University of British Columbia

Chapter 8

Scan Conversion (part 2)– Drawing Polygons on Raster Display

Rasterizing Polygons/Triangles

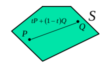
- Basic surface representation in rendering
- Why?
  - Lowest common denominator
    - Can approximate any surface with arbitrary accuracy
      - All polygons can be broken up into triangles
  - Guaranteed to be:
    - Planar
    - Triangles - Convex
  - Simple to render
    - Can implement in hardware

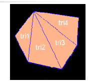
University of British Columbia

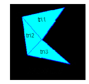
Triangulation

- Convex polygons easily triangulated
- Concave polygons present a challenge
- Convexity - formal definition:

Object S is **convex** iff for any two points $P, Q \in S$, $tP + (1-t)Q \subseteq S$, $t \in [0,1]$.



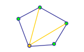


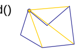


University of British Columbia

OpenGL Triangulation

- Simple convex polygons
  - break into triangles, trivial
  - glBegin(GL_POLYGON) ... glEnd()
- Concave or non-simple polygons
  - break into triangles, more effort
  - gluNewTess(), gluTessCallback(), ...

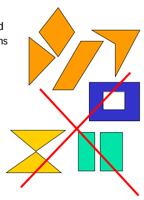




University of British Columbia

Polygon Rasterization

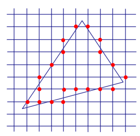
- Assumptions – well behaved
  - simple - no self intersections
  - simply connected
  - (no holes)
- Solutions
  - Flood fill
  - Scan line
  - Implicit test



University of British Columbia

Formulation

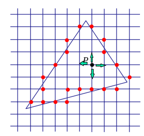
- Input
  - polygon P with rasterized edges
- Problem: Fill its interior with specified color on graphics display



University of British Columbia

Flood Fill Algorithm

- Input
  - polygon P with rasterized edges
  - $P = (x,y) \in P$ point inside P



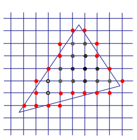
University of British Columbia

Flood Fill

FloodFill (Polygon P, int x, int y, Color C)
if not (**OnBoundary** (x, y, P) or **Colored** (x, y, C))
begin
PlotPixel (x, y, C);
FloodFill $(P, x+1, y, C)$;
FloodFill $(P, x, y+1, C)$;
FloodFill $(P, x, y-1, C)$;
FloodFill $(P, x-1, y, C)$;
end ;

University of British Columbia

Flood Fill



- Drawbacks?

University of British Columbia

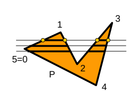
Flood Fill - Drawbacks

- How do we find a point inside?
- Pixels visited up to 4 times to check if already set
- Need per-pixel flag indicating if set already
  - clear for every polygon!

University of British Columbia

Scanline Algorithm

- Observation: Each intersection of straight line with boundary moves it from/into polygon
- Detect (& set) pixels inside polygon boundary (simple closed curve) with set of horizontal lines (pixel apart)



University of British Columbia

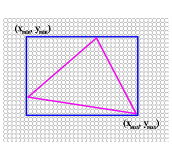
Scanline

ScanConvert (Polygon P, Color C)
For $y := 0$ to ScreenYMax do
$I \Leftarrow$ Points of intersections of edges of P with line $Y = y$;
Sort I in increasing X order and
Fill with color C alternating segments ;
end ;

- Limit to *bounding box* to speed up
- Other enhancements....

University of British Columbia

Bounding Box

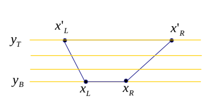


University of British Columbia

Edge Walking

- Scanline is more efficient for specific polygons – trapezoids (triangles)

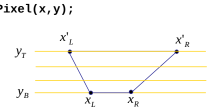
scanTrapezoid$(x_L, x_R, y_B, y_T, x'_L, x'_R)$



University of British Columbia

Edge Walking

```
for (y=yB; y<=yT; y++) {
  xl = intersect(Y=y,(xL,x'L));
  xr = intersect(Y=y,(xR,x'R));
  for (x=xl; x<=xl; x++)
    setPixel(x,y);
}
```

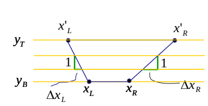


University of British Columbia

Edge Walking

- Exploit continuous L and R edges

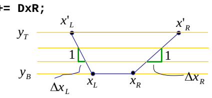
scanTrapezoid$(x_L, x_R, y_B, y_T, \Delta x_L, \Delta x_R)$



University of British Columbia

Edge Walking

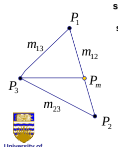
```
for (y=yB; y<=yT; y++) {
  for (x=xL; x<=xR; x++)
    setPixel(x,y);
  xL += DxL;
  xR += DxR;
}
```



University of British Columbia

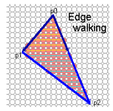
Edge Walking Triangles

- Split triangles into two regions with continuous left and right edges



scanTrapezoid$(x_3, x_m, y_3, y_1, \frac{1}{m_{13}}, \frac{1}{m_{12}})$

scanTrapezoid$(x_2, x_2, y_2, y_3, \frac{1}{m_{23}}, \frac{1}{m_{12}})$



University of British Columbia

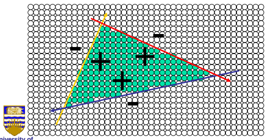
Edge Walking Triangles

- Issues
  - Many small triangles
    - setup cost is non-trivial
  - Clipping triangles produces non-triangles

University of British Columbia

Modern Rasterization

- Define a triangle from implicit edge equations:



University of British Columbia

Computing Edge Equations

- Computing A,B,C from (x_1, y_1), (x_2, y_2)

$$Ax_1 + By_1 + C = 0$$
$$Ax_2 + By_2 + C = 0$$

- Two equations, three unknowns
- Express ***A, B*** in terms of ***C***

University of British Columbia

Computing Edge Equations

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = -C \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$A = \frac{-C - By_1}{x_1}$$
$$B(x_1 y_2 - x_2 y_1) = C(x_2 - x_1)$$
(special case if $x_1 = 0$)

- Choose $C = x_2 y_1 - x_1 y_2$ for convenience
- Then $A = y_2 - y_1$ and $B = x_1 - x_2$
  - Our original implicit formula
- Note – in literature you can find same equation multiplied by -1
  - Changes sides

University of British Columbia

Edge Equations

- Given P_0, P_1, P_2, what are our three edges?
- *Half-spaces defined by the edge equations must share the same sign on the interior of the triangle*
  - Consistency (Ex: $[P_0 P_1], [P_1 P_2], [P_2 P_0]$)
- *How do we make sure that sign is positive?*
  - Test & flip if needed (***A= -A, B= -B, C= -C***)

University of British Columbia

Edge Equations: Code

- Basic structure of code:
  - Setup: compute edge equations, bounding box
  - (Outer loop) For each scanline in bounding box...
  - (Inner loop) ...check each pixel on scanline:
    - evaluate edge equations
    - draw pixel if all three are positive

University of British Columbia

Edge Equations: Code

```
findBoundingBox(&xmin, &xmax, &ymin, &ymax);
setupEdges (&a0,&b0,&c0,&a1,&b1,&c1,&a2,&b2,&c2);

for (int y = yMin; y <= yMax; y++) {
        for (int x = xMin; x <= xMax; x++) {
                float e0 = a0*x + b0*y + c0;
                float e1 = a1*x + b1*y + c1;
                float e2 = a2*x + b2*y + c2;
                if (e0 > 0 && e1 > 0 && e2 > 0)
                                Image[x][y] = TriangleColor;
        }
}
```

University of British Columbia

Edge Equations: Code

```
  // more efficient inner loop
for (int y = yMin; y <= yMax; y++) {
        float e0 = a0*xMin + b0*y + c0;
        float e1 = a1*xMin + b1*y + c1;
        float e2 = a2*xMin + b2*y + c2;
        for (int x = xMin; x <= xMax; x++) {
                if (e0 > 0 && e1 > 0 && e2 > 0)
                        Image[x][y] = TriangleColor;
                e0 += a0;   e1+= a1;    e2 += a2;
        }
}
```

University of British Columbia

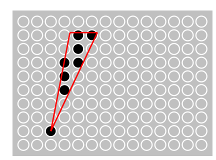
Triangle Rasterization Issues

- *Exactly which pixels should be lit?*
  - Pixels inside triangle edges
- *What about pixels exactly on the edge?*
  - Draw - BUT order of triangles matters (it shouldn't)
  - Don't draw - BUT gaps possible between triangles
- Need consistent (if arbitrary) rule
  - Example: draw pixels on left or top edge, but not on right or bottom edge

University of British Columbia

Triangle Rasterization Issues

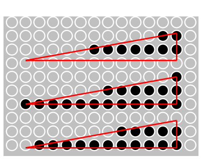
- Sliver



University of British Columbia

Triangle Rasterization Issues

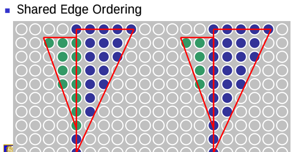
- Moving Slivers



University of British Columbia

Triangle Rasterization Issues

- Shared Edge Ordering



University of British Columbia

Interpolation – access triangle interior

- Interpolate between vertices:
  - z
  - r,g,b - colour components
  - u,v - texture coordinates
  - N_x, N_y, N_z - surface normals
- Equivalent
  - Bilinear interpolation
  - Barycentric coordinates

University of British Columbia

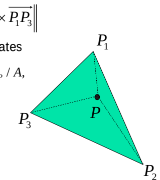
Barycentric Coordinates

- Area

$$A = \frac{1}{2} \left\| \overrightarrow{P_1 P_2} \times \overrightarrow{P_1 P_3} \right\|$$

- Barycentric coordinates

$$a_1 = A_{P_2 P_3 P} / A, a_2 = A_{P_3 P_1 P} / A,$$
$$a_3 = A_{P_1 P_2 P} / A,$$
$$P = a_1 P_1 + a_2 P_2 + a_3 P_3$$

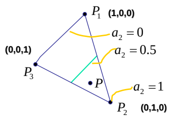


University of British Columbia

Barycentric Coordinates

- weighted combination of vertices

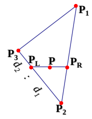
$$P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3$$
$$a_1 + a_2 + a_3 = 1$$
$$0 \le a_1, a_2, a_3 \le 1$$



University of British Columbia

Barycentric Coords:Alternative formula

- For point P on scanline:

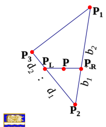


$$P_L = P_2 + \frac{d_1}{d_1 + d_2}(P_3 - P_2)$$
$$= (1 - \frac{d_1}{d_1 + d_2})P_2 + \frac{d_1}{d_1 + d_2}P_3 =$$
$$= \frac{d_2}{d_1 + d_2}P_2 + \frac{d_1}{d_1 + d_2}P_3$$

University of British Columbia

Computing Barycentric Coords

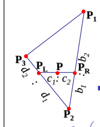
- similarly:



$$P_R = P_2 + \frac{b_1}{b_1 + b_2}(P_1 - P_2)$$
$$= (1 - \frac{b_1}{b_1 + b_2})P_2 + \frac{b_1}{b_1 + b_2}P_1 =$$
$$= \frac{b_2}{b_1 + b_2}P_2 + \frac{b_1}{b_1 + b_2}P_1$$

University of British Columbia

Computing Barycentric Coords



- combining

$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$
$$P_L = \frac{d_2}{d_1 + d_2}P_2 + \frac{d_1}{d_1 + d_2}P_3$$
$$P_R = \frac{b_2}{b_1 + b_2}P_2 + \frac{b_1}{b_1 + b_2}P_1$$

- gives

$$P = \frac{c_2}{c_1 + c_2}\left(\frac{d_2}{d_1 + d_2}P_2 + \frac{d_1}{d_1 + d_2}P_3\right) + \frac{c_1}{c_1 + c_2}\left(\frac{b_2}{b_1 + b_2}P_2 + \frac{b_1}{b_1 + b_2}P_1\right)$$

University of British Columbia

Computing Barycentric Coords

- thus

$$P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3$$

with

$$a_1 = \frac{c_1}{c_1 + c_2}\frac{b_1}{b_1 + b_2}$$
$$a_2 = \frac{c_2}{c_1 + c_2}\frac{d_2}{d_1 + d_2} + \frac{c_1}{c_1 + c_2}\frac{b_2}{b_1 + b_2}$$
$$a_3 = \frac{c_2}{c_1 + c_2}\frac{d_1}{d_1 + d_2}$$

University of British Columbia

Computing Barycentric Coords

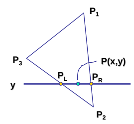
- Can verify barycentric properties

$$a_1 + a_2 + a_3 = 1$$
$$0 \le a_1, a_2, a_3 \le 1$$

University of British Columbia

Bilinear Interpolation

- Interpolate quantity along L and R edges, as a function of y
  - then interpolate quantity as a function of x



University of British Columbia