- Can avoid drawing unpickable objects
Select/Hit

**Advantages**
- Faster
  - OpenGL support means hardware accel
  - Only do clipping work, no shading or rasterization
- Flexible precision
  - Size of region controllable
- Flexible architecture
  - Custom code possible, e.g. guaranteed frame rate

**Disadvantages**
- More complex

Coming Up...

**Tuesday:**
- Texture Mapping

**Thursday:**
- Sampling