Scan Conversion & Hidden Surface Removal

4-02-15
Outline

- From Geometry to Pixels:
  - Scan conversion
  - Antialiasing
  - Hidden Surface Removal

Read:
- Angel, Chapter 8, 8.8 - 8.12

Project#2 posted due: April 23rd
Recap: Rasterizing a Line
Bresenham’s Algorithm

- Decision variable $a - b$
  - If $a - b > 0$ choose lower pixel
  - If $a - b \leq 0$ choose higher pixel
- Goal: avoid explicit computation of $a - b$
- Step 1: re-scale $d = (x_2 - x_1)(a - b) = \Delta x (a - b)$
- $d$ is always integer
Bresenham’s Algorithm

- Compute $d$ at step $k + 1$ from $d$ at step $k$!
- Case: $j$ did not change ($d_k > 0$)
  - $a$ decreases by $m$, $b$ increases by $m$
  - $(a - b)$ decreases by $2m = 2(\Delta y/\Delta x)$
  - $\Delta x(a-b)$ decreases by $2\Delta y$
Bresenham’s Algorithm

Case: j did change ($d_k \leq 0$)
- a decreases by $m-1$, b increases by $m-1$
- $(a - b)$ decreases by $2m - 2 = 2(\Delta y/\Delta x - 1)$
- $\Delta x(a-b)$ decreases by $2(\Delta y - \Delta x)$
Bresenham’s Algorithm

- So $d_{k+1} = d_k - 2\Delta y$ if $d_k > 0$
- And $d_{k+1} = d_k - 2(\Delta y - \Delta x)$ if $d_k \leq 0$
- Final (efficient) implementation:

```c
void draw_line(int x1, int y1, int x2, int y2) {
    int x, y = y0;
    int dx = 2*(x2-x1), dy = 2*(y2-y1);
    int dydx = dy-dx, D = (dy-dx)/2;

    for (x = x1 ; x <= x2 ; x++) {
        write_pixel(x, y, color);
        if (D > 0) D -= dy;
        else {y++; D -= dydx;}
    }
}
```
Filling in the Frame Buffer

- Fill at end of pipeline
  - Convex Polygons only
  - Nonconvex polygons assumed to have been tessellated (efficient algorithms, e.g. 8.10.2)
  - Shades (colors) have been computed for vertices (Gouraud shading)
  - Combine with z-buffer algorithm
  - March across scan lines interpolating shades
  - Incremental work small
Polygon Scan Conversion

- Scan Conversion = Fill
- How to tell inside from outside
  - Convex easy
  - Nonsimple difficult
- Many algorithms, including
  - Odd even test
    - count edge crossings
  - Winding number
    - compute encirclements
Winding Number

- Count clockwise encirclements of point

  winding number = 1

  winding number = 2

- Alternate definition of inside: inside if winding number $\neq 0$
Fill and Sort

- Polygon processor - black box whose inputs are the vertices for a set of 2D polygons, and whose output is a frame buffer with the correct pixels set.
- If a point is inside a polygon, color it with the inside (fill) color => sort the pixels in the frame buffer into those that are inside the polygon and those that are not.
- Different polygon fill algorithms according to different ways of sorting the points:
  - Flood Fill
  - Scan line fill
  - Odd-even fill
Flood Fill

- Fill can be done recursively if we know a seed point located inside (WHITE)
- Scan convert edges into buffer in edge/inside color (BLACK)

```c
flood_fill(int x, int y) {
    if(read_pixel(x,y)== WHITE) {
        write_pixel(x,y,BLACK);
        flood_fill(x-1, y);
        flood_fill(x+1, y);
        flood_fill(x, y+1);
        flood_fill(x, y-1);
    }
}
```
Using Interpolation

$C_1 C_2 C_3$ specified by `glColor` or by vertex shading

$C_4$ determined by interpolating between $C_1$ and $C_2$

$C_5$ determined by interpolating between $C_2$ and $C_3$

interpolate between $C_4$ and $C_5$ along span

scan line

span (in red)
Scan Line Fill

- Can also fill by maintaining a data structure of all intersections of polygons with scan lines
  - Sort by scan line
  - Fill each span

[Diagram showing vertex order generated by vertex list and desired order]

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Data Structure

Scanlines

Intersections

\[ j \rightarrow x_1 \rightarrow x_2 \]

\[ j + 1 \rightarrow x_3 \rightarrow x_4 \]

\[ j + 2 \rightarrow x_4 \rightarrow x_5 \rightarrow x_7 \rightarrow x_8 \]
Outline

- Rasterization:
  - Scan conversion for Lines
  - Scan conversion for Polygons
- Antialiasing
Aliasing

- Ideal rasterized line should be 1 pixel wide

- Choosing best y for each x (or visa versa) produces aliased raster lines
Antialiasing by Area Averaging

- Color multiple pixels for each x depending on coverage by ideal line

original

antialiased

magnified

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Aliasing problems can be serious for polygons

- Jaggedness of edges
- Small polygons neglected
- Need compositing so color of one polygon does not totally determine color of pixel

All three polygons should contribute to color
Aliasing

- Artifacts created during scan conversion
- Inevitable (going from continuous to discrete)
- Aliasing (name from digital signal processing): we sample a continuous image at grid points
- Effect
  - Jagged edges
  - Moire patterns

Moire pattern from sandlotscience.com
More Aliasing

No antialiasing
Antialiasing for Line Segments

- Use area averaging at boundary

(a) (b) (c) (d)

- (c) is aliased, magnified
- (d) is antialiased, magnified
Antialiasing by Supersampling

- Mostly for off-line rendering (e.g., ray tracing)
- Render, say, 3x3 grid of mini-pixels
- Average results using a filter
- Can be done adaptively
  - Stop if colors are similar
  - Subdivide at discontinuities
Supersampling Example

- Other improvements
  - Stochastic sampling: avoid sample position repetitions
  - Stratified sampling (jittering): perturb a regular grid of samples
Temporal Aliasing

- Sampling rate is frame rate (30 Hz for video)
- Example: spokes of wagon wheel in movies
- Solution: supersample in time and average
  - Fast-moving objects are blurred
  - Happens automatically with real hardware (photo and video cameras)
    - Exposure time is important (shutter speed)
  - Effect is called motion blur

Motion blur
Wagon Wheel Effect

Source: YouTube

wagon_wheel_video
Motion Blur Example

Achieve by stochastic sampling in time

T. Porter, Pixar, 1984
16 samples / pixel / timestep
Clipping and Visibility

- Clipping has much in common with hidden-surface removal.
- In both cases, we are trying to remove objects that are not visible to the camera.
- Often we can use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline.
Hidden Surface Removal

- Object-space approach: use pairwise testing between polygons (objects)
  - Partially obscuring
  - Can draw independently

- Worst case complexity $O(n^2)$ for $n$ polygons
Painter’s Algorithm

- Render polygons a back to front order so that polygons behind others are simply painted over.

B behind A as seen by viewer

Fill B then A

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Depth Sort

- Requires ordering of polygons first
  - $O(n \log n)$ calculation for ordering
  - Not every polygon is either in front or behind all other polygons
- Order polygons and deal with easy cases first, harder later

Polygons sorted by distance from COP
Easy Cases

- A lies behind all other polygons
  - Can render

- Polygons overlap in z but not in either x or y
  - Can render independently
Overlap in all directions but one polygon is fully on one side of the other

cyclic overlap

penetration
Back-Face Removal (Culling)

- face is visible iff $90 \geq \theta \geq -90$
  equivalently $\cos \theta \geq 0$
  or $v \cdot n \geq 0$

- plane of face has form $ax + by + cz + d = 0$
  but after normalization $n = (0 \ 0 \ 1 \ 0)^T$

- need only test the sign of $c$

- In OpenGL we can simply enable culling but it may not work correctly if we have non-convex objects
Look at each projector (\(nm\) for an \(n \times m\) frame buffer) and find closest of \(k\) polygons

- Complexity \(O(nmk)\)
- Ray tracing
- \(z\)-buffer
z-Buffer Algorithm

- Use a buffer called the z or depth buffer to store the depth of the closest object at each pixel found so far.
- As we render each polygon, compare the depth of each pixel to depth in z buffer.
- If less, place shade of pixel in color buffer and update z buffer.
If we work scan line by scan line as we move across a scan line, the depth changes satisfy
\[ a \Delta x + b \Delta y + c \Delta z = 0 \]

Along scan line
\[ \Delta y = 0 \]
\[ \Delta z = - \frac{a}{c} \Delta x \]

In screen space \( \Delta x = 1 \)
Scan-Line Algorithm

- Can combine shading and HSR (hidden surface removal) through scan line algorithm

scan line i: no need for depth information, can only be in no polygon or in one polygon

scan line j: need depth information only when in more than one polygon
Visibility Testing

- In many realtime applications, such as games, we want to eliminate as many objects as possible within the application
  - Reduce burden on pipeline
  - Reduce traffic on bus
- Partition space with Binary Spatial Partition (BSP) Tree
Simple Example

consider 6 parallel polygons

The plane of A separates B and C from D, E and F
BSP Tree (Binary Space Partition)

- Can continue recursively
  - Plane of C separates B from A
  - Plane of D separates E and F
- Can put this information in a BSP tree
  - Use for visibility and occlusion testing