CS488

Polygon Clipping and Circles

Luc RENAMBOT
Previous Lectures

- Frame buffers
- Drawing a line (Midpoint Line Algorithm)
- Polygon Filling (Edge-table algorithm)
- Line Clipping (Cohen-Sutherland algorithm)
  - Polygon Clipping
  - Circles
Polygon Clipping

Clipping rectangle
Polygon Clipping

Clipping rectangle
Polygon Clipping

Clipping rectangle
Sutherland-Hodgman Polygon Clipping

- Unlike line-clipping where we selectively clipped against each edge, here we successively clip a polygon against all four edges of the clip rectangle.

- S.H. polygon-clipping algorithm uses a divide-and-conquer strategy: It solves a series of simple and identical problems that, when combined, solve the overall problem.

- The simple problem is to clip a polygon against a single infinite clip edge.
Steps

1. **Clipping rectangle**
   - **start**

2. **Right clip boundary**
   - a)

3. **Bottom clip boundary**
   - b)

4. **Top clip boundary**
   - c)

5. **Left clip boundary**
   - d)
S.H. Polygon Clipping

- a polygon with vertices V1, V2, ..., Vn
- edges between vertices Vi and Vi+1, and from Vn to V1

- (The full algorithm is given (in C) in the white book as program 3.49 on p.128)
The four types of edges are

1. Edges that are totally inside the clip window: add the second inside vertex point
2. Edges that are leaving the clip window: add the intersection point as a vertex
3. Edges that are entirely outside the clip window: add nothing
4. Edges that are entering the clip window: save the intersection and inside points as vertices
Algorithm

• For each of the four clipping edges
  • Repeatedly for each vertex \( V = V_n, V_1, V_2, \ldots V_n \)
    given an edge from vertex \( s \) to vertex \( p \)
    assume \( s \) has already been dealt with
      • If \( s \) and \( p \) are both inside the clip rectangle \( \rightarrow \) output \( p \)
      • If \( s \) is inside and \( p \) is outside the clip rectangle \( \rightarrow \) output \( i \) (the intersection of edge \( sp \) with the clip edge)
      • If \( s \) and \( p \) are both outside the clip rectangle \( \rightarrow \) output nothing
      • If \( s \) is outside and \( p \) is inside the clip rectangle \( \rightarrow \) output \( i \) (the intersection of edge \( sp \) with the clip edge) and then \( p \)
    • output edges become new set of polygon edges
Example

Clipping on Ymax edge

- 1-2 : both inside $\rightarrow$ add 2
- 2-3 : keep 2-3' and clip 3'-3 $\rightarrow$ add 3'
- 3-4 : both outside $\rightarrow$ add nothing
- 4-5 : keep 5'-5 and clip 4-5' $\rightarrow$ add 5' and 5
- 5-6 : both inside $\rightarrow$ add 6
- 6-7 : keep 6-7' and clip 7'-7 $\rightarrow$ add 7'
- 7-8 : both outside $\rightarrow$ add nothing
- 8-9 : keep 9'-9 and clip 8-9' $\rightarrow$ add 9' and 9
- 9-1 : both inside $\rightarrow$ add 1
- returning 2,3',5',5,6,7',9',9,1 as the new vertex sequence

The new vertex sequence is then checked against the Ymin edge and so on through the Xmax edge and the Xmin edge.
More Examples
The algorithm used to draw circles is very similar to the *Midpoint Line algorithm*

8 way-symmetry: for a circle centered at (0,0) and given that point (x,y) is on the circle, the following points are also on the circle:

- \((-x, y)\)  \((x, -y)\)  \((-x, -y)\)  \((y, x)\)
- \((-y, x)\)  \((y, -x)\)  \((-y, -x)\)
Definition

- Given a circle centered at (0,0) with radius R:
  - \( R^2 = X^2 + Y^2 \)
  - \( F(X,Y) = X^2 + Y^2 - R^2 \)
Symmetry

(-x,y) (x,y)
(-y,x) (y,x)
(-y,-x) (y,-x)
(-x,-y) (x,-y)

R/√2
We choose to work in the 1/8 of the circle (45 degrees) from x=0 (y-axis) to x = y = R/sqrt(2) (45 degrees clockwise from the y-axis.)

So for any point (Xi,Yi) we can plug Xi,Yi into the above equation and

- F(Xi,Yi) = 0 → (Xi,Yi) is on the circle
- F(Xi,Yi) > 0 → (Xi,Yi) is outside the circle
- F(Xi,Yi) < 0 → (Xi,Yi) is inside the circle
Steps

- Given that we have illuminated the pixel at \((X_p, Y_p)\) we will next either illuminate
  - the pixel to the EAST \((X_p + 1, Y_p)\)
  - or the pixel to the SOUTHEAST \((X_p + 1, Y_p - 1)\)
Algorithm

- We again create a decision variable $d$ set equal to the function evaluated at the midpoint
  
  $$d = F(X_p + 1, Y_p - 0.5)$$

- We plug the midpoint into the above $F()$ for the circle and see where the midpoint falls in relation to the circle.
  - $d > 0$ (midpoint is outside) $\rightarrow$ pick SOUTHEAST pixel
  - $d < 0$ (midpoint is inside) $\rightarrow$ pick EAST pixel
  - $d = 0$ (midpoint is on circle) $\rightarrow$ ***CHOOSE*** to pick SOUTHEAST pixel
Choices

Previous

Current

P \((X_p, Y_p)\)
Compute “d”

- \( d_{\text{current}} = F(X_p + 1, Y_p - 0.5) \)
- \( d_{\text{current}} = (X_p + 1)^2 + (Y_p - 0.5)^2 - R^2 \)
- \( d_{\text{current}} = X_p^2 + 2X_p + 1 + Y_p^2 - Y_p + 0.25 - R^2 \)
if the EAST pixel is chosen

- \( d_{\text{new}} = F(X_p + 2, Y_p - 0.5) \)
- \( d_{\text{new}} = (X_p + 2)^2 + (Y_p - 0.5)^2 - R^2 \)
- \( d_{\text{new}} = X_p^2 + 4X_p + 4 + Y_p^2 - Y_p + 0.25 - R^2 \)
- \( d_{\text{new}} - d_{\text{current}} = \Delta \text{E} = 2X_p + 3 \)
if the SOUTHEAST pixel is chosen

- \( d_{\text{new}} = F(X_p + 2, Y_p - 1.5) \)
- \( d_{\text{new}} = (X_p + 2)^2 + (Y_p - 1.5)^2 - R^2 \)
- \( d_{\text{new}} = X_p^2 + 4X_p + 4 + Y_p^2 - 3Y_p + 2.25 - R^2 \)
- \( d_{\text{new}} - d_{\text{current}} = \delta_{SE} = 2X_p - 2Y_p + 5 \)
First Step

- Initial point \((X_0, Y_0)\) is known to be \((0, R)\)
  - so initial \(M\) is at \((1, R - 0.5)\)
  - so initial \(d = F(1, R - 0.5)\)

\[
= 1^2 + (R - 0.5)^2 - R^2 \\
= 1 + R^2 - R + 0.25 - R^2 \\
= 1.25 - R
\]
Optimization

• Unfortunately while deltaSE and deltaE are integral, d is still a real variable, not an integer so:

  • \( h = d - 0.25 \)
  
  • \( h \) is initialized as \( 1 - R \) (instead of \( 1.25 - R \))
  
  • \( h \) is compared to as \( h < 0.25 \) (instead of \( d < 0 \))

• but since \( h \) starts off as an integer (assuming an integral \( R \)) and \( h \) is only incremented by integral amounts (deltaE and deltaSE) we can ignore the 0.25 and compare \( h < 0 \)
Final Algorithm

\[
\begin{align*}
X &= 0; \\
Y &= \text{radius}; \\
d &= 1 - \text{radius}; \\
\text{draw8Points}(X,Y); \\
\text{while}(Y > X) \\
&\quad \text{if}(d < 0) \\
&\quad\quad \text{add } 2 \times X + 3 \text{ to } d \\
&\quad\quad \text{add } 1 \text{ to } X \\
&\quad \text{else} \\
&\quad\quad \text{add } 2 \times (X-Y) + 5 \text{ to } d \\
&\quad\quad \text{add } 1 \text{ to } X \\
&\quad\quad \text{subtract } 1 \text{ from } Y \\
&\quad \text{draw8Points}(X,Y); \\
\end{align*}
\]

The full algorithm is given (in C) in the white book as figure 3.16 on p.86.
Building with Symmetry
Building with Symmetry
Building with Symmetry
Comments

- This is still somewhat bad in that there is a multiplication to compute the new value of the decision variable. The book shows a more complicated algorithm which does this multiplication only once.

- Ellipses \( F(x,y) = b^2 X^2 + a^2 Y^2 - a^2 b^2 = 0 \) are handled in a similar manner, except that 1/4 of the ellipse must be dealt with at a time and that 1/4 must be broken into 2 parts based on where the slope of the tangent to the ellipse is -1 (in first quadrant).
Filling Circles

- Circles have the advantage of being convex polygons, so each scan line will have a single span.
- The span extrema can be computed using the same method as was used to draw the pixels on the boundary of the circle, computing them for 1/8 of the circle and using 8-way symmetry to get the other 7/8. These extrema can then be stored in an array of Xmin and Ymin indexed by Y.
At this point we have discussed the primitive operations to set the contents of the frame buffer. Now we are going to go up a level of abstraction and look at how geometric transformations are used to alter the view of a 2D model: how we can translate, scale, and rotate the model, and how transformations affect what the viewport 'sees' → Geometric Transformations

I highly recommend looking over section 5.1 in the textbook which reviews matrices and vectors, before coming to next class. Along with the review in the discussion section, this should prepare you for the coming of matrices