



University of
British Columbia

Chapter 5



Scan Conversion – Drawing Polygons on Raster Display



Rasterizing Polygons/Triangles


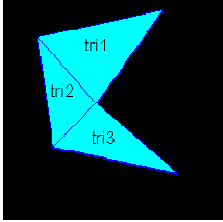
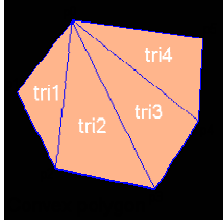
- Basic surface representation in rendering
- Why?
 - Lowest common denominator
 - Can approximate any surface with arbitrary accuracy
 - All polygons can be broken up into triangles
 - Guaranteed to be:
 - Planar
 - Triangles - Convex
 - Simple to render
 - Can implement in hardware



University of
British Columbia

Triangulation


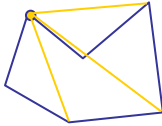
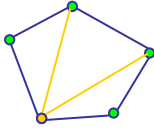
- Convex polygons easily triangulated
- Concave polygons present a challenge



University of British Columbia

OpenGL Triangulation

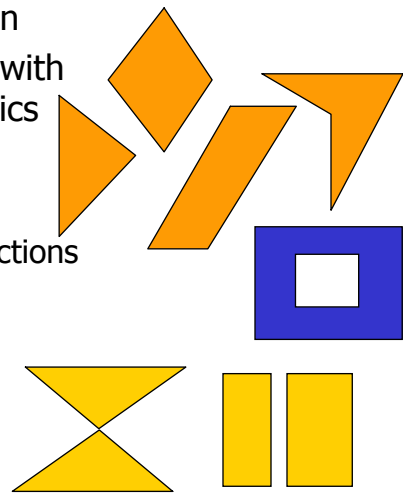

- Simple convex polygons
 - break into triangles, trivial
 - glBegin(GL_POLYGON) ... glEnd()
- Concave or non-simple polygons
 - break into triangles, more effort
 - gluNewTess(), gluTessCallback(), ...



University of British Columbia

Problem

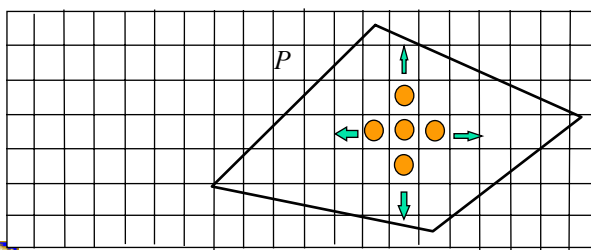

- Input: closed 2D polygon
- Problem: Fill its interior with specified color on graphics display
- Assumptions –
 - simple - no self intersections
 - simply connected
- Solutions
 - Flood fill
 - Scan conversion
 - Implicit test

University of British Columbia

Flood Fill Algorithm

- P polygon with n vertices v_0 to v_{n-1} ($v_n = v_0$)
- C color
- $P = (x,y) \in P$ point inside P


University of British Columbia

Flood Fill

- Draw edges
- Run:


```

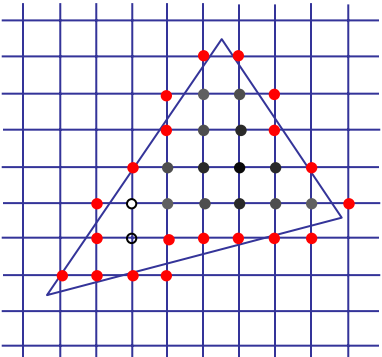

FloodFill (Polygon P, int x, int y, Color C)
if not (OnBoundary (x, y, P) or Colored (x, y, C))
begin
PlotPixel (x, y, C);
FloodFill (P, x + 1, y, C);
FloodFill (P, x, y + 1, C);
FloodFill (P, x, y - 1, C);
FloodFill (P, x - 1, y, C);
end ;
      
```
- Drawbacks?



University of
British Columbia

Flood Fill

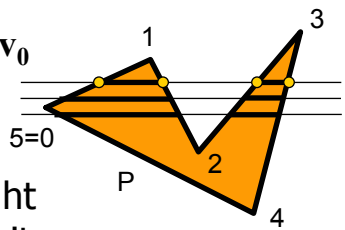
- Pixels visited up to 4 times to check if already set
- Need per-pixel flag indicating if set already
 - clear for every polygon!


University of
British Columbia

Scanline Algorithm

- P polygon with n vertices v_0 to v_{n-1} ($v_n = v_0$)
- C color
- 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)



The diagram shows an orange polygon labeled 'P' with vertices numbered 1, 2, 3, and 4. A horizontal scanline is drawn at the bottom, labeled '5=0'. Two yellow dots on the scanline indicate its intersections with the polygon's boundary.




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

The diagram illustrates a triangle drawn on a grid. A blue rectangle, representing the bounding box, encloses the triangle. The top-left corner of the bounding box is labeled (x_{min}, y_{min}) and the bottom-right corner is labeled (x_{max}, y_{max}) . The triangle's vertices are marked with pink dots.

University of British Columbia

Edge Walking

- Scanline is more efficient for specific polygons
 - trapezoids (triangles)
- Past graphics hardware
 - Exploit continuous L and R edges on trapezoid
 - Use Bresenham

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

The diagram shows a trapezoid with vertices at (x_L, y_B) , (x_R, y_B) , (x_L, y_T) , and (x_R, y_T) . Horizontal scanlines are drawn at y_B and y_T . The left edge is a line with a slope of 1, and the right edge is also a line with a slope of 1. The horizontal change for the left edge is Δx_L and for the right edge is Δx_R . The bottom edge is a horizontal line between x_L and 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;
}
    
```

Edge Walking Triangles

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


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

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




Edge Walking Triangles

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


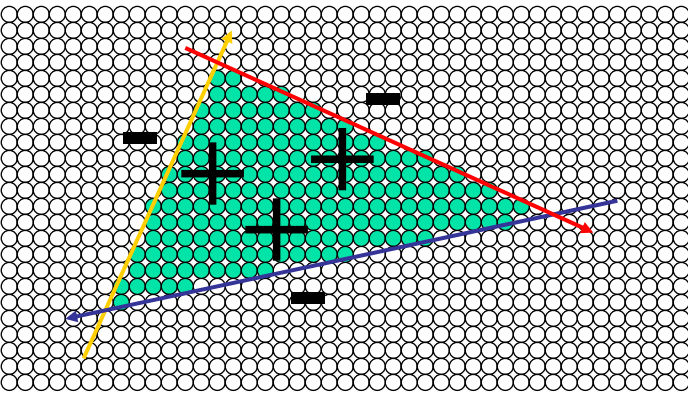


University of British Columbia




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


$$\begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = -C \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} A \\ B \end{bmatrix} = \frac{-C}{x_0 y_1 - x_1 y_0} \begin{bmatrix} y_1 - y_0 \\ x_1 - x_0 \end{bmatrix}$$

- choose $C = x_0 y_1 - x_1 y_0$ for convenience
- Then $A = y_0 - y_1$ and $B = x_0 - x_1$




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


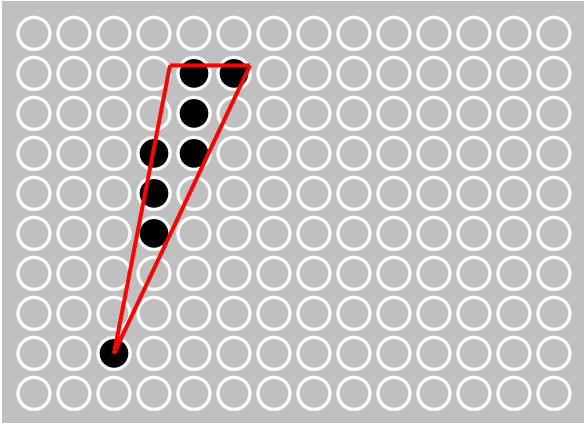
- *Exactly which pixels should be lit?*
 - Pixels inside triangle edges
- *What about pixels exactly on the edge?*
 - Draw - BUT order of triangles matters (it shouldn't)
 - Don't draw - BUT gaps possible between triangles
- Need consistent (if arbitrary) rule
 - Example: draw pixels on left or top edge,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



University of
British Columbia

Triangle Rasterization Issues

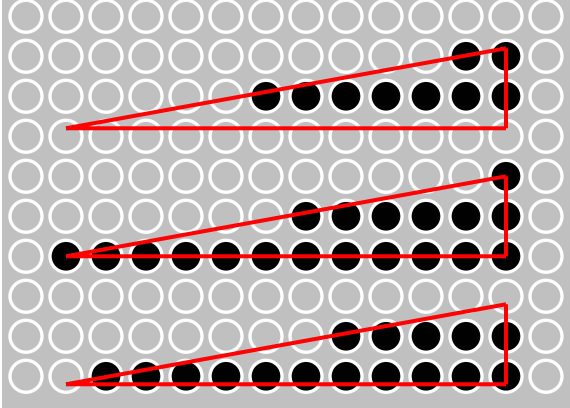
- Sliver




University of
British Columbia

Triangle Rasterization Issues

- Moving Slivers



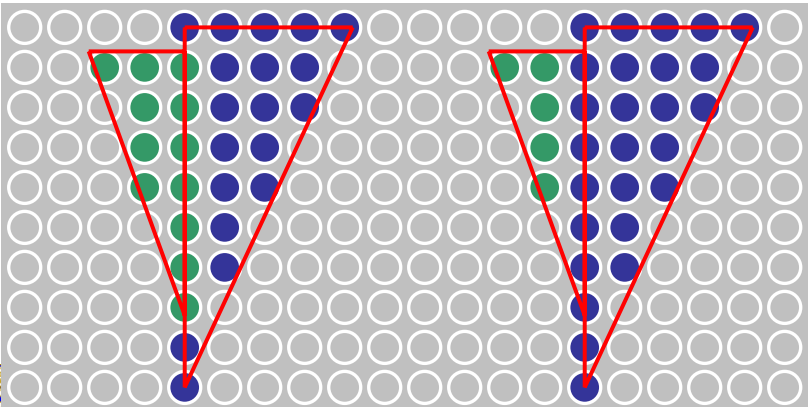
The diagram shows a grid of white circles on a gray background. Three red triangles are overlaid on the grid. The top triangle is slanted upwards to the right. The middle triangle is slanted downwards to the right. The bottom triangle is slanted upwards to the right. The edges of these triangles are jagged, following the grid lines, which illustrates the 'Moving Slivers' issue in scan conversion.




University of
British Columbia

Triangle Rasterization Issues


- Shared Edge Ordering



The diagram shows a grid of white circles on a gray background. Two red triangles are overlaid on the grid, sharing a vertical edge. The pixels on this shared edge are colored blue, while the pixels inside the triangles are colored green. This illustrates the 'Shared Edge Ordering' issue in scan conversion.




University of
British Columbia




Interpolation

- 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

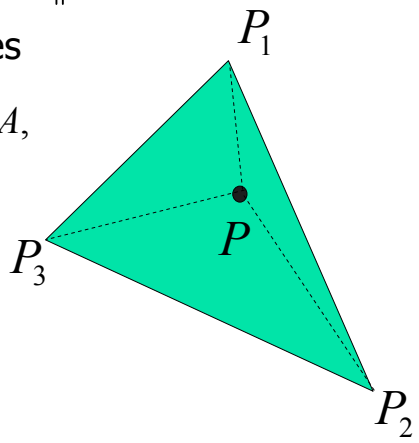

- Area

$$A = \frac{1}{2} \left\| \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3} \right\|$$
- Barycentric coordinates


$$a_1 = A_{P_2P_3P} / A, a_2 = A_{P_3P_1P} / A,$$

$$a_3 = A_{P_1P_2P} / A,$$

$$P = a_1P_1 + a_2P_2 + a_3P_3$$

University of
British Columbia



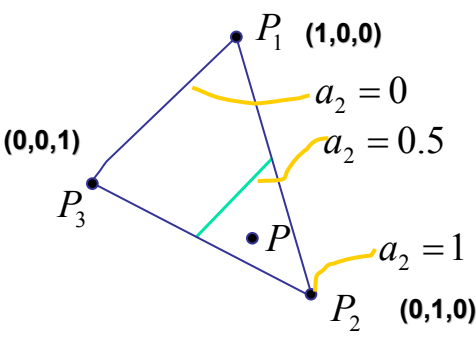
Barycentric Coordinates


- weighted combination of vertices

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


$$a_1 + a_2 + a_3 = 1$$

$$0 \leq a_1, a_2, a_3 \leq 1$$



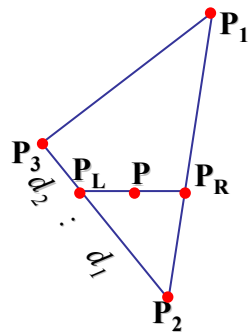


University of
British Columbia



Barycentric Coords: Alternative formula


- For point P on scanline:



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

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

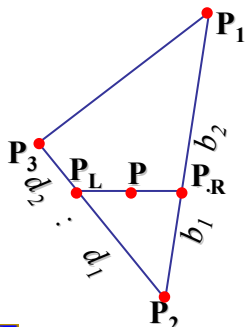
$$= \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3$$



University of
British Columbia


Computing Barycentric Coords

■ similarly:



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

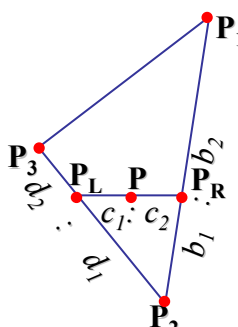
$$= (1 - \frac{b_1}{b_1 + b_2}) P_2 + \frac{b_1}{b_1 + b_2} P_1 =$$

$$= \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$


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



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

The diagram illustrates the bilinear interpolation process on a triangle. The triangle has vertices P_1 (top), P_2 (bottom right), and P_3 (left). A horizontal line at height y intersects the left edge at point P_L and the right edge at point P_R . A point $P(x,y)$ is marked on this line between P_L and P_R . The University of British Columbia logo is in the bottom left corner.