2D Graphics Primitives IV

Clipping polygons

Antialiasing

Clipping Polygons

If just clip lines

Want to clip polygons
Sutherland-Hodgeman Polygon Clipper

Divide and conquer algorithm

Clipping against clip-region  
Clip against each region edge

Parallel implementation  
Pipeline implementation

Input a list of points (vertices) 
Each stage of pipeline outputs an updated list of vertices

Can generalize to any convex polygon clip region
How Clip Polygon by a Boundary?

Need concept of inside and outside of boundary

Consider a polygon described by a sequence of vertices

\[ V = (v_1, v_2, v_3, ..., v_n) \]

Want to clip against clip-region to get clipped polygon

\[ C = (c_1, c_2, c_3, ..., c_m) \]

In general will \( m = n \)?

Let \( E \) be the edge list of \( V \)

\[ E = \{ sp \mid s = v_i, p = v_{i+1} \} + \{ v_n v_1 \} \]

note we need to include last edge
For each edge sp in E

if s and p are both inside boundary, then output p

if s is inside and p is outside then output intersection, i

if s and p both outside, then output nothing

if s is outside and p is inside, then output i and p

The output list from one boundary is the input list for the next
CDQ: Hand Simulation of Sutherland-Hodgman Polygon Clipping

\[ V = (a, b, c, d, e, f, g) \]

\[ E = ? \]

Find the updated \( E \) after clipping by:

- right boundary
  \[ E = \]

- bottom boundary
  \[ E = \]

- left boundary
  \[ E = \]

- top boundary
  \[ E = \]

Is this the desired output for this polygon?
Weiler-Atherton Polygon Clipping

Solves problem just encountered

Very general
- Can clip convex polygon by a convex polygon
- Can clip a concave polygon by a convex polygon

Clip region is called the clip polygon

Polygon to be clipped is called the subject polygon

Traces clockwise around the polygons
Algorithm

position = start
save = off
trace subject (position)
    if hit clip
        if (entering clip and subject not saved)
            save = on
            trace subject(current)
        else if (leaving clip and clip right not saved))
            save = on
            push current onto stack
            trace clip(current)
        else
            save = off
    if stack empty
        end
    else
        popstack (current)
        trace subject (current)
end

trace clip (position)
    if hit subject
        if subject right not saved
            turn on save
            trace subject (current)
        else
            turn off save
            if stack empty
                end
            else
                popstack (current)
                trace subject (current)
            end
end
Antialiasing

Or: "How to overcome the Jaggies"

When scan converting lines (or filled polygons) get jaggies - small step edges

By product of digitization

Can we improve things by increasing the resolution of the display?

Not exactly

Does increase the number of distinct orientations we can display
But still a problem

Can we overcome this problem with binary displays?

How to overcome the problem with grey-scale displays
Method 1: Prefiltering

When scan converting an object, compute the value for each pixel that will diminish the perceived aliasing.

Pitteway and Watkinson came up with a modification to Bresenham's line drawing algorithm which does the calculations whilst scan converting.

Difficult to compute this for nonstraight lines.

Two main approaches:

Unweighted Area Sampling

Weighted Area Sampling

Unweighted Area Sampling:

Determine what percentage of a pixel's area is covered by the object.

Value of pixel is linear function of this percentage
- Totally covered - max value (foreground color)
- Totally uncovered - min value (background color)
- 50% covered - half way between max and min

What value for each pixel?
Weighted Area Sampling

If line passes through middle of pixel, may want to weight it more than if it passes by pixel edge

(Assume now that pixels are square tiles tesselating (tiling) a surface)

Use weighted filter function

\[ \text{filtered output} = \text{area under } f(x,y) \times g(x,y) \]

for box filter at pixel \(i,j\)

\[ f(x,y) = 1, \text{ if } i \leq x \leq i+1, \text{ and } j \leq y \leq j+1 \]
\[ = 0, \text{ otherwise} \]
Box filter is just the Unweighted Prefiltering

To give more weight at center of pixel
    Could use cone shaped filter

May want the base of the cone to be greater than one pixel in diameter

If diameter of cone base (support) = one pixel width for square pixels, then is all of the scene sampled?

If make support too small, then not all scene is sampled
If make support too large, then may blur scene too much

Support diameter = 2 pixel widths is good compromise
Method 2: Supersampling

Scan convert a grid that has more pixels than actual display

For each display pixel, compute the average value of the corresponding set of supersampled pixels

Here dashed line intersections are the displayed pixels, solid and solid-dashed intersections are supersampled pixels

With this double sampling, average display pixel and its eight-connected neighbors

(See example)

What is relation between prefiltering and supersampling?
Method 3: Postfiltering

Just as could use weighted sampling filter in prefiltering, can use weighted sampling filter in postfiltering

Supersample the scene

Instead of using straight average, use a weighted average of supersample pixels to compute value of display pixel

If double sample, could use 3x3 filter or even 5x5, 7x7 or greater.

Example filters:

\[
\begin{array}{ccc}
\frac{1}{8} & 0 & 1 \\
1 & 4 & 1 \\
0 & 1 & 0 \\
\end{array}
\quad \frac{1}{16} \quad \begin{array}{ccc}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{array}
\quad \frac{1}{81} \quad \begin{array}{ccc}
1 & 2 & 3 & 2 & 1 \\
2 & 4 & 6 & 4 & 2 \\
3 & 6 & 9 & 6 & 3 \\
2 & 4 & 6 & 4 & 2 \\
1 & 2 & 3 & 2 & 1 \\
\end{array}
\]

If quadruple sample, what would be smallest reasonable filter size?

How many extra pixels?

What is effect of using larger filter?