

Computer Graphics 7 - Rasterisation

Tom Thorne

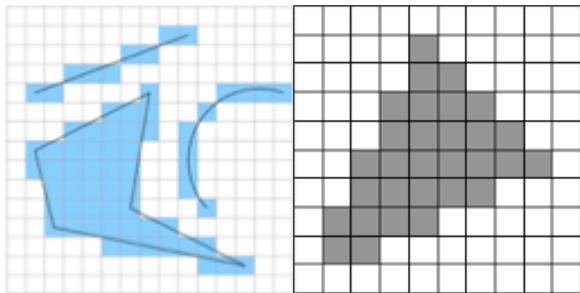
Slides courtesy of Taku Komura
www.inf.ed.ac.uk/teaching/courses/cg

Overview

- ▶ **Line rasterisation**
- ▶ Polygon rasterisation
- ▶ Mean value coordinates
- ▶ Decomposing polygons

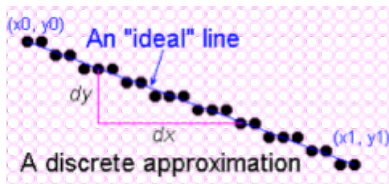
Rasterisation

- ▶ After projection, polygons are still described in continuous screen coordinates
- ▶ We need to use these polygons to colour in pixels on the screen



Rasterising lines

We need to convert a line described in a continuous coordinate system into a set of discrete pixels



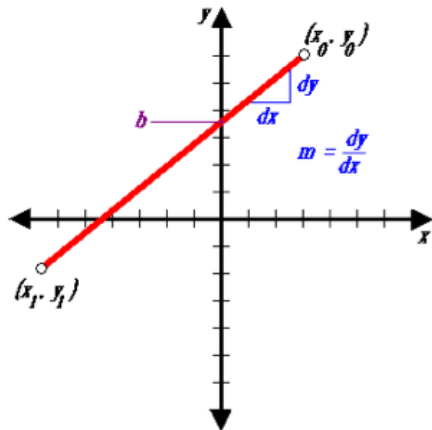
Simple line drawing

Linear algebra:

$$y = mx + b$$

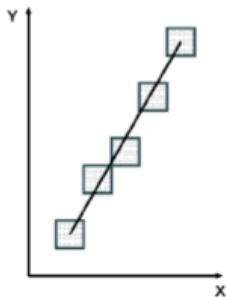
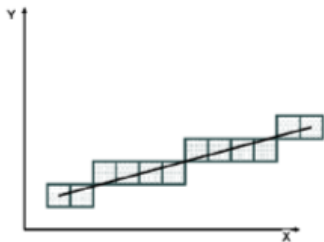
A very simple approach:

- ▶ Increment x , calculate new y
- ▶ Cast x and y to integers



Simple line drawing

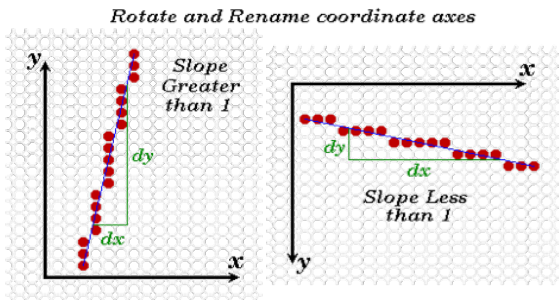
- ▶ For lines where $m \leq 1$ this seems to work well
- ▶ When $m > 1$ this doesn't work, the line becomes discontinuous



Symmetry

If $m \leq 1$ increment along the x axis, otherwise when $m > 1$, increment along y axis

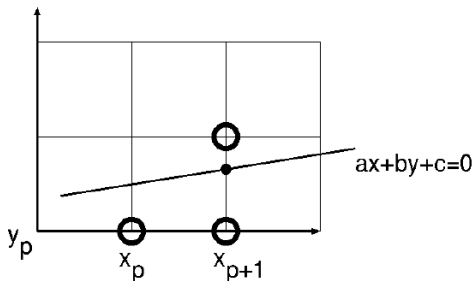
- ▶ This still requires a lot of floating point arithmetic



Midpoint algorithm

Iterate over steps, having lit a pixel (x_p, y_p) at step p :

- ▶ Check where the line intersects x_{p+1}
- ▶ Colour in (x_{p+1}, y_p) or (x_{p+1}, y_{p+1}) depending on which is closer to the intersection



Testing for the side of a line

- ▶ Assume the line is between (x_l, y_l) and (x_r, y_r)
- ▶ The slope of the line will be $\frac{dy}{dx}$ where $dx = x_r - x_l$ and $dy = y_r - y_l$

If $y = mx + c$ then $y = \frac{dy}{dx}x + c$ and so:

$$F(x, y) = ax + by + c = 0$$

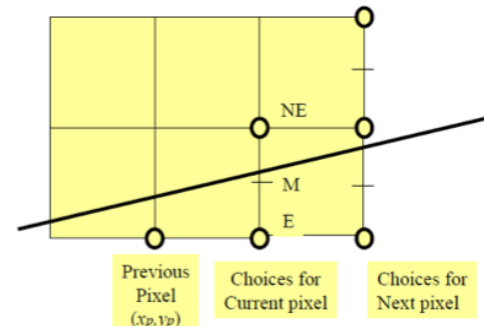
$$F(x, y) = dy \cdot x - dx \cdot y + c = 0$$

Decision variable

Assuming $\frac{dy}{dx} < 1$ using symmetry, evaluate F at point M :

$$d = F(x_p + 1, y_p + \frac{1}{2}) = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

Then d is a *decision variable*, if $d \leq 0$ then E is chosen as the next pixel, otherwise NE is chosen.



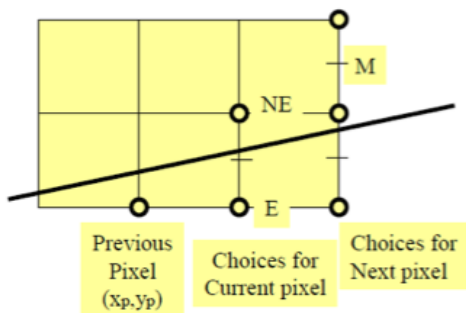
Updating the decision variable

If we chose NE:

$$d' = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

Then since $d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$,

$$d' = d + a + b = d + dy - dx$$



Initial value of d

The line starts from (x_l, y_l) , so:

$$\begin{aligned}d_{start} &= F(x_l + 1, y_l + \frac{1}{2}) \\&= a(x_l + 1) + b(y_l + \frac{1}{2}) + c \\&= ax_l + by_l + c + a + \frac{b}{2} \\&= F(x_l, y_l) + a + \frac{b}{2}\end{aligned}$$

But since (x_l, y_l) is on the line, $F(x_l, y_l) = 0$, so:

$$d_{start} = dy - \frac{dx}{2}$$

Then to avoid floating point operations, we multiply d by 2.

Decision variables

After we multiply by 2, $d = 2(ax + by + c)$.

$$d_{start} = 2dy - dx$$

$$d'_E = d + 2dy$$

$$d'_{NE} = d + 2dy - 2dx$$

Then we only need integer operations

Summary of the mid-point algorithm

- ▶ Start at first endpoint
- ▶ Calculate initial value for d
- ▶ Decide between two next pixels based on decision variable
- ▶ Update the decision based upon which pixel is chosen
- ▶ Iterate

Midpoint algorithm

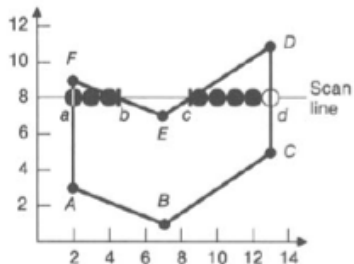
```
void MidpointLine(int
x1,y1,x2,y2)
{
int dx=x2-x1;
int dy=y2-y1;
int d=2*dy-dx;
int increE=2*dy;
int incrNE=2*(dy-dx);
x=x1;
y=y1;
WritePixel(x,y);
while (x < x2) {
    if (d<= 0) {
        d+=incrE;
        x++;
    } else {
        d+=incrNE;
        x++;
        y++;
    }
    WritePixel(x,y);
}
```


Overview

- ▶ Line rasterisation
- ▶ **Polygon rasterisation**
- ▶ Mean value coordinates
- ▶ Decomposing polygons

Scanline algorithm

- Fill pixels within a polygon scanline by scanline

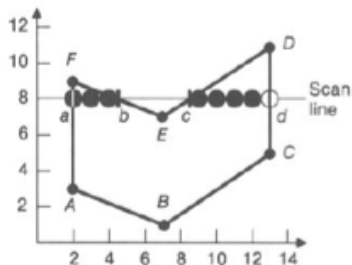


Scanline algorithm

On every scanline:

- ▶ Find intersections of scan line with all edges of the polygon
- ▶ Sort intersections in increasing order of x coordinate
- ▶ Fill in pixels between all pairs of intersections

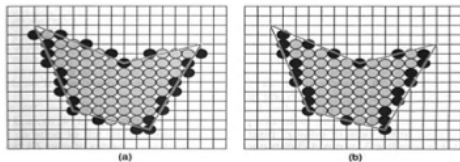
Works with concave polygons



Span extrema

Only turn on pixels that have their centre interior to the polygon

- ▶ Otherwise pixels overlap with adjacent polygons



This is done by rounding up values on left edges and down on right edges

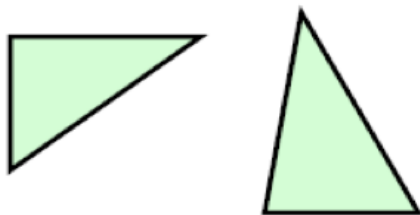
Scanline algorithm

Pros:

- ▶ Simple

Cons:

- ▶ Hard to parallelise efficiently
- ▶ Special cases can occur and require exception handling



- sliver: not even a single pixel wide



Barycentric coordinates for triangles

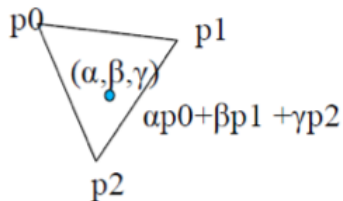
- ▶ Allow us to check whether a pixel is inside or outside a triangle
- ▶ Makes it easy to interpolate attributes between vertices
- ▶ Used in GPUs
- ▶ Easy to parallelise

Barycentric coordinates for triangles

Given a 2D triangle with vertices p_0 , p_1 , p_2 . For any point in the plane p :

$$\begin{aligned} p &= p_0 + \beta(p_1 - p_0) + \gamma(p_2 - p_0) \\ &= (1 - \beta - \gamma)p_0 + \beta p_1 + \gamma p_2 \\ &= \alpha p_0 + \beta p_1 + \gamma p_2 \end{aligned}$$

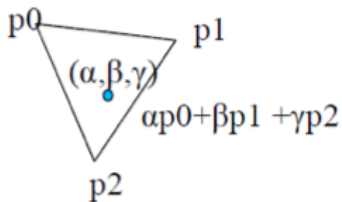
$$\alpha + \beta + \gamma = 1$$



Barycentric coordinates for triangles

The values $\alpha, \beta, \gamma \in [0, 1]$ if and only if p is inside the triangle.

α, β, γ are the *barycentric coordinates* of the point p .



Calculating barycentric coordinates

If the triangle is composed of $p_0 = (x_0, y_0)$, $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, then for a point (x, y) :

$$\alpha = \frac{f_{12}(x,y)}{f_{12}(x_0,y_0)}, \quad \beta = \frac{f_{20}(x,y)}{f_{20}(x_1,y_1)}, \quad \gamma = \frac{f_{01}(x,y)}{f_{01}(x_2,y_2)}$$

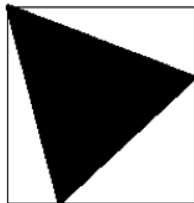
where $f_{ab} = (y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a$

Bounding box of a triangle

We calculate a bounding box around the triangle, by taking the minimum and maximum vertex coordinates in each direction:

$$x_{min}, y_{min} = \min(x_0, x_1, x_2), \min(y_0, y_1, y_2)$$

$$x_{max}, y_{max} = \max(x_0, x_1, x_2), \max(y_0, y_1, y_2)$$



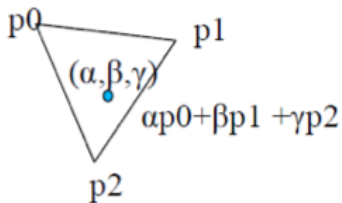
Scanning inside the triangle

- ▶ For each pixel in the bounding box, compute the barycentric coordinates
- ▶ Shade the pixel if all three values $\alpha, \beta, \gamma \in [0, 1]$



Interpolation

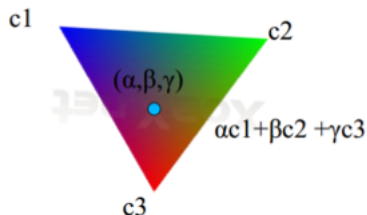
Barycentric coordinates can be used to interpolate attributes of triangle vertices, for example colour, depth, normal vectors or texture coordinates.



Interpolation of colour

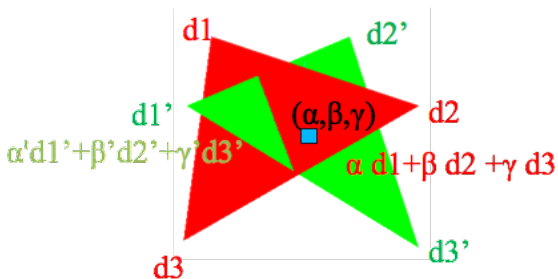
Gouraud shading:

- ▶ Calculate colour at vertices and interpolate the colour over the surface

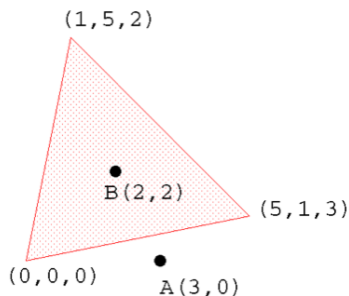


Interpolation of depth

- ▶ When triangles overlap each other, depth needs to be calculated at each pixel in case they intersect
- ▶ Calculate using barycentric coordinates
- ▶ Used in Z-buffering



Exercise

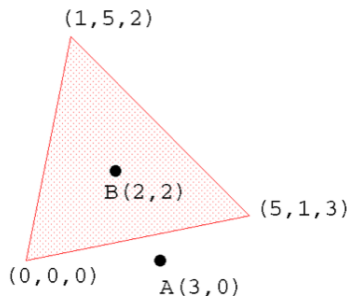


- ▶ What are the barycentric coordinates of A and B?
- ▶ What is the surface depth (Z coordinate) at B

$$\gamma = \frac{(y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0}{(y_0 - y_1)x_2 + (x_1 - x_0)y_2 + x_0y_1 - x_1y_0}$$

$$\beta = \frac{(y_0 - y_2)x + (x_2 - x_0)y + x_0y_2 - x_2y_0}{(y_0 - y_2)x_2 + (x_2 - x_0)y_2 + x_0y_2 - x_2y_0}$$

Exercise



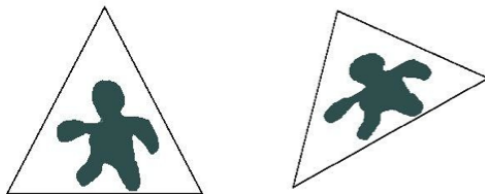
Barycentric coordinates

- ▶ $A = \left(\frac{1}{2}, \frac{5}{8}, -\frac{1}{8}\right)$
- ▶ $B = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$

Depth at $B = \frac{5}{3}$

Shape editing

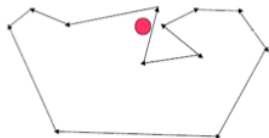
We can apply the same barycentric coordinates within a triangle when its shape is edited



General polygons

Barycentric coordinates for polygons with more vertices:

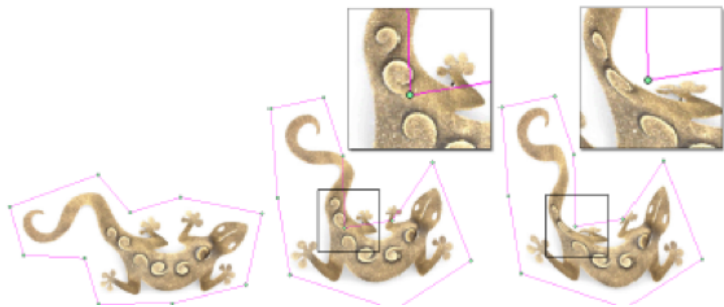
$$v = \frac{\sum_i w_i p_i}{\sum_i w_i}$$



Barycentric coordinates for 3D meshes:

- ▶ Mean value coordinates
- ▶ Harmonic coordinates (generalised barycentric coordinates)

Shape editing

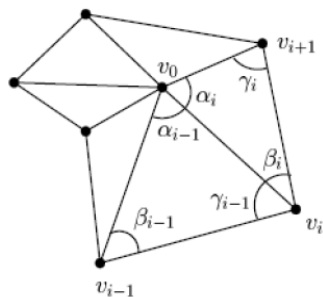


Mean value coordinates

Coordinates that can:

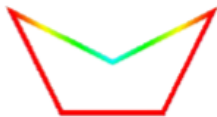
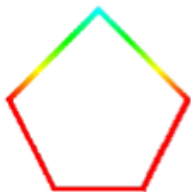
- ▶ smoothly interpolate boundary values
- ▶ works with concave polygons
- ▶ works in 3D

$$w_i = \frac{\tan \alpha_{i-1}/2 + \tan \alpha_i/2}{\|v_i - v_0\|}$$



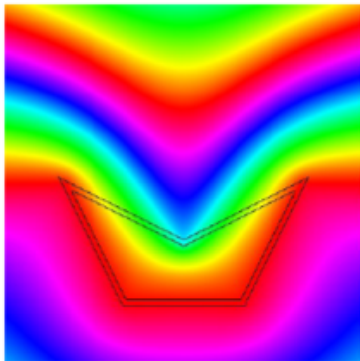
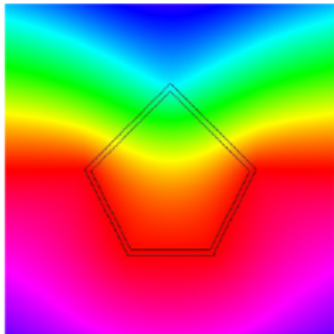
Mean value coordinates

- ▶ Can interpolate convex and concave polygons
- ▶ Smoothly interpolates the interior as well as the exterior



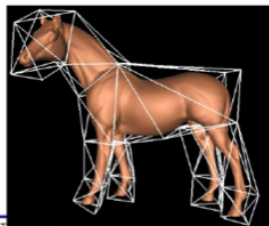
Mean value coordinates

- ▶ Can interpolate convex and concave polygons
- ▶ Smoothly interpolates the interior as well as the exterior



Mean value coordinates

- ▶ Can be computed in 3D
- ▶ Applicable for mesh editing

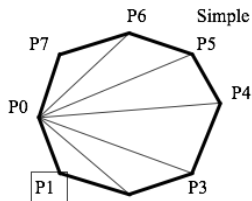


Overview

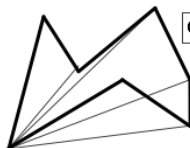
- ▶ Line rasterisation
- ▶ Polygon rasterisation
- ▶ Mean value coordinates
- ▶ **Decomposing polygons**

Polygon decomposition

For polygons with more than three vertices, we usually decompose them into triangles



Simple for convex polygons.



Concave more difficult.

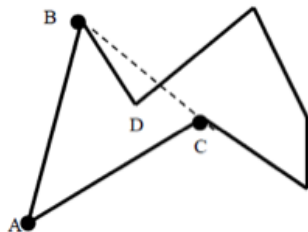
Polygon decomposition

Algorithm:

- ▶ Find leftmost vertex and label it A
- ▶ Compose potential triangle out of A and adjacent vertices B and C
- ▶ Check to see if another point of the polygon is inside the triangle ABC
- ▶ If all other points are outside ABC, remove ABC from the polygon and proceed with next leftmost vertex

Polygon decomposition

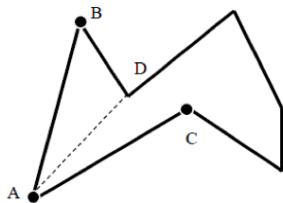
- ▶ Left most vertex is A
- ▶ Form a triangle between A and the adjacent B and C
- ▶ Check if all other vertices are outside this triangle



Vertex 'D' fails test.

Polygon decomposition

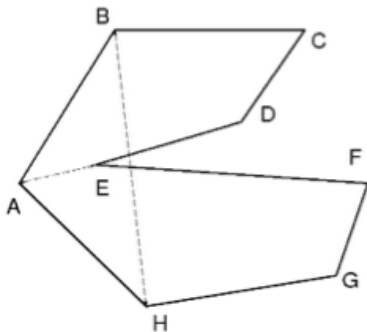
If a vertex is inside, split the polygon by the inside vertex and point A and continue:



Test ABD in same manner as before,

Polygon decomposition

The new edge may split the polygon in two. If so recurse over each polygon:



Split into ABCDE and AEF GH.

Summary

Rasterisation:

- ▶ Line rasterisation, midpoint algorithm
- ▶ Triangle rasterisation, scanline algorithm, barycentric coordinates
- ▶ Mean value coordinates
- ▶ Polygon decomposition into triangles

References

Midpoint and scanline algorithm:

- ▶ Foley Chapter 3.2, 3.5, 3.6

Barycentric coordinates:

- ▶ Shirley Chapter 2.7

Mean value coordinates:

- ▶ Floater, M. S. Mean value coordinates. *Computer Aided Geometric Design*, 20(1), 19–27, 2003
- ▶ Ju, T., Schaefer, S., & Warren, J. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics*, 24(3), 561–566, 2005.

Polygon decomposition:

- ▶ <http://www.siggraph.org/education/materials/HyperGraph/scanline/outprims/polygon1.htm>