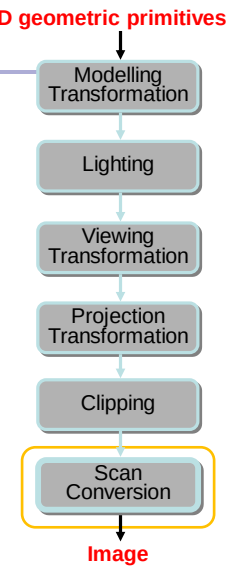
3D Graphics Programming Tools

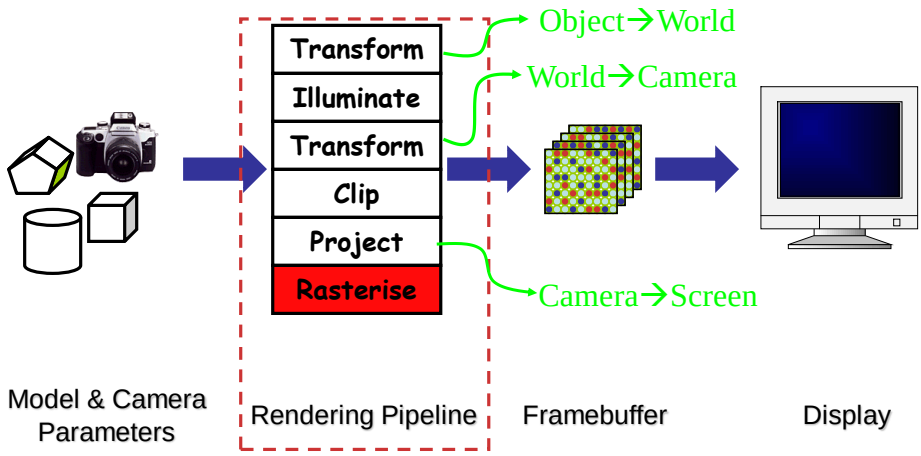
Rasterisation

Today's agenda

- Polygons
- Polygon rasterisation
- Triangulation
- Edge walking
- Edge equations
- Active edge table

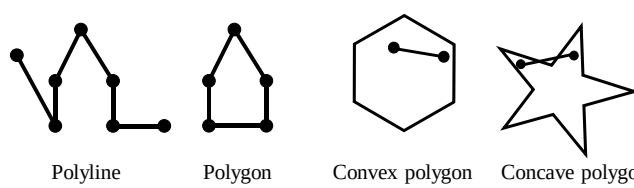


The rendering pipeline



Lines and polylines

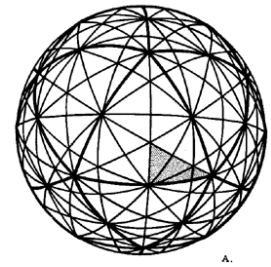
- Polylines
  - lines drawn between ordered points to create more complex forms
  - Same first and last point make closed polyline or *polygon*. If it does not intersect itself, called *simple polygon*.
  - Convex polygons → for every pair of points in the polygon, the segment between them is fully contained in the polygon
  - Concave polygons → Not convex: some two points in the polygon are joined by a segment not fully contained in the polygon

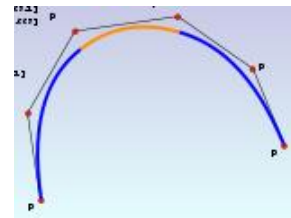


Polygons

- In interactive graphics → polygons rule the world!
- Two main reasons
  - Lowest common denominator for surfaces
    - Can represent any surface *with arbitrary accuracy*
  - Mathematical simplicity lends itself to simple, regular rendering algorithms
    - Such algorithms embed well in hardware

(Alternatives: Splines, mathematical functions, volumetric isosurfaces…)





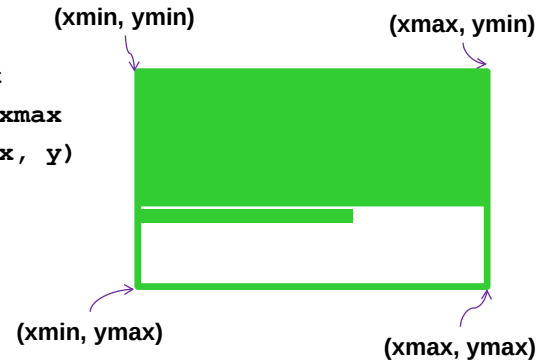
Filling shapes

- Filling shapes
  - Turn on all the pixels on a raster display that are inside a mathematical shape
- Questions before filling:
  - Is the shape closed with a boundary?
  - Which pixel is inside and which is outside?
  - What color/pattern should the shape be filled with?

Filling rectangles

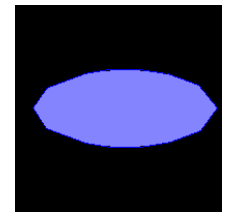
- If the rectangle is aligned with the x and y axis, then we can easily determine which pixels lie inside the rectangle.

```
for y = ymin to ymax
    for x = xmin to xmax
          SetPixel (x, y)
```



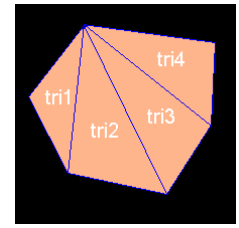
Rasterising polygons

- Triangle is the minimal unit of a polygon
  - All polygons can be broken up into triangles
  - Triangles are guaranteed to be:
    - Planar (flat)
    - convex

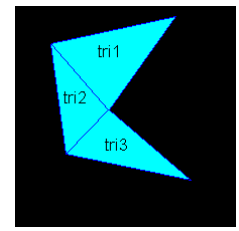


Triangulation

- Convex polygons easily triangulated



- Concave polygons present a challenge



Concave polygon triangulation

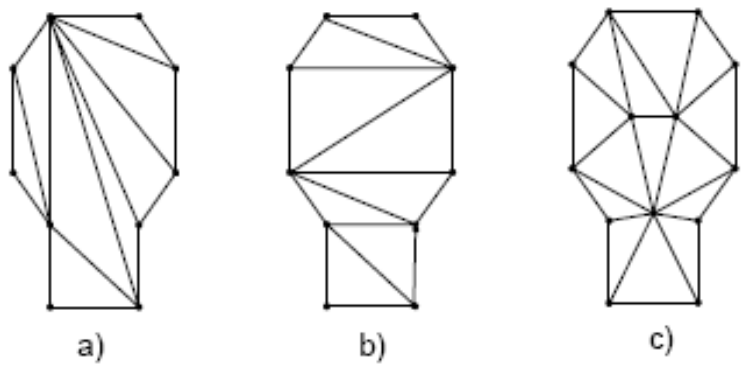
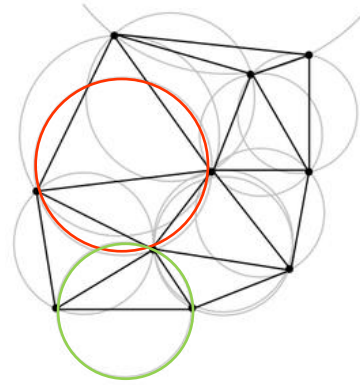
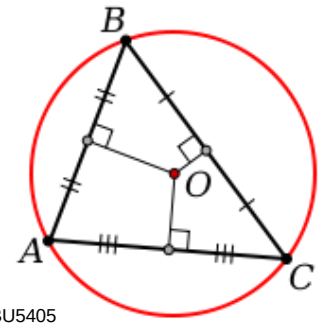


Figure 1: a) Low quality triangulation; b) High quality triangulation; c) Triangulation with Steiner's Points

Delaunay triangulation

A **Delaunay triangulation** for a set **P** of points in the plane is a triangulation DT(**P**) such that no point in **P** is inside the circumcircle of any triangle in DT(**P**).

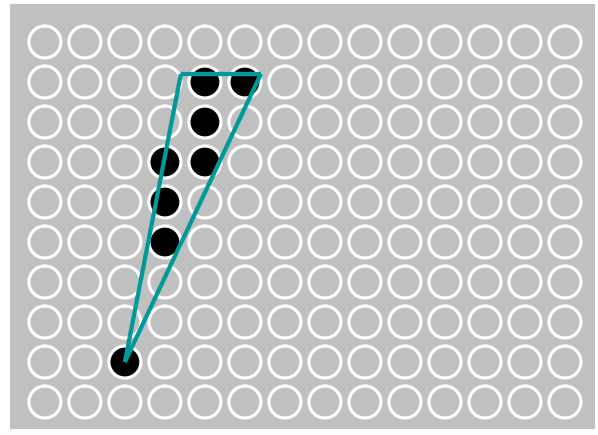




- Flip algorithms
- Incremental
- Divide and conquer
- Sweepline

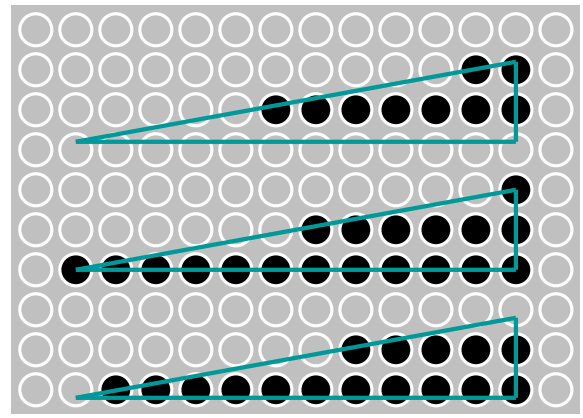
Triangle rasterisation issues

- Sliver



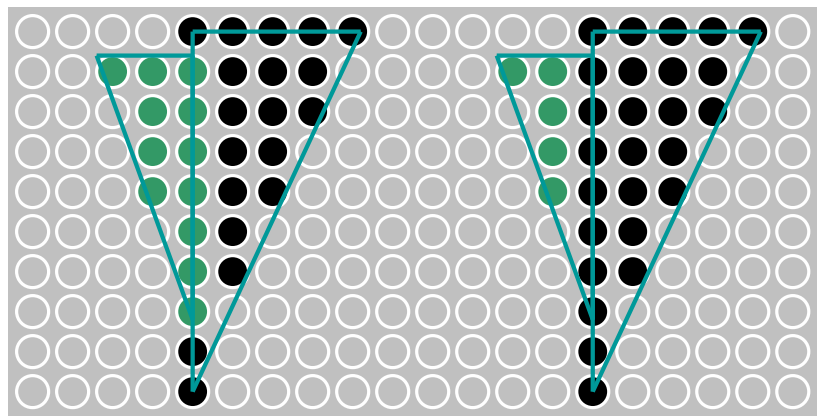
Triangle rasterisation issues

- Moving slivers



Triangle rasterisation issues

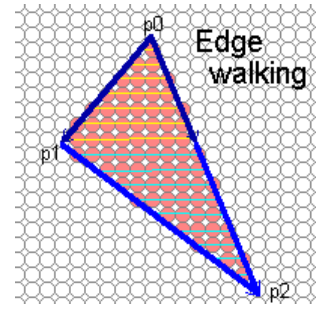
- Shared edge ordering



Rasterising triangles

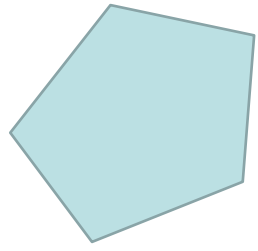
- Interactive graphics hardware
  - commonly uses *edge walking* or *edge equation* techniques for rasterising triangles

- Edge walking: basic idea
  - draw edges
    - interpolate colors down edges
  - fill in horizontal spans for each scanline
    - at each scanline, interpolate edge colors across span



Edge walking and convex shapes

Why do we want convex shapes for rasterisation?

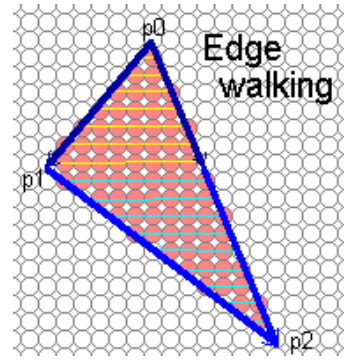


convex shape

non convex shape

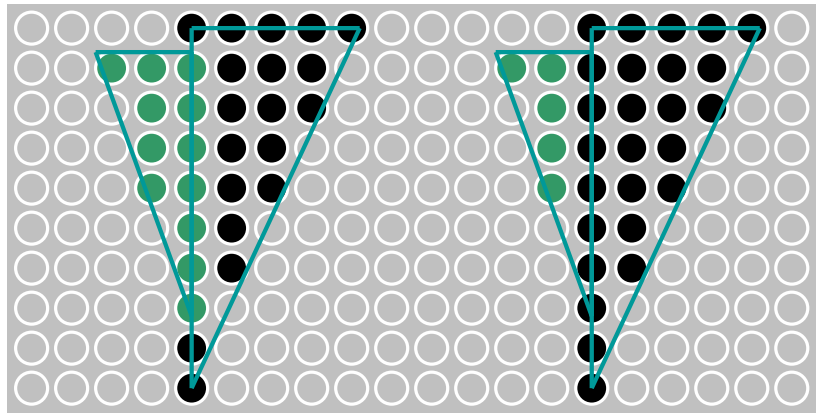
Edge walking

- Order the three triangle vertices in x and y
  - Find middle point in y dimension and compute if it is to the left or right of polygon.
- We know where left and right edges are.
  - Proceed from top scanline downwards
  - Fill each span
  - Until breakpoint (middle point) or bottom vertex is reached
- Advantage
  - can be made very fast (optimised)

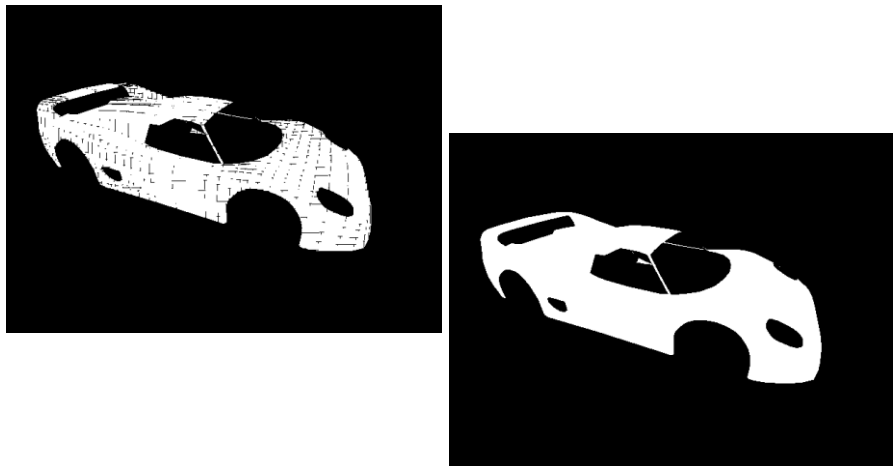


Triangle rasterisation issues

- Shared edge ordering



Triangle rasterisation issues



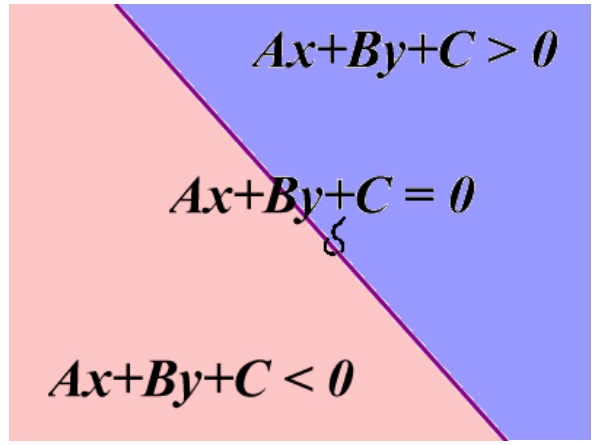
Edge equations

- Edge equation → the equation of the line defining that edge
  - Implicit equation of a line

    $Ax + By + C = 0$
  - Given a point (x,y), plugging x & y into this equation tells us whether the point is:
    - on the line: $Ax + By + C = 0$
    - "above" the line: $Ax + By + C > 0$
    - "below" the line: $Ax + By + C < 0$

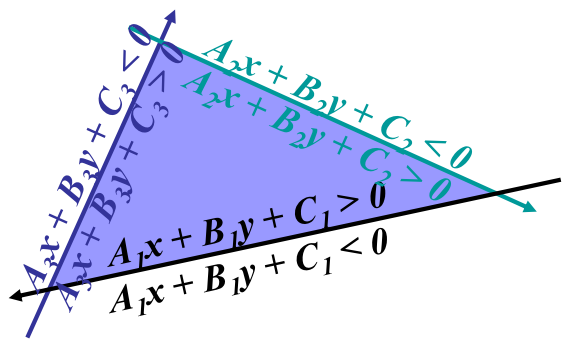
Edge equations

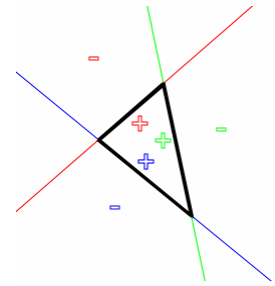
- Edge equations thus define two *half-spaces*:



Edge equations

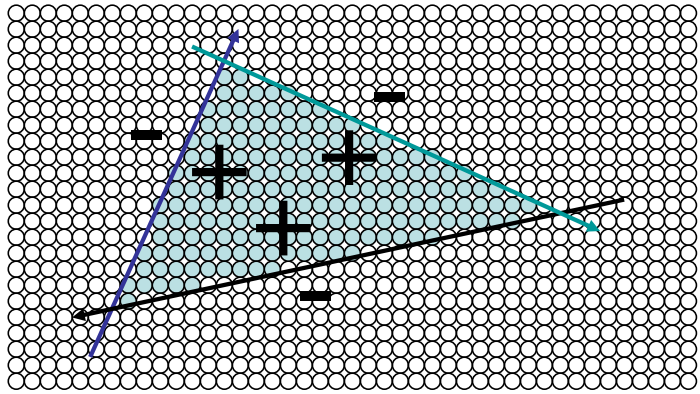
- And a triangle can be defined as the intersection of three positive half-spaces





Edge equations

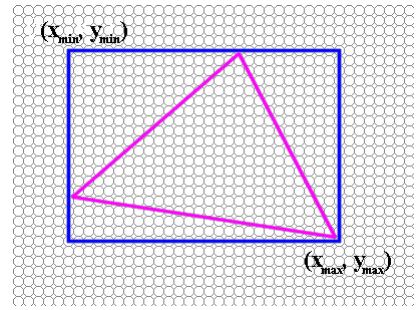
- So…simply turn on those pixels for which all edge equations evaluate to > 0



Using edge equations

How would you implement an edge-equation rasteriser ?

- *Which pixels do you consider?*
- *How do you compute the edge equations?*
- *How do you orient them correctly?*



Edge equations

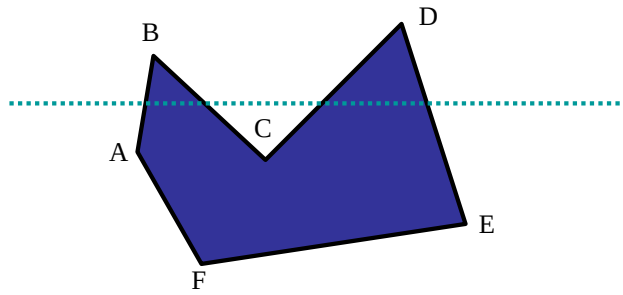
- We can find edge equation from two vertices
- Given three corners $\mathbf{P_0}$, $\mathbf{P_1}$, $\mathbf{P_2}$ of a triangle, what are our three edges?
- *To make sure that the half-spaces defined by the edge equations all share the same sign on the interior of the triangle*
  → Be consistent (Ex: $[\mathbf{P_0}\,\mathbf{P_1}]$, $[\mathbf{P_1}\,\mathbf{P_2}]$, $[\mathbf{P_2}\,\mathbf{P_0}]$)
- *To make sure that sign is positive?*
  → Test, and flip if needed ($A = -A$, $B = -B$, $C = -C$)

Triangle rasterisation issues (summary)

- Pixels inside the triangle edges
  – should be lit
- Pixels exactly on the edge
  – Draw them: order of triangles matters (it shouldn't)
  – Don't draw them: gaps possible between triangles
- We need a consistent (if arbitrary) rule
  – Example: draw pixels on left or top edge, but not on right or bottom edge

General polygon rasterisation

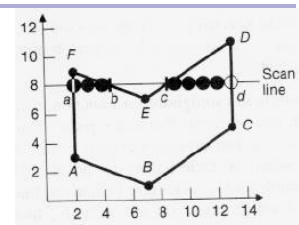
- Consider the following polygon:



- How do we know whether a given pixel on the scanline is inside or outside the polygon?

General polygon rasterisation

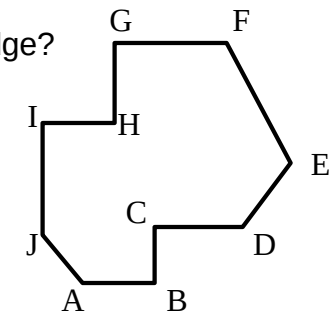
Basic idea: use a parity test



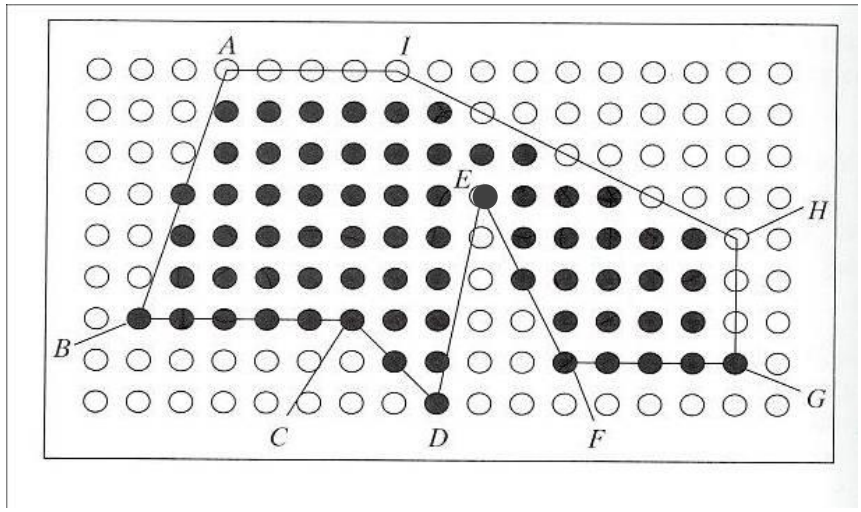

```
for each scanline
    edgeCnt = 0;
    for each pixel on scanline left to right
       if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
       // draw the pixel if edgeCnt odd
       if (edgeCnt % 2)
            setPixel(pixel);
```

General polygon rasterisation

- NB: count the vertices carefully
  - If exactly on pixel boundary?
  - Shared vertices?
  - Vertices defining horizontal edge?



Edge walking …



Faster polygon rasterisation

- Problem: how can we optimise the code?

```
for each scanline
    edgeCnt = 0;
    for each pixel on scanline left to right
        if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
        // draw the pixel if edgeCnt odd
        if (edgeCnt % 2)
            setPixel(pixel);
```

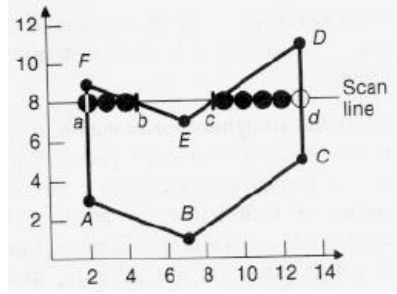
Note: big cost → testing pixels against each edge

- Solution: active edge table *(AET)*

Queen Mary University of London

Active edge table

- Idea
  - Edges intersecting a given scanline are likely to intersect the next scanline
  - The order of edge intersections does not change much from scanline to scanline



Queen Mary University of London

Active edge table

- The active-edge table is a data structure that consists of all the intersection points of the edges with the current scanline.

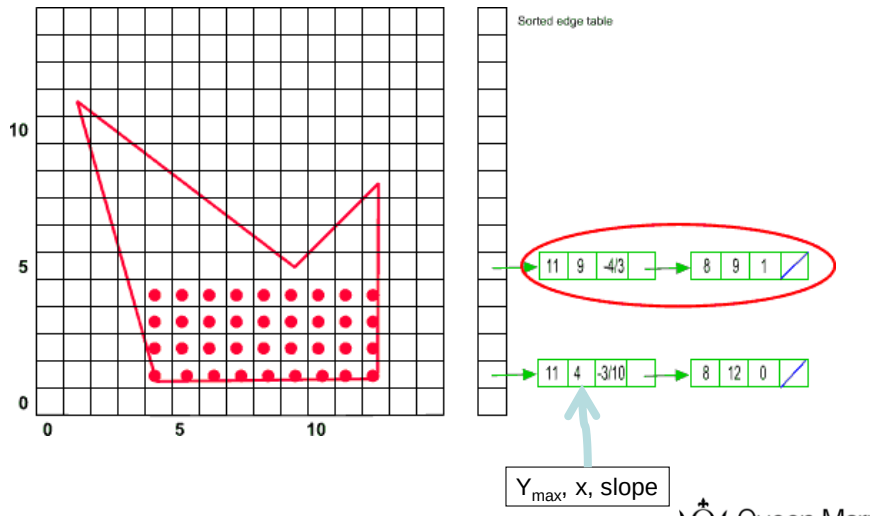
- These intersection points are sorted by increasing *x* coordinate. This allows the intersection points to be paired off, and be used for filling the scanline appropriately.

- As the scan conversion moves on to the next scanline, the AET is updated so that it properly represents that scanline.

Active edge table

- Algorithm: for scanline from bottom to top…
  - Sort all edges by their minimum y coordinate
  - Starting at bottom, add edges with $Y_{min} = 0$ to AET
  - For each scanline:
    - Sort edges in AET by x intersection
    - Walk from left to right, setting pixels by parity rule
    - Increment scanline
    - Retire edges with $Y_{max} < Y$
    - Add edges with $Y_{min} < Y$
    - Recalculate edge intersections
  - Stop when $Y > Y_{max}$ for last edges

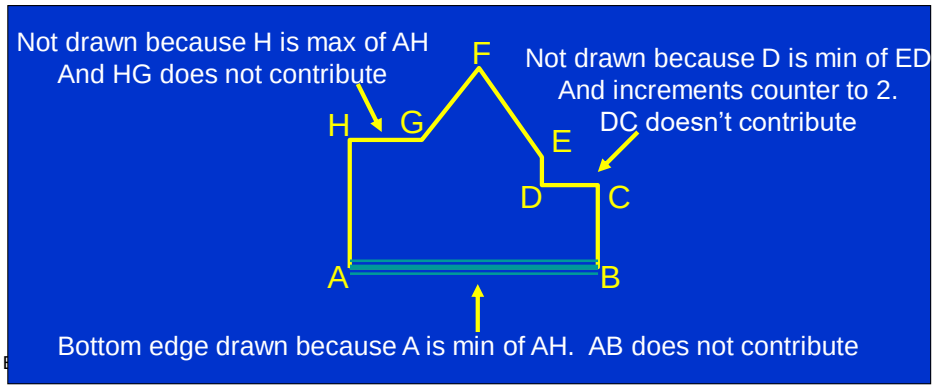
Active edge table



Queen Mary University of London

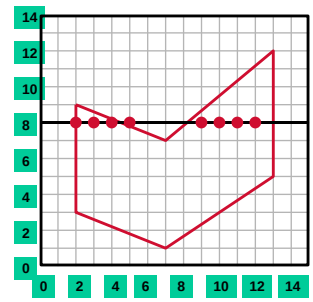
Review: polygon rasterisation

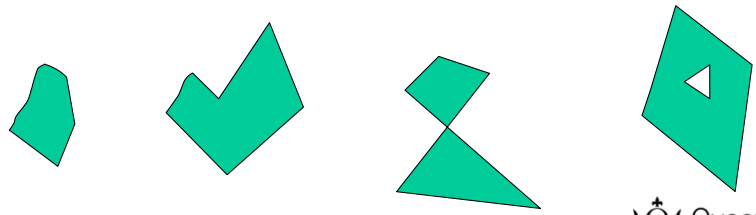
1. For scanline → determine all polygon edges that intersect current scanline
2. Sort edge intersections with scanline in least to greatest order
3. Use parity count to determine when pixels are drawn
4. Horizontal lines do not contribute to parity count
5. Y_{min} endpoints do contribute to parity count
6. Y_{max} endpoints do not contribute to parity count



Summary: filling polygons

- Use scan lines
- Edge coherence
- Finding intersections





What did we learn today?

- Polygons
- Polygon rasterisation
- Triangulation
- Edge walking
- Edge equations
- Active edge table