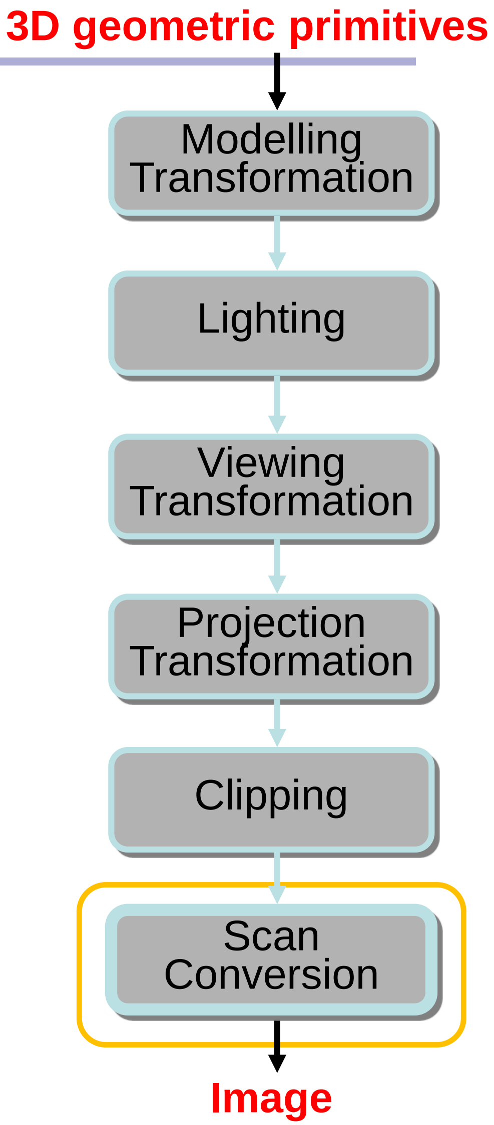
3D Graphics Programming Tools

Rasterisation

Queen Mary
University of London

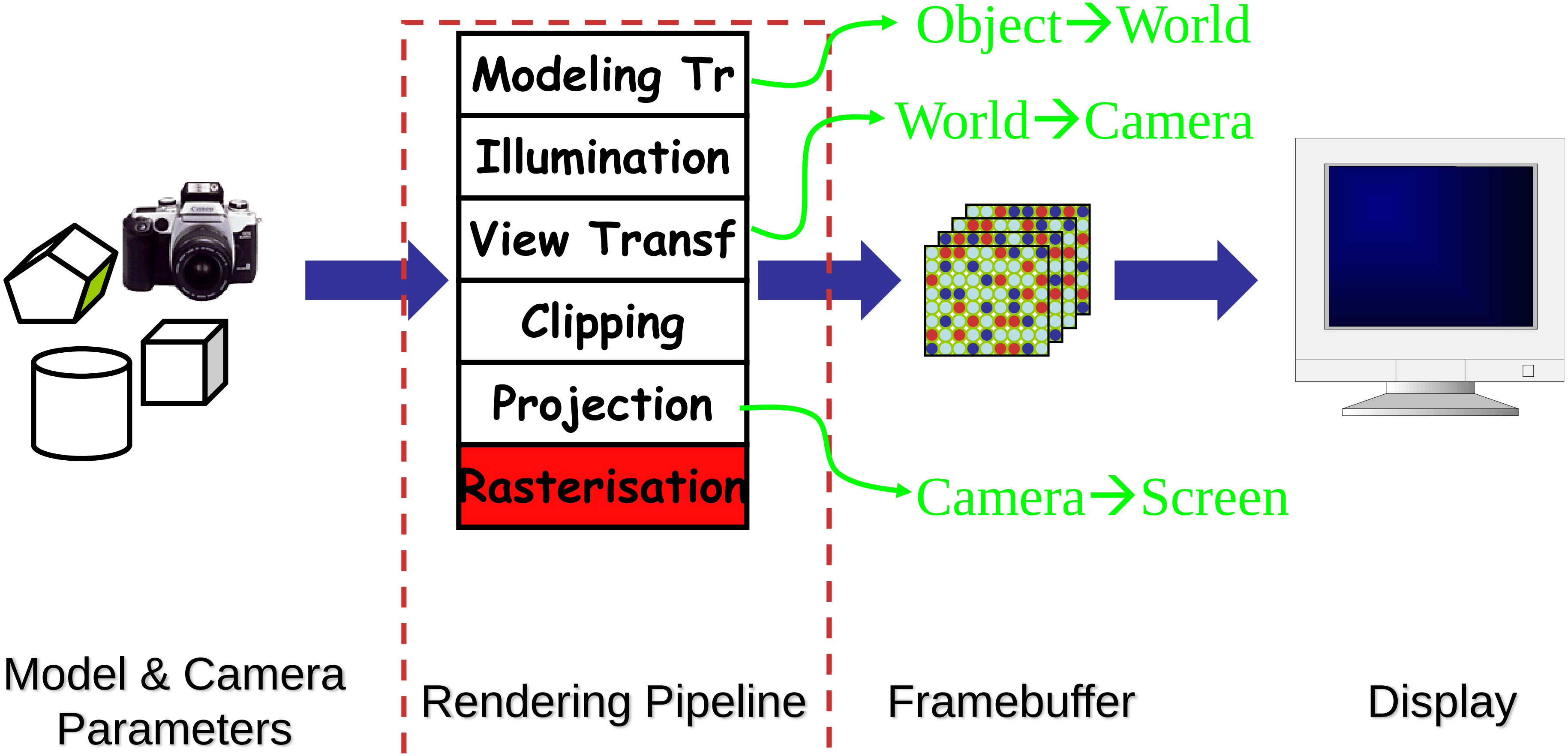
Today's agenda

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



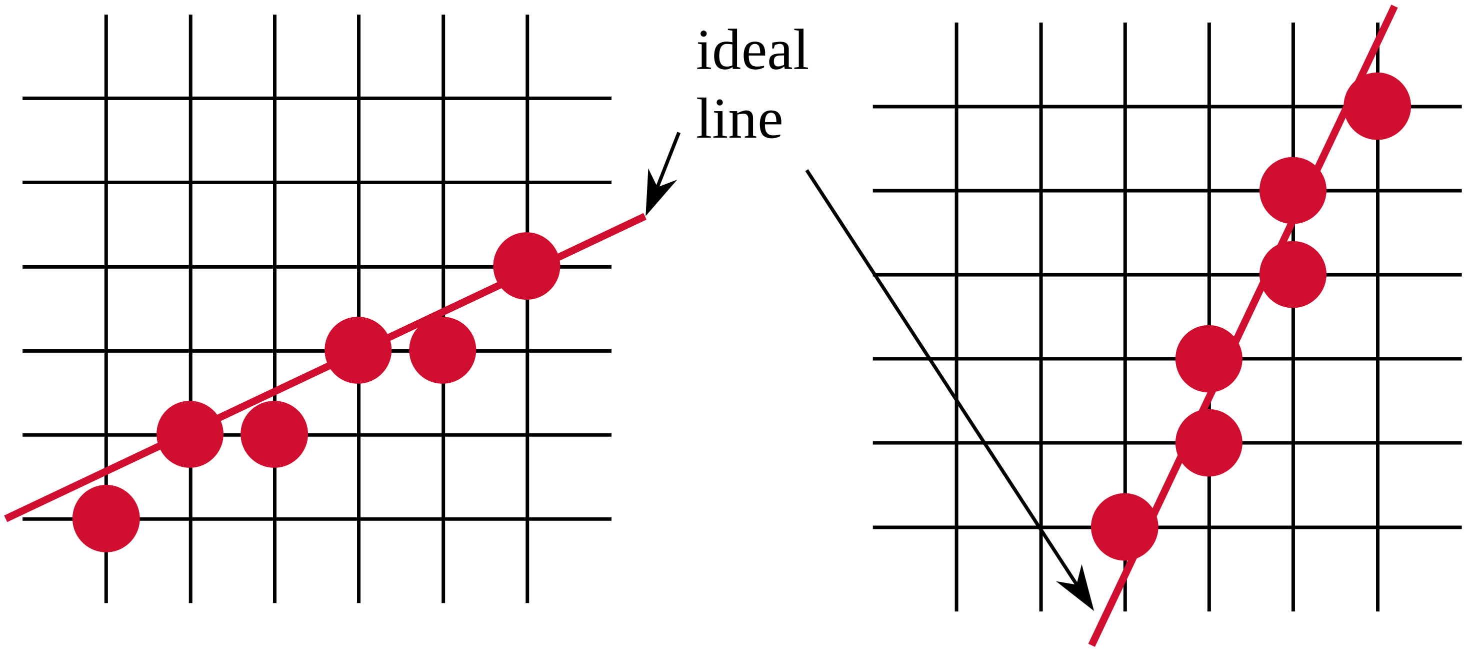
Queen Mary University of London

The rendering pipeline

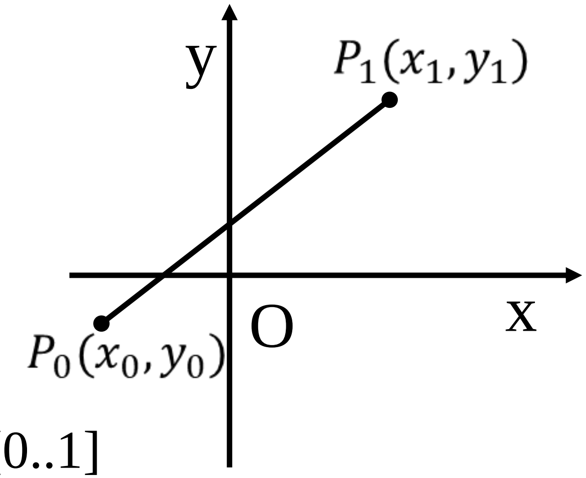


Scan Converting Lines

- Rasterization (scan-conversion) : turn 2D primitives into sets of pixels



Mathematics of Lines

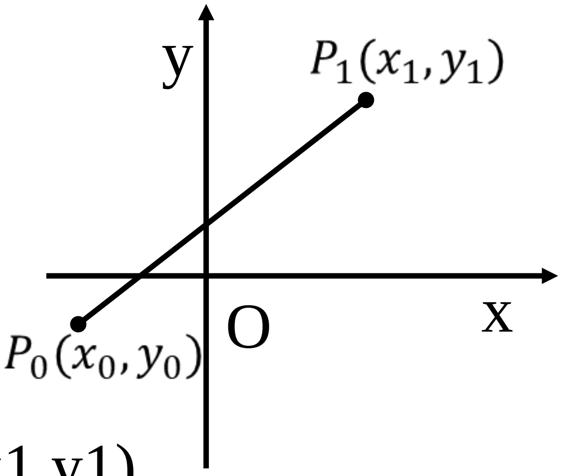


- Equation of a (2D) line: ax + by + c = 0
  Direction: (–b, a)
  Normal vector: (a, b)
- Parametric equation of a segment [P0-P1] :
  x(t) = x0 + t*(x1-x0) = (1-t)*x0 + t*x1
  y(t) = y0 + t*(y1-y0) = (1-t)*y0 + t*y1, where t in [0..1]
- For a line from (x0, y0) to (x1, y1), the line equation is
  y = mx + c where
  m is the slope of the line m = (y1 – y0) / (x1 – x0)
  c = y0 – m*x0
- More generally $\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}$

$$F(x, y) = (y - y_1)(x_2 - x_1) - (x - x_1)(y_2 - y_1) = 0$$
$$F(x, y) = (x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1) = 0$$

Queen Mary
University of London

Naive rasterization algorithm



1. start with the smallest of (x0,y0)
2. compute corresponding value of y
3. SetPixel(x, round(y))
4. increment x and loop until reaching max(x1,y1)

Cost: 1 float multiplication + 1 float addition + 1 round per loop

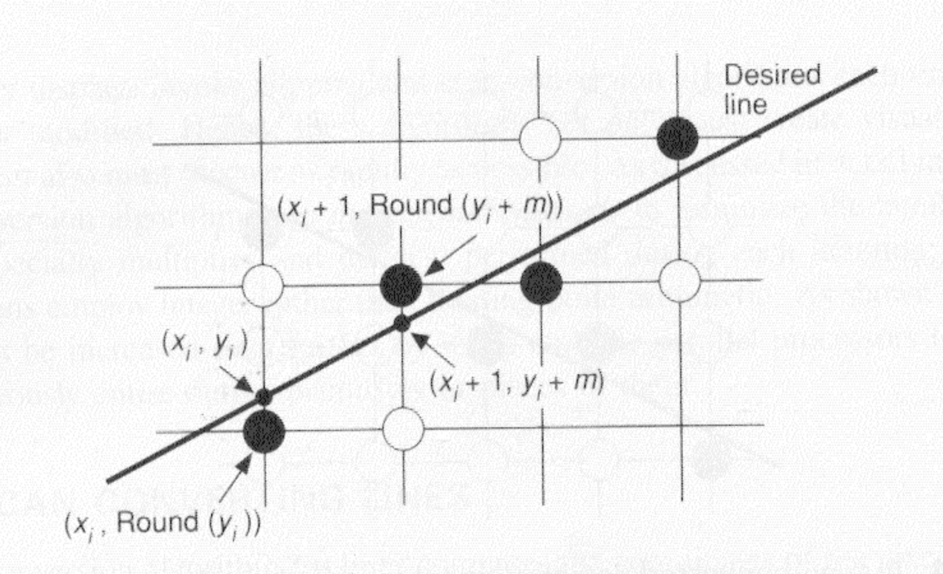
Faster Approaches

- Use integer calculations
- Avoid divides and multiplies
- Use incremental computations
- Use spatial coherence

Basic Incremental Algorithm

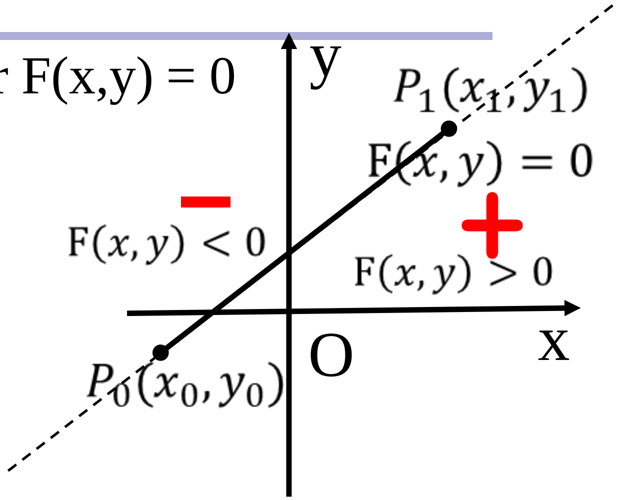
Digital differential analyzer (DDA) algorithm:

```
void Line(int x0, int y0, int x1, int y1)
{
   int x;
   float dy, dx, y, m;
   dy = y1 - y0; /* Floating */
   dx = x1 - x0; /* Floating */
   m = dy/dx; /* Floating division */
   y = y0;    /* Floating */
   for (x = x0; x<=x1,x++){
      SetPixel(x, round(y));
      y += m; /* Increment */
   }
}
```



Cost: 1 float add + 1 round per loop

Faster with Midpoints



Line equation: (x1 – x0)(y – y0) = (y1 – y0)(x – x0) or F(x,y) = 0

F(x,y) = (y1 – y0)(x – x0) – (x1 – x0)(y – y0)

= (y1–y0)x–(x1–x0)y – (y1–y0)x0+(x1–x0)y0

F(x+1, y) – F(x,y) = (y1–y0)

F(x+1, y+1) – F(x,y) = (y1–y0) – (x1–x0)

F(x+1, y+1/2) – F(x,y) = (y1–y0) – (x1–x0)/2

If point P(x,y) drawn, the next point is either P(x+1,y) or P(x+1,y+1)
To decide which point, use the relative position of the midpoint M = (x+1, y+1/2) with respect to the line, which half-plane it is, positive or negative.

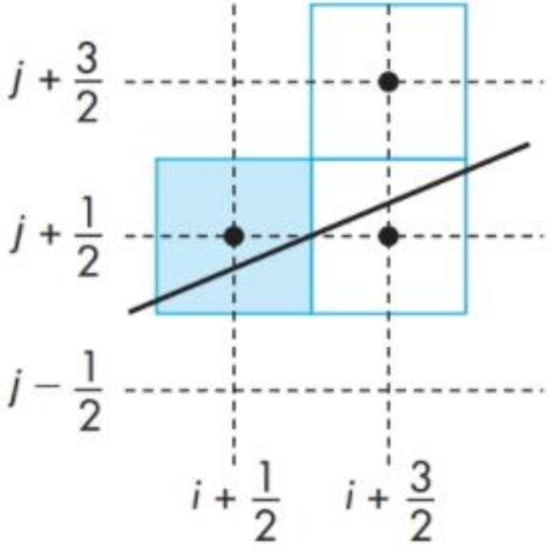
We use 2F(x,y) = 0, dx=x1–x0, dy=y1–y0, then we have:

d(x0, y0) = 2dy – dx

d(x+1, y+1) = 2F(x+1, y+1) – 2F(x,y) = 2dy – 2dx

d(x+1, y) = 2F(x+1, y) – 2F(x,y) = 2dy

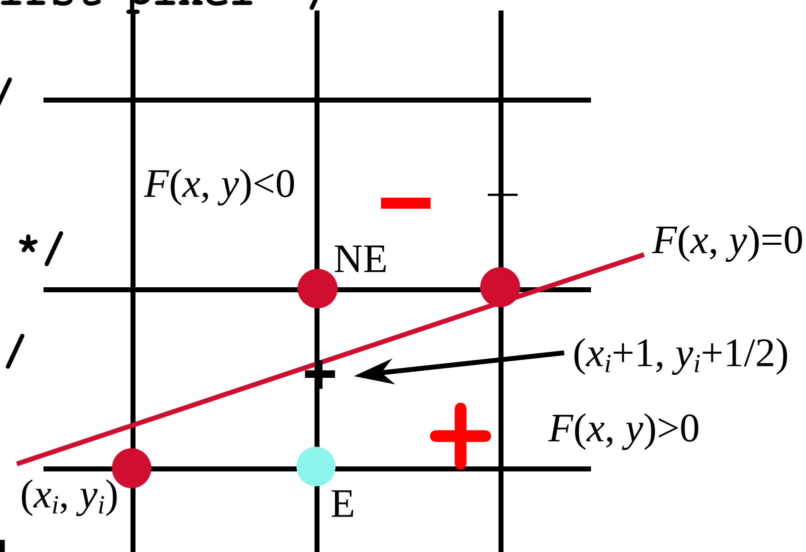
as the updating for every move.



Bresenham's Midpoint Line Algorithm

Bresenham's algorithm:

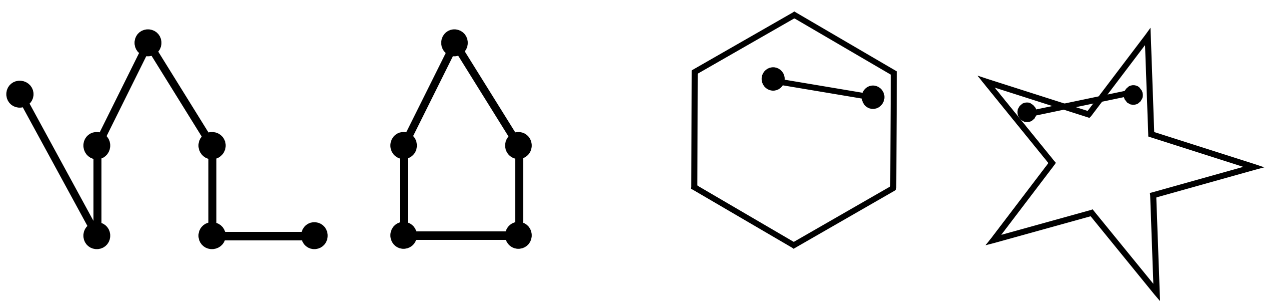
```
void MidpointLine(int x0, int y0, int x1, int y1)
{  int dx,dy,incrE,incrNE,d,x,y;
   dx=x1-x0;  dy=y1-y0;
   d=2*dy-dx;            /* initial value of d */
   incrE=2*dy;           /* increment for move to E */
   incrNE=2*(dy-dx);     /* increment for move to NE */
   x=x0;  y=y0;
   DrawPixel(x,y)          /* draw the first pixel */
   while (x<x1){
        if (d<=0){          /* choose E */
               d+=incrE;
               x++;        /* move E */
        } else {            /* choose NE */
               d+=incrNE;
               x++;  y++; /* move NE */
        }
        SetPixel (x,y);
   }
}
```



Cost: 1 integer add per pixel

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

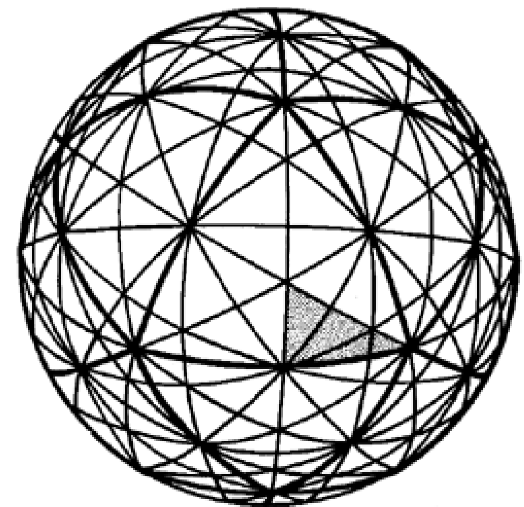


Polyline Polygon Convex polygon Concave polygon

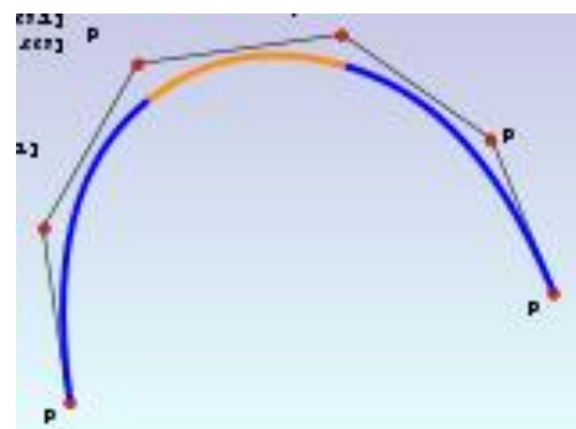
Queen Mary University of London

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…)



Queen Mary
University of London

Filling shapes

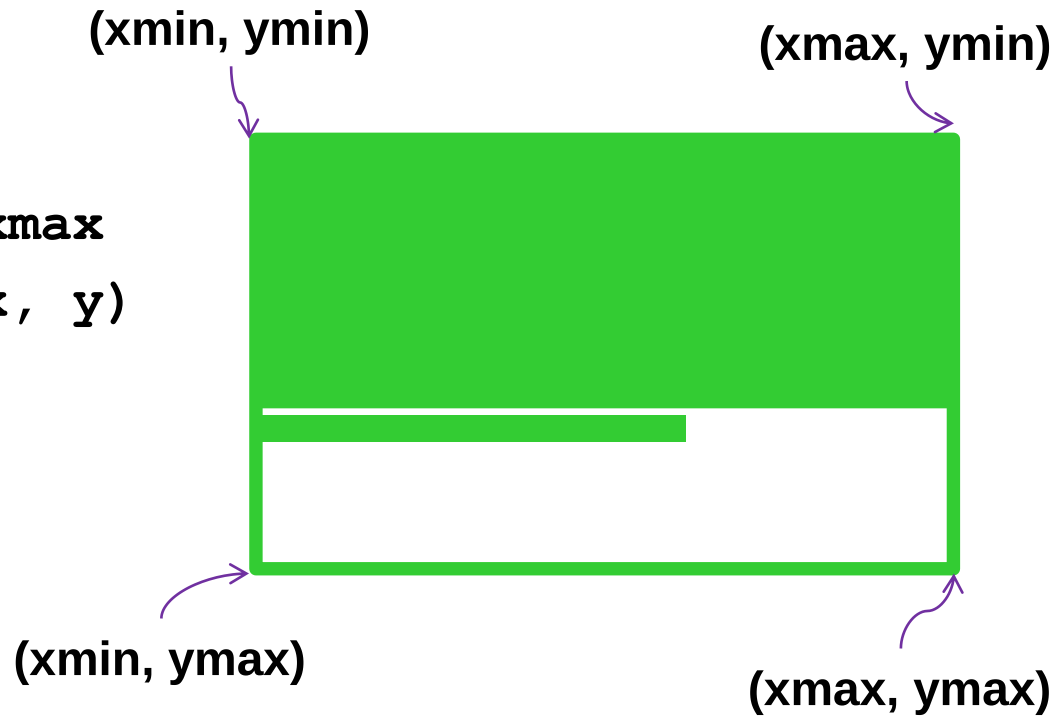
- Filling shapes
  - Turn on all the pixels on a raster display that are inside a mathematical shape

- 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

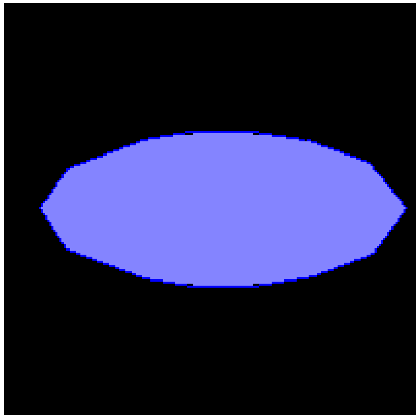
- <u>If the rectangle is aligned with the x and y axis</u>, then we can easily determine which pixels lie inside the rectangle.

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

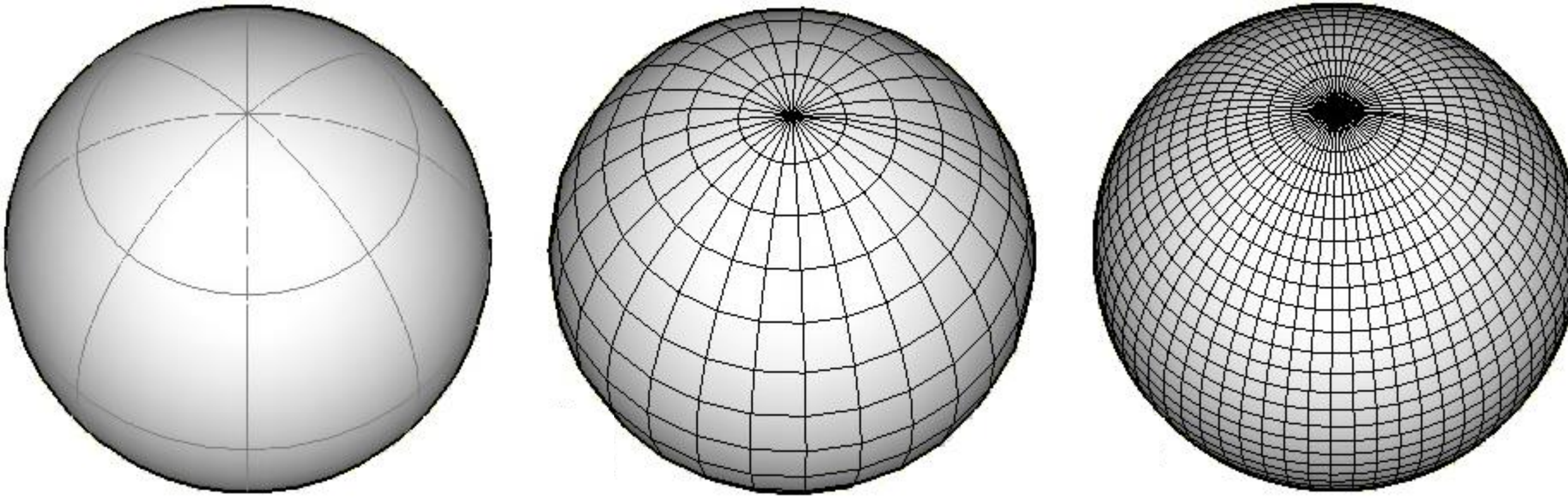


Rasterising polygons

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

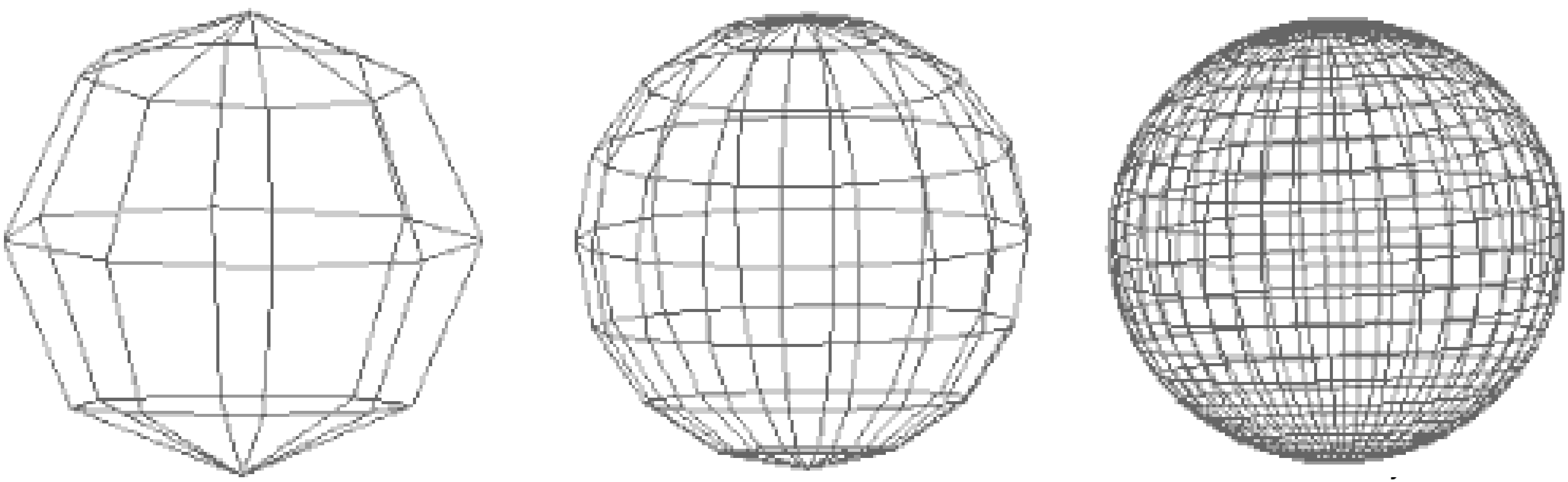


Queen Mary
University of London



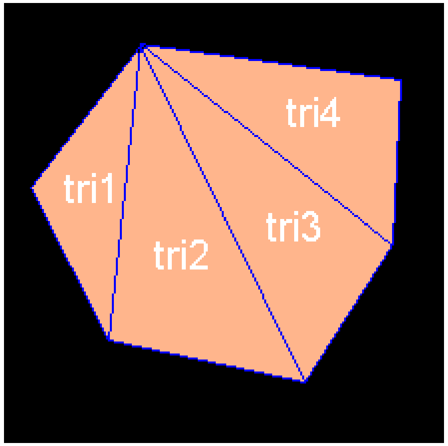
Surfaces of spheres with lines of longitude and latitude

Polygon mesh approximation

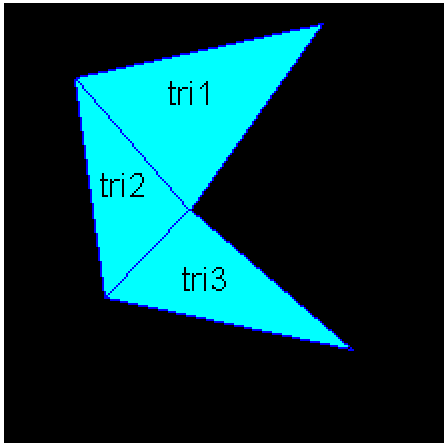


Triangulation

- Convex polygons easily triangulated



- Concave polygons present a challenge



Concave polygon triangulation

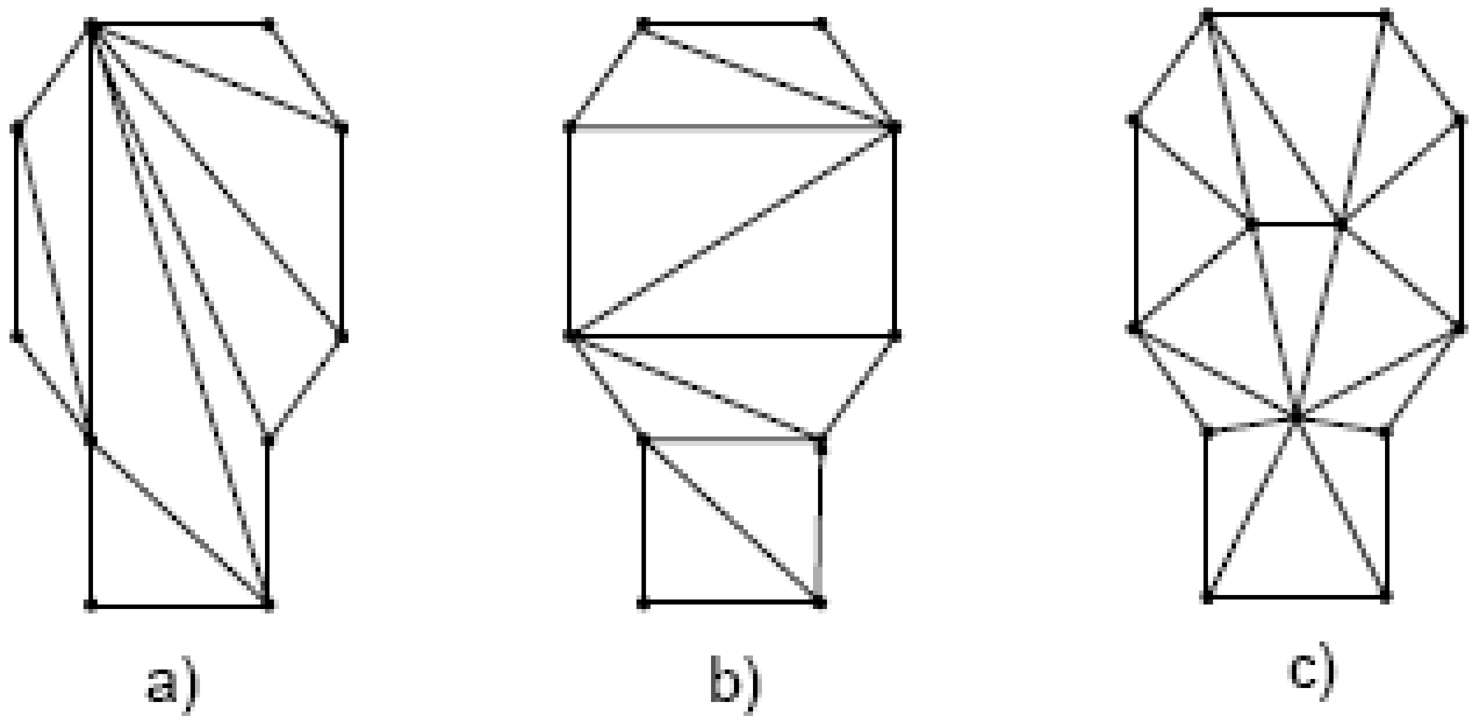
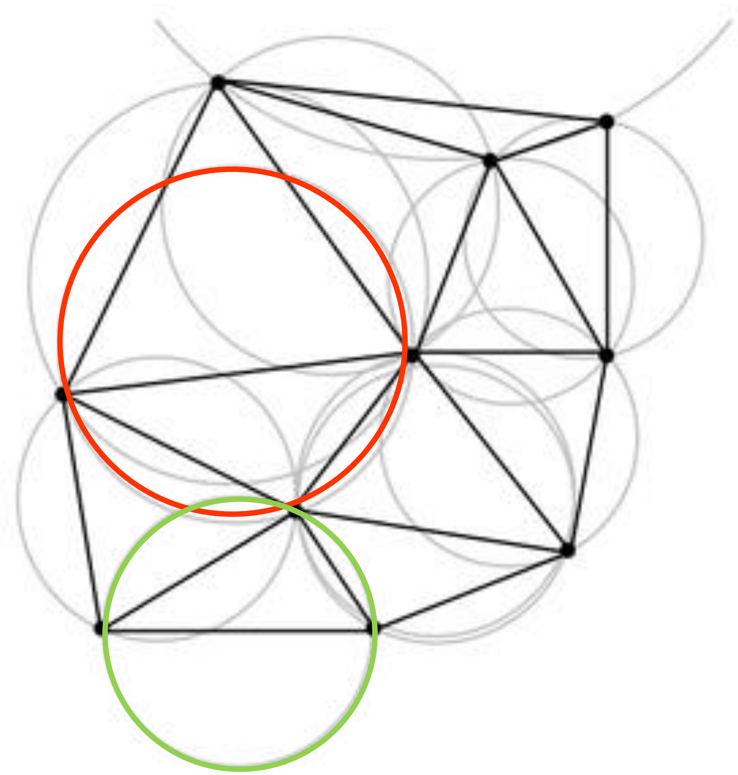
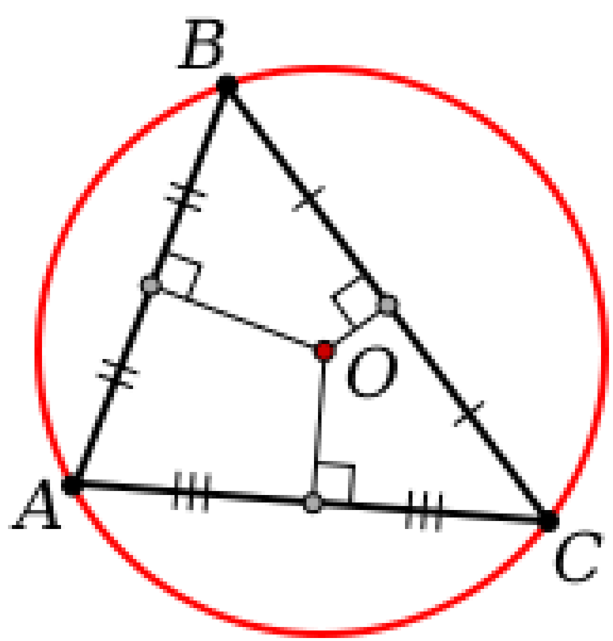


Figure 1: a) Low quality triangulation; b) High quality triangulation; c) Triangulation with Steiner's Points

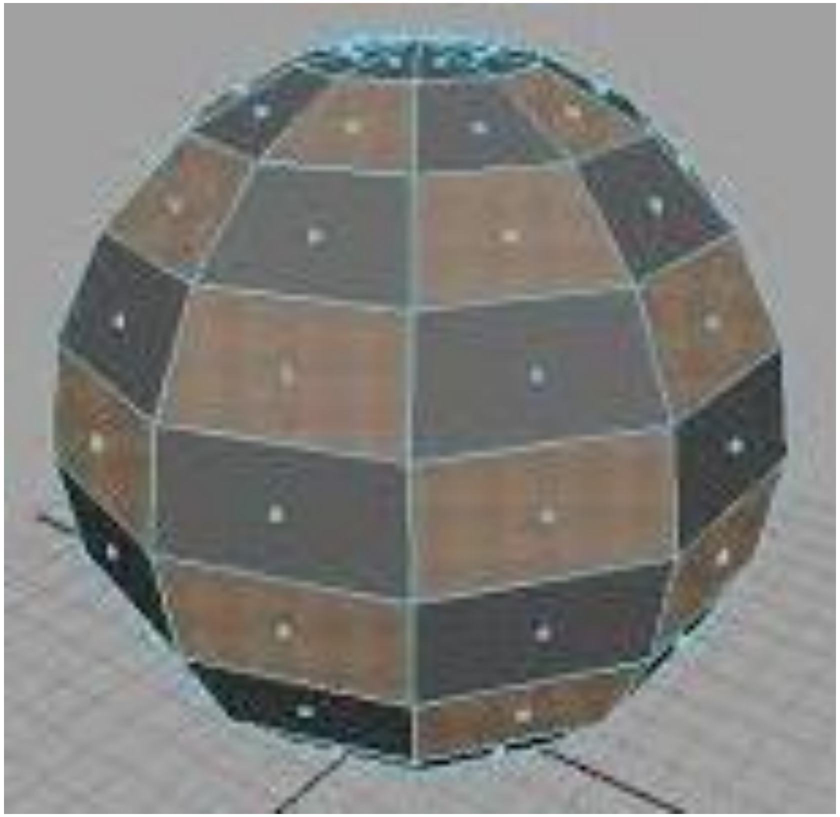
Delaunay triangulation

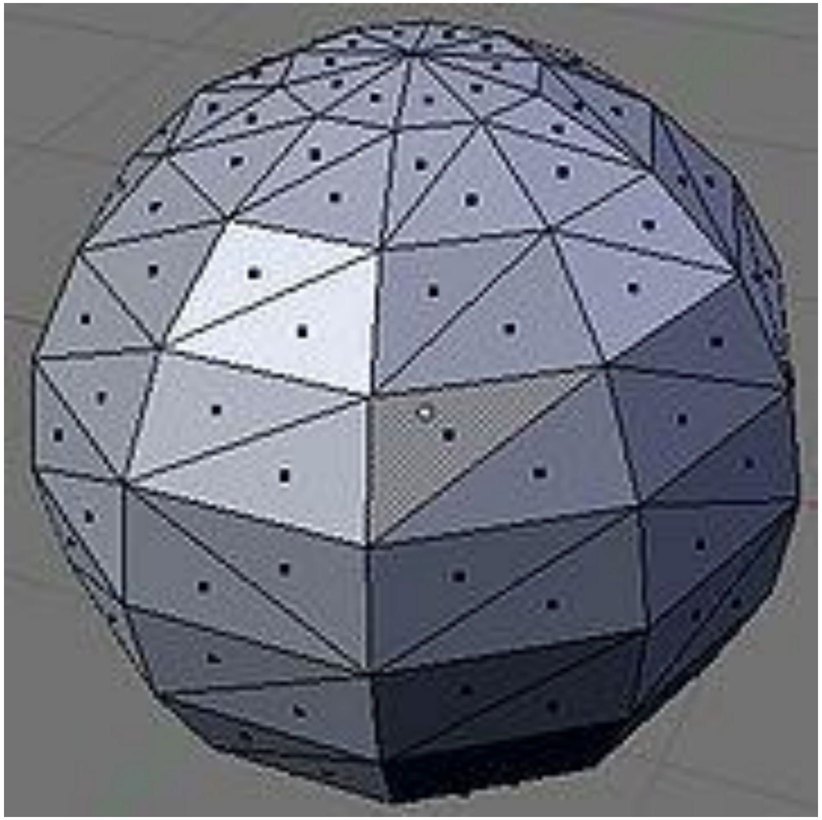
A **Delaunay triangulation** for a set **P** of vertices in the plane is a triangulation DT(**P**) such that no vertex in **P** is inside the circumcircle of any triangle in DT(**P**).



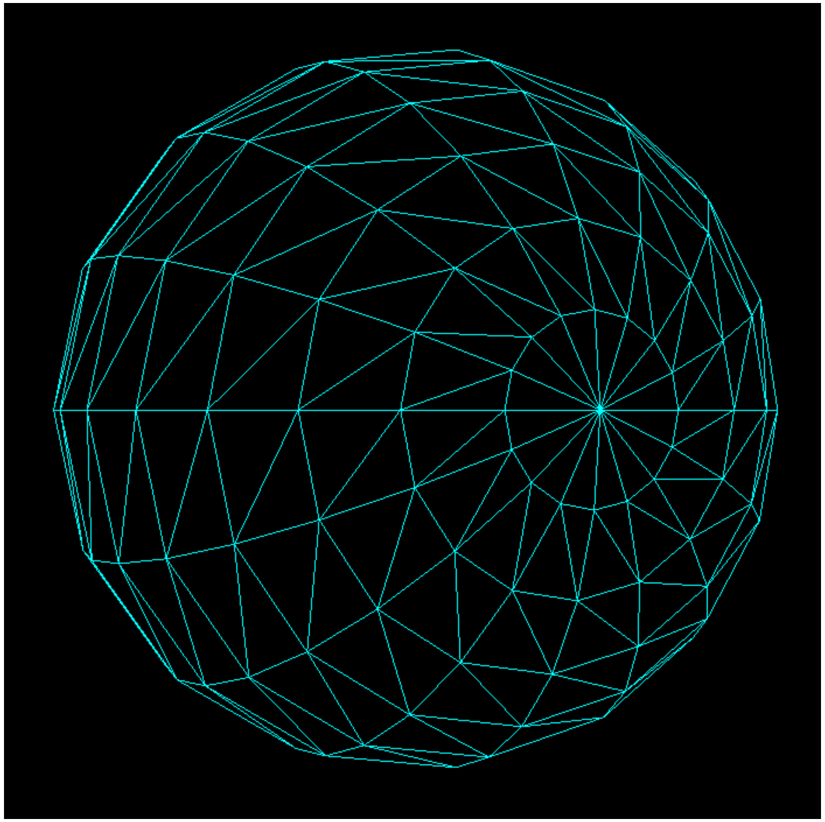


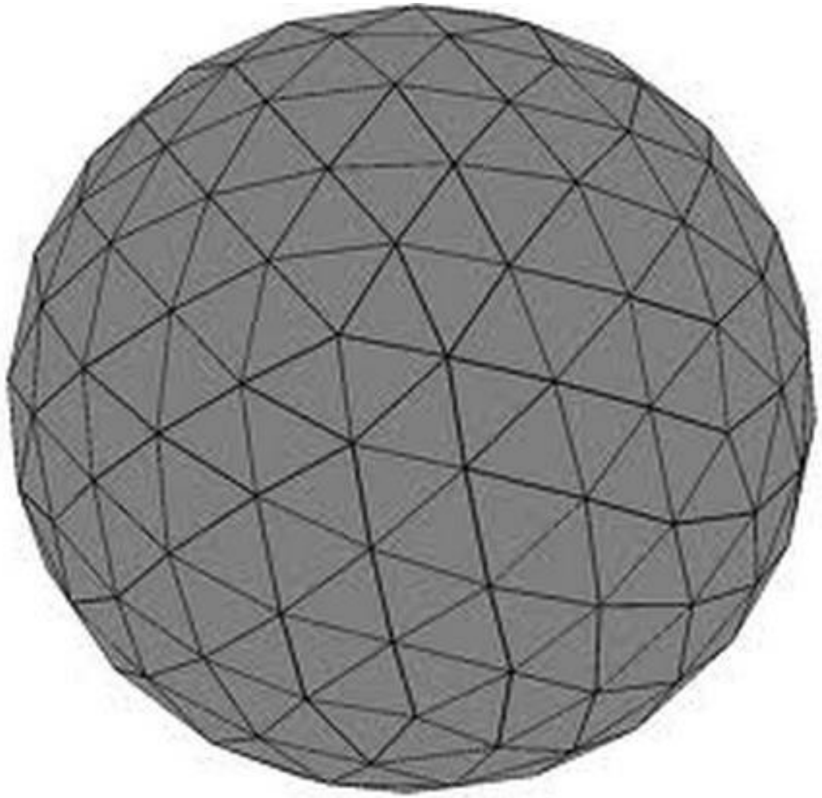
- Flip algorithms
- Incremental
- Divide and conquer
- Sweepline

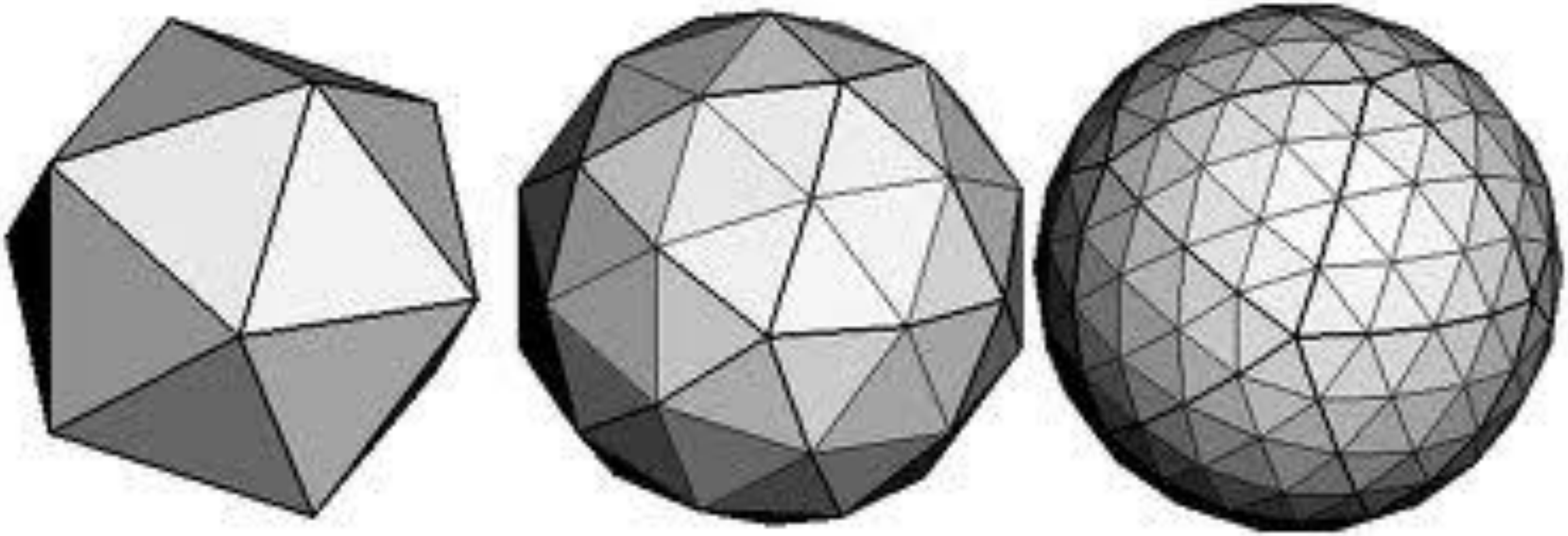




Triangulation

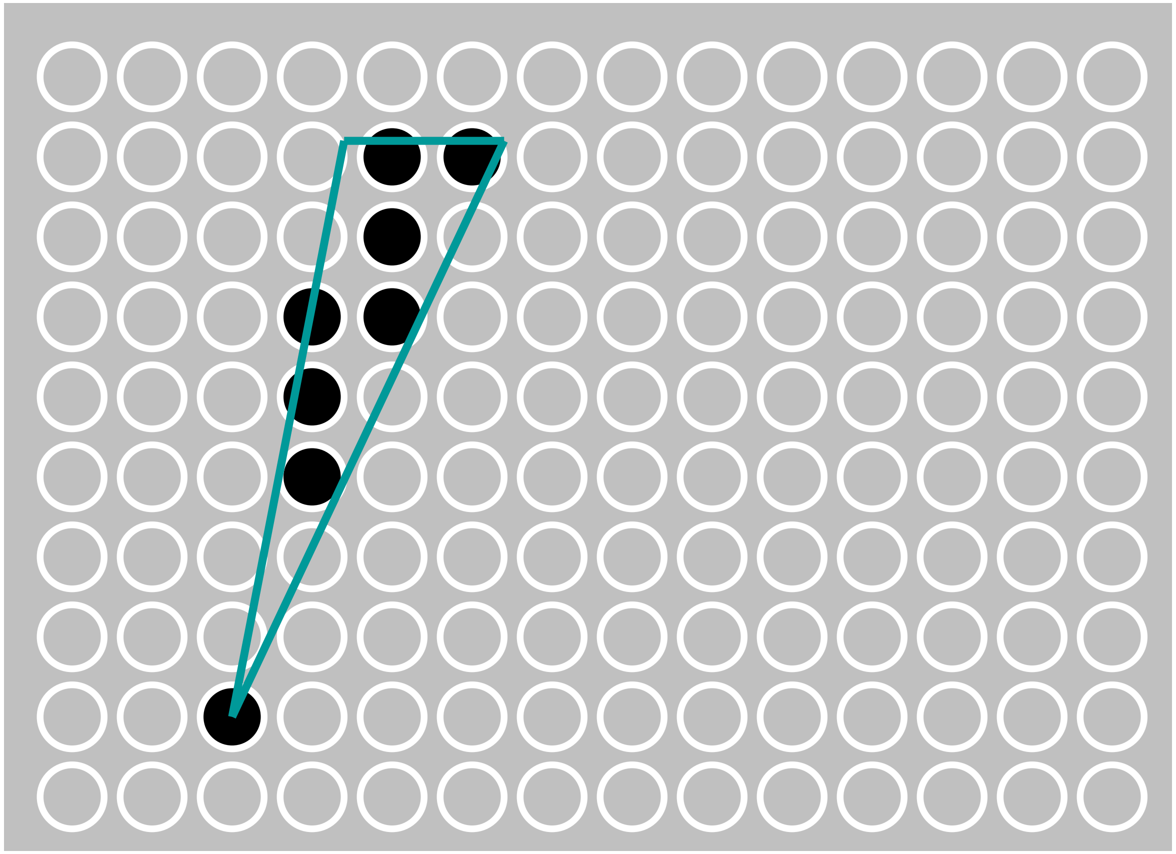






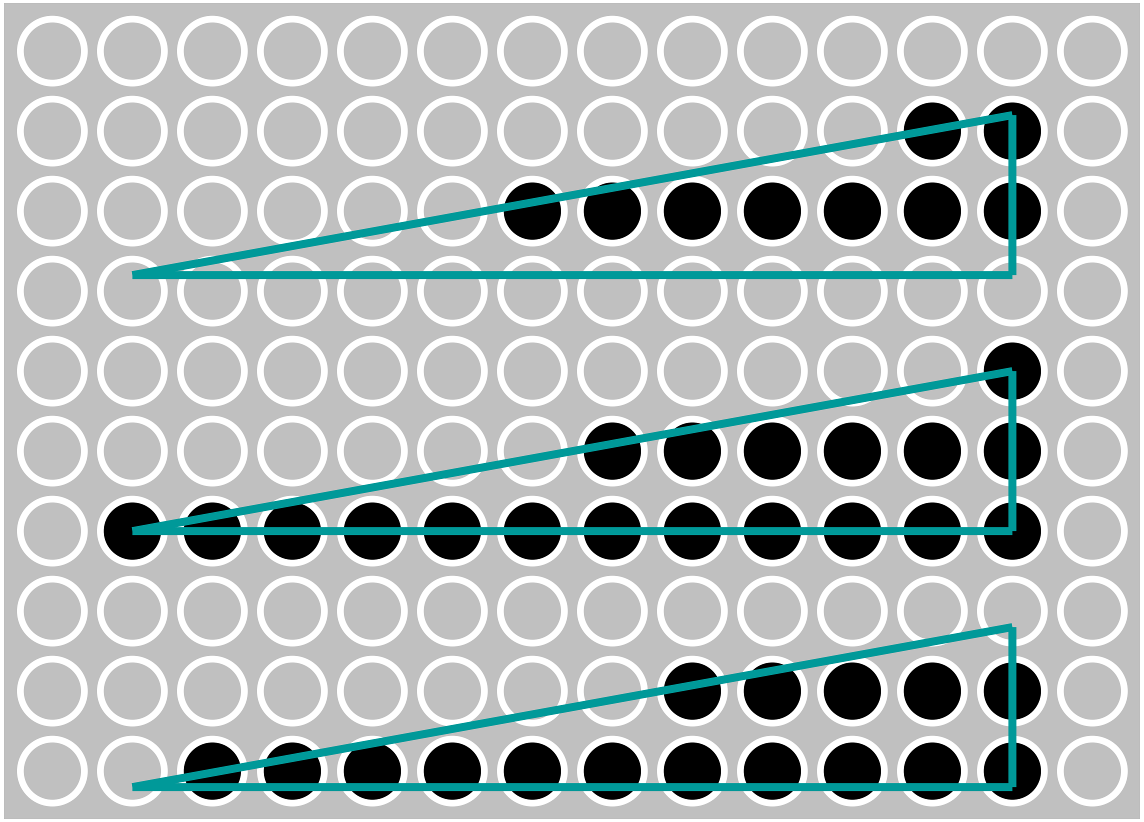
Triangle rasterisation issues

- Shape issue: Too thin to fill



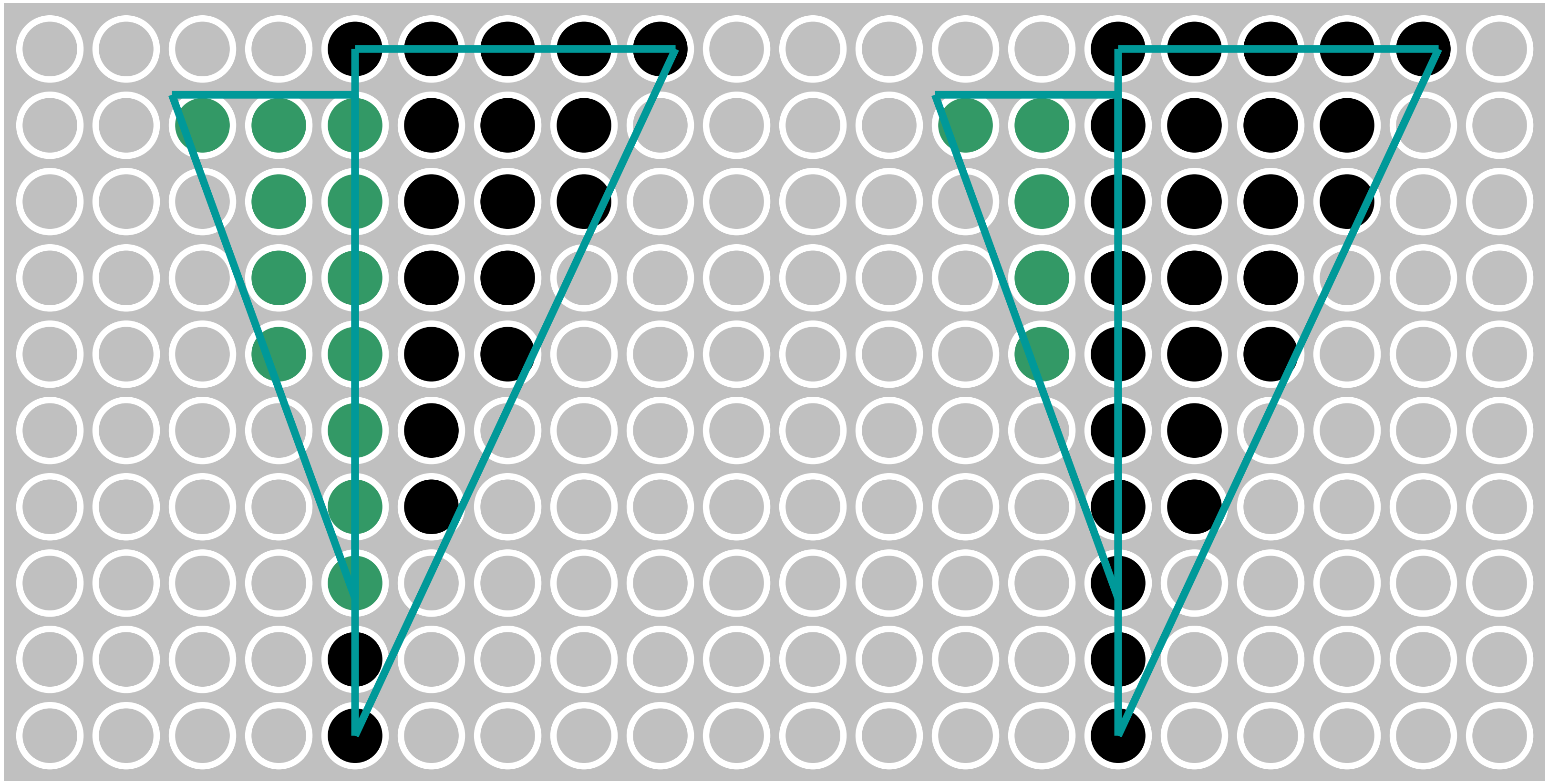
Triangle rasterisation issues

- Position issue: different at different positions



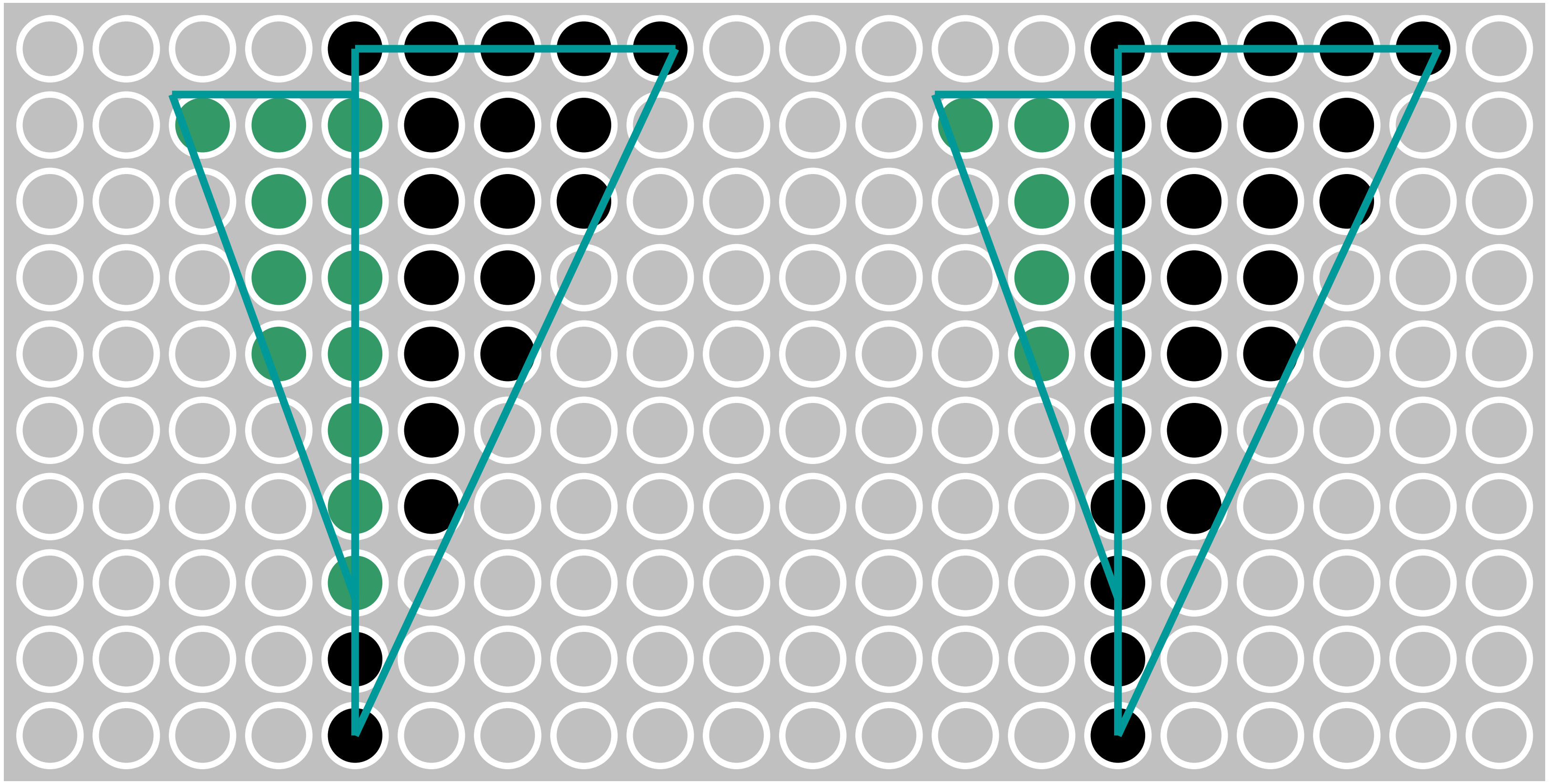
Triangle rasterisation issues

- Shared edge issue: ordering

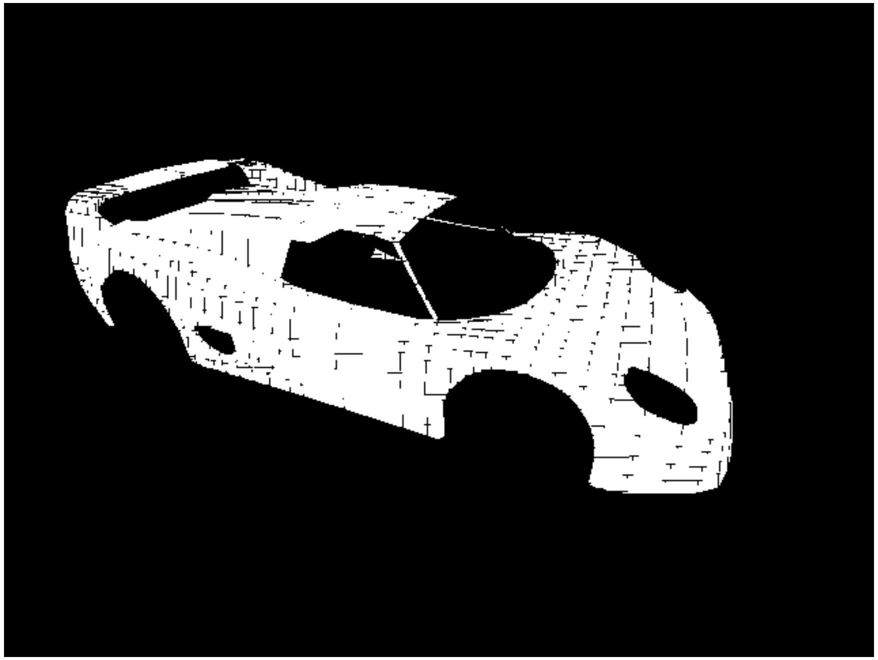


Triangle rasterisation issues

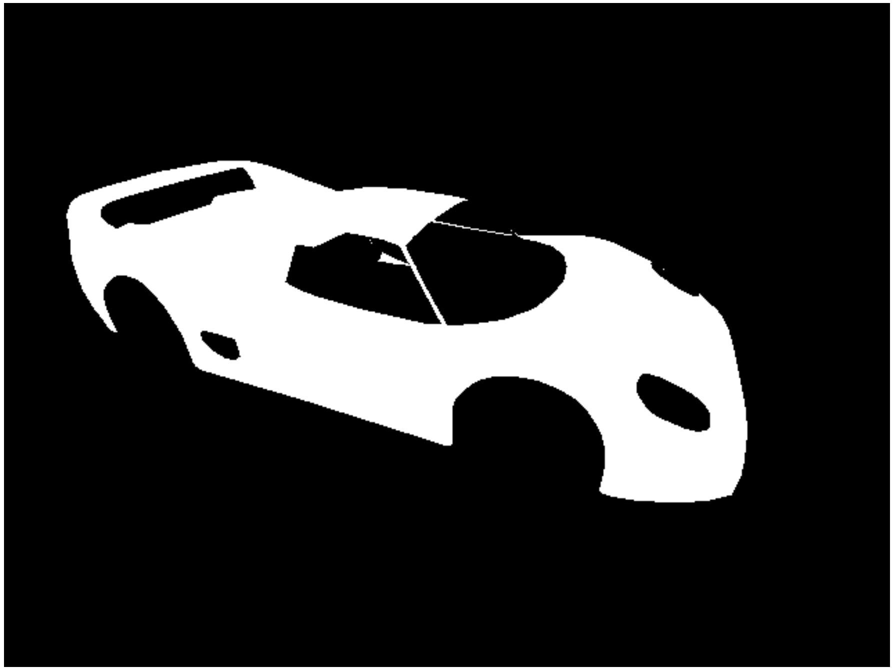
- Shared edge ordering



Triangle rasterisation issues



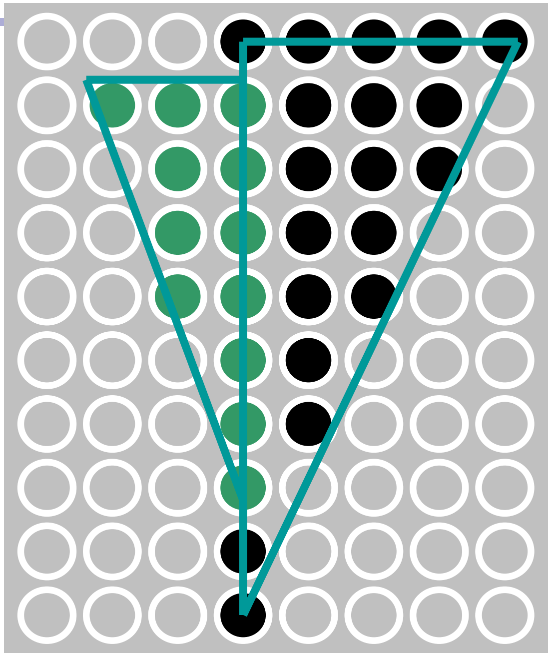
Shared edge issue: gaps



Shared edge issue: fixed

Queen Mary
University of London

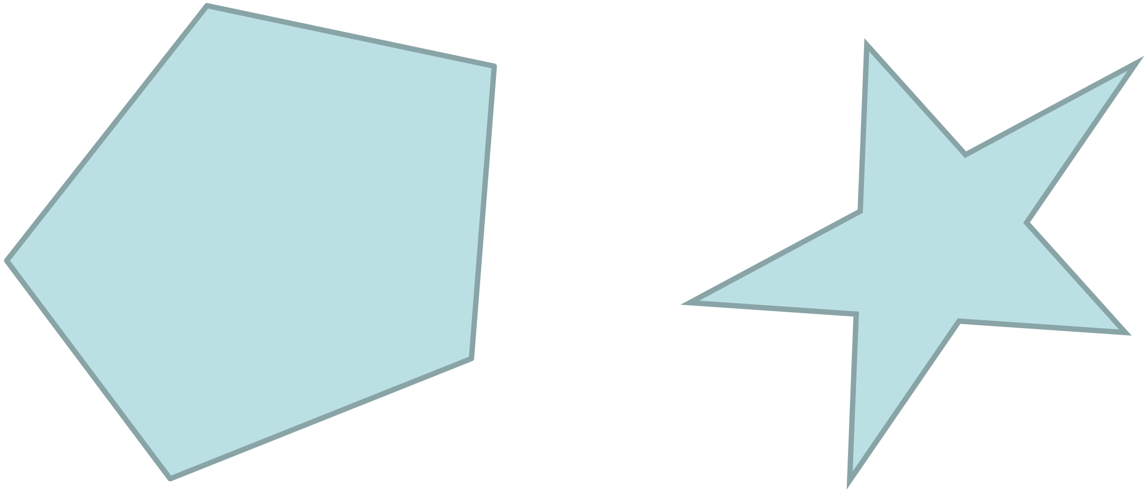
Triangle rasterisation issues (summary)



- Pixels inside the triangle edges
  - should be lit
- Pixels exactly on the edge
  - Draw them: order of triangles matters
  - 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

Filling convex shapes

Why do we want convex shapes for rasterisation?

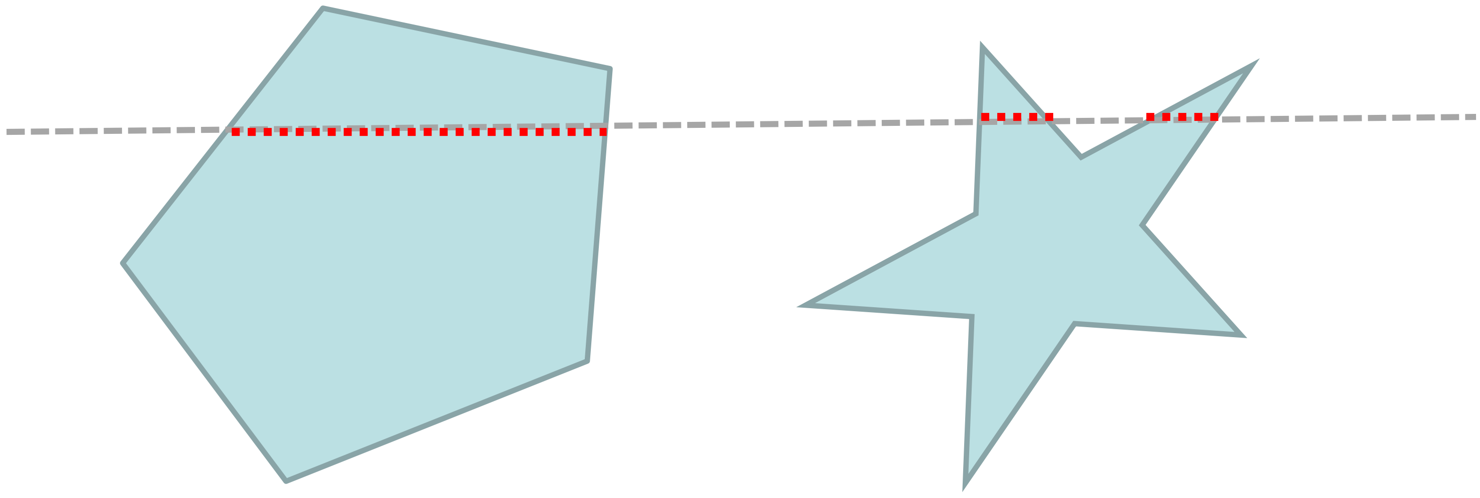


convex shape **non convex shape**

Filling convex shapes

Why do we want convex shapes for rasterisation?

One good answer is because, in a convex shape, any scanline is guaranteed to contain at most one segment or *span*

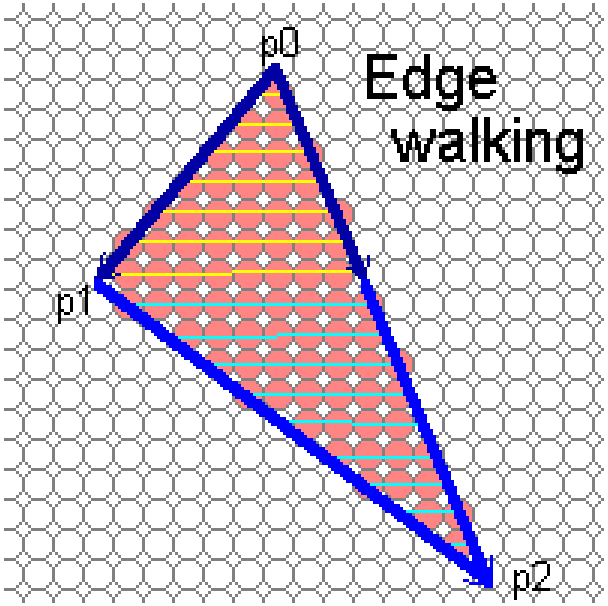


convex shape **non convex shape**

Rasterising triangles

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

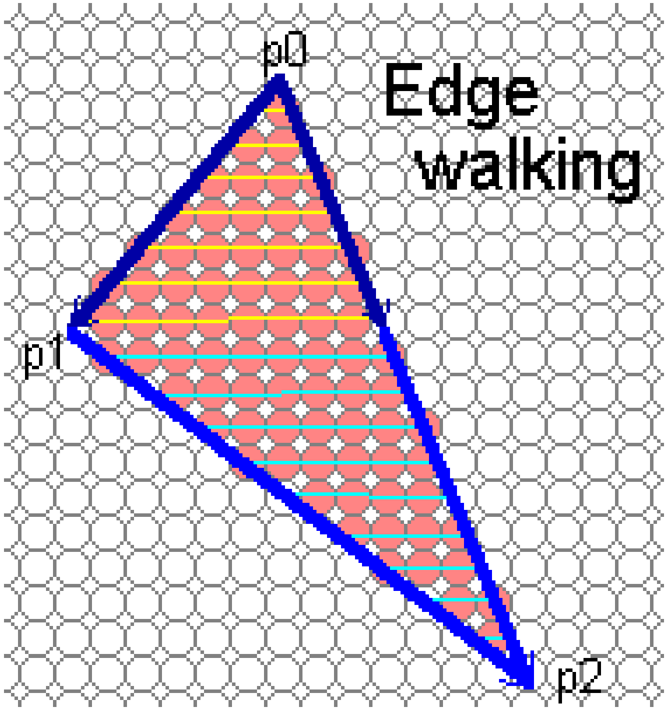
- Edge walking: basic idea
  - draw edges
    - interpolate colors down edges
  - fill in horizontal spans for each scanline
    - at each scanline, interpolate edge colors across span



Queen Mary University of London

Edge walking

- Order the three triangle vertices in x and y
  - Find the middle vertex in y dimension and find 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, all the pixels in-between
  - Until breakpoint (middle vertex) or bottom vertex is reached
- Advantage
  - can be made very fast (optimised)



Edge equations

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

    $Ax + By + C = 0$

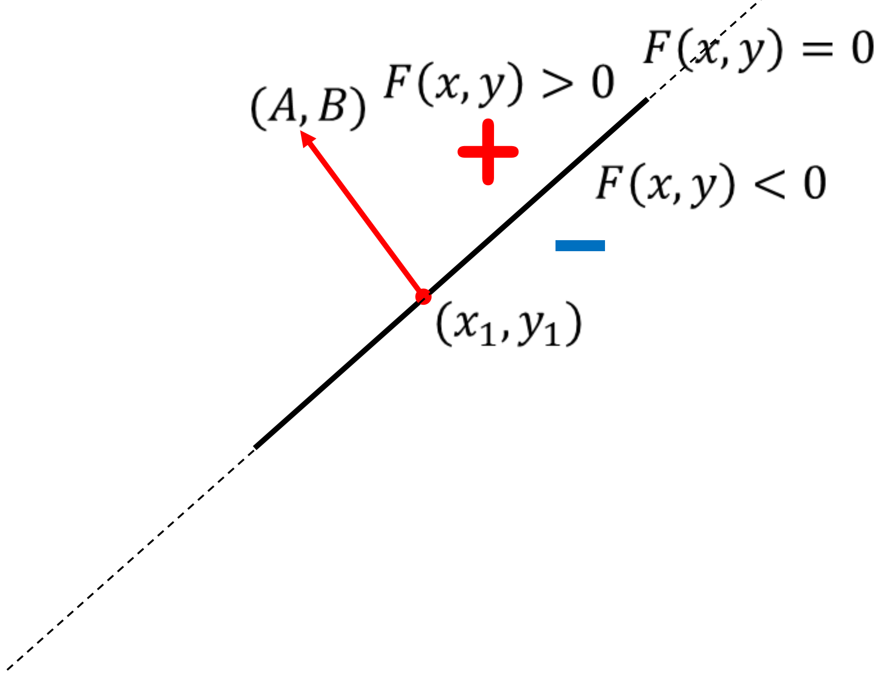
  - Given a point ($\mathbf{x},\mathbf{y}$), plugging $\mathbf{x}$ & $\mathbf{y}$ into this equation tells us whether the point is:
    - <u>on the line</u>: $Ax + By + C = 0$
    - <u>“above” the line</u>: $Ax + By + C > 0$
    - <u>“below” the line</u>: $Ax + By + C < 0$

Edge equations

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

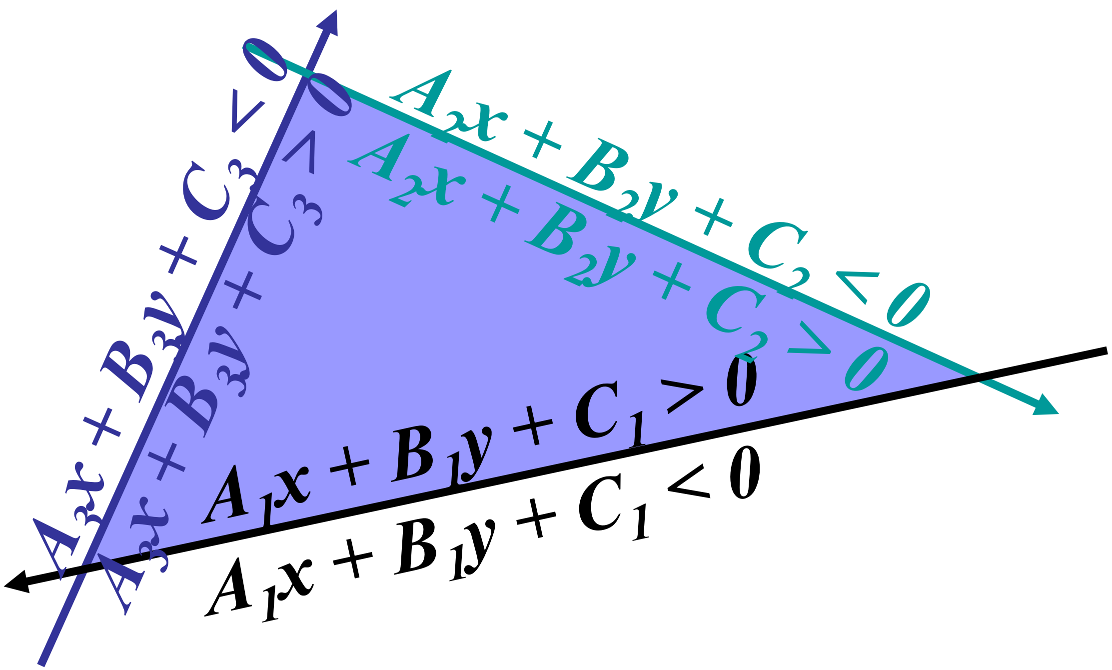
2D line equation:

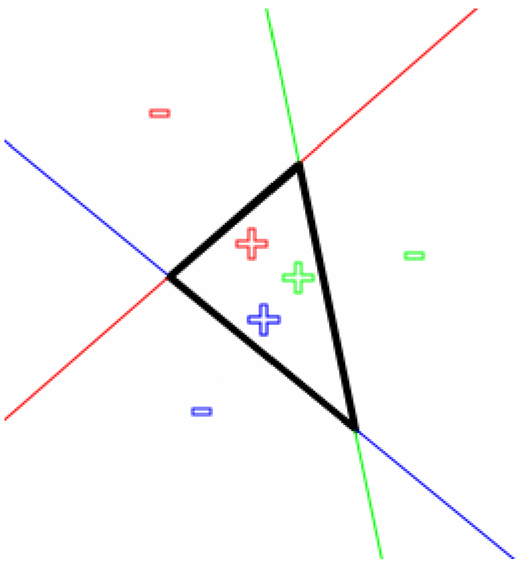
$$F(x,y) = A(x - x_1) + B(y - y_1) = 0$$



Edge equations

- For an edge of 2 vertices, take the third vertex as in the positive half-space, a triangle can be defined as the intersection of three positive half-spaces

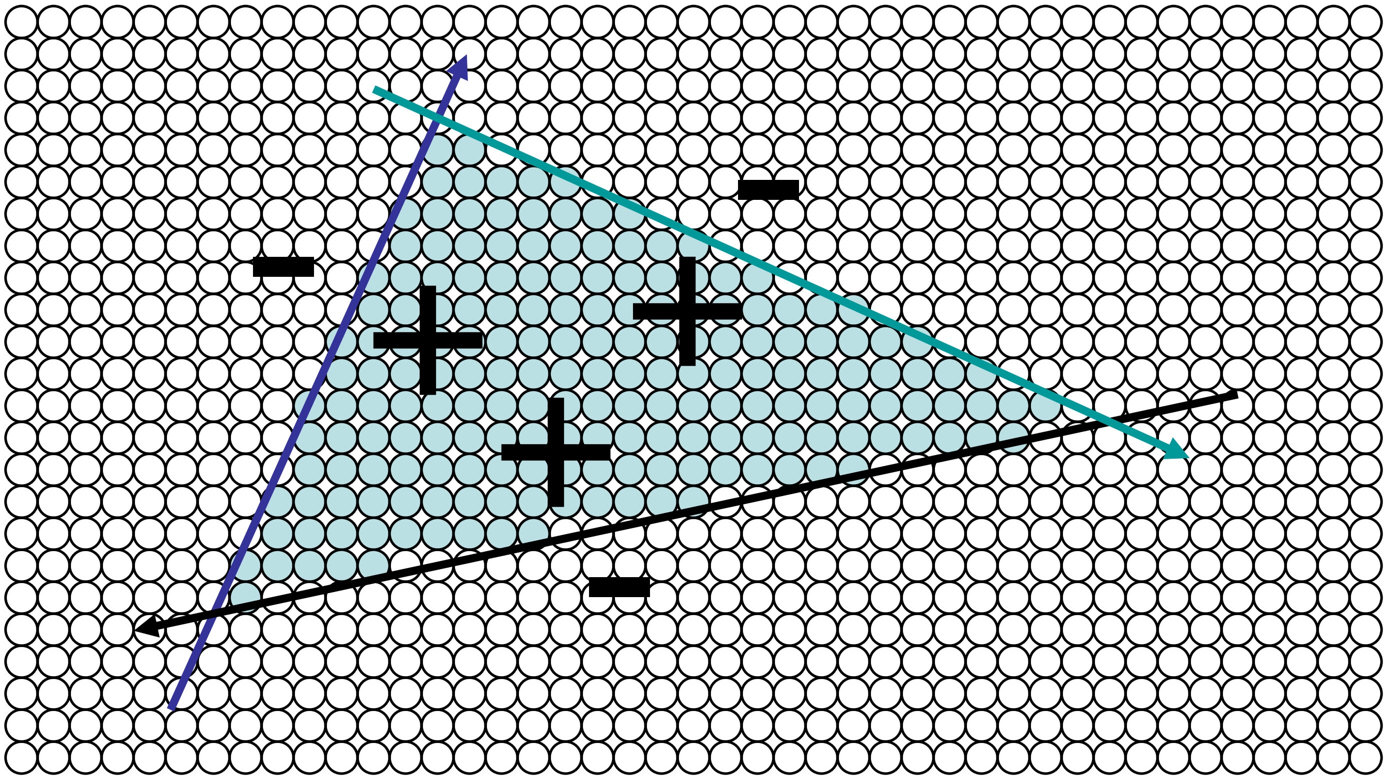




Queen Mary
University of London

Edge equations

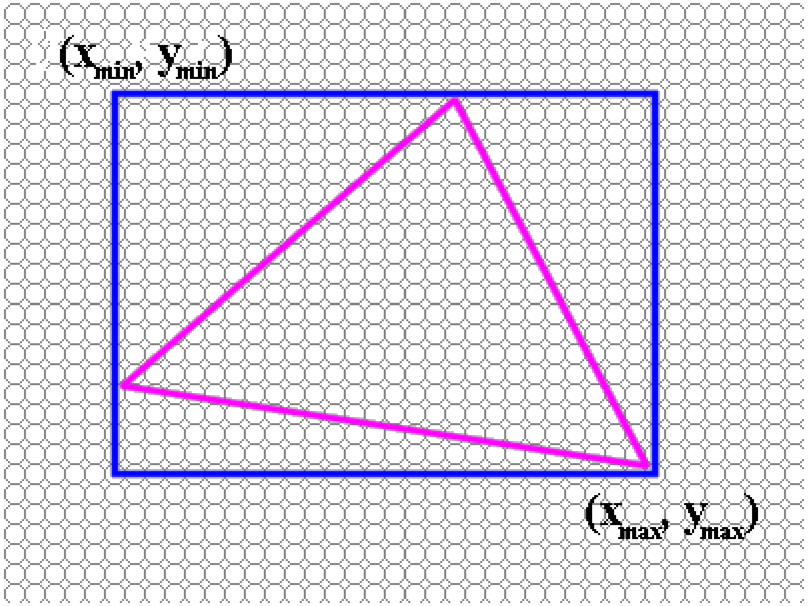
- So…simply turn on those pixels for which all edge equations evaluate to > 0



Using edge equations

How would you implement an edge-equation rasteriser ?

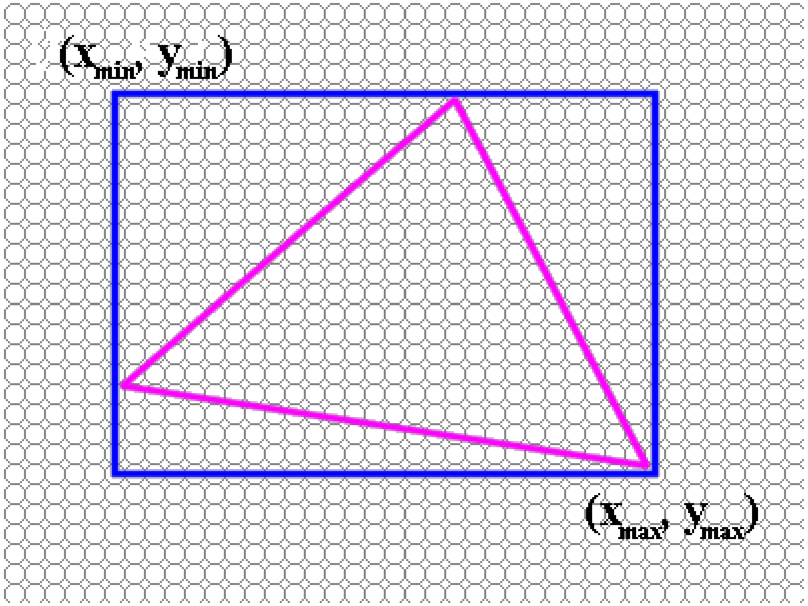
- *Which pixels do you consider?*
- *How do you compute the edge equations?*
- *How do you orient them correctly?*



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?*



Which pixels?
compute min, max bounding box

Edge equations?
compute from vertices

Orientation?
ensure area is positive

Queen Mary
University of London

Edge equations

- We can find edge equation from two vertices

- Given three corners $\mathbf{P_0, P_1, P_2}$ 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: $\mathbf{[P_0\,P_1], [P_1\,P_2], [P_2\,P_0]}$)

- *To make sure that sign is positive?* $\boldsymbol{Ax + By + C = 0}$
  → Test, and flip if needed ($\boldsymbol{A= -A, B= -B, C= -C}$)

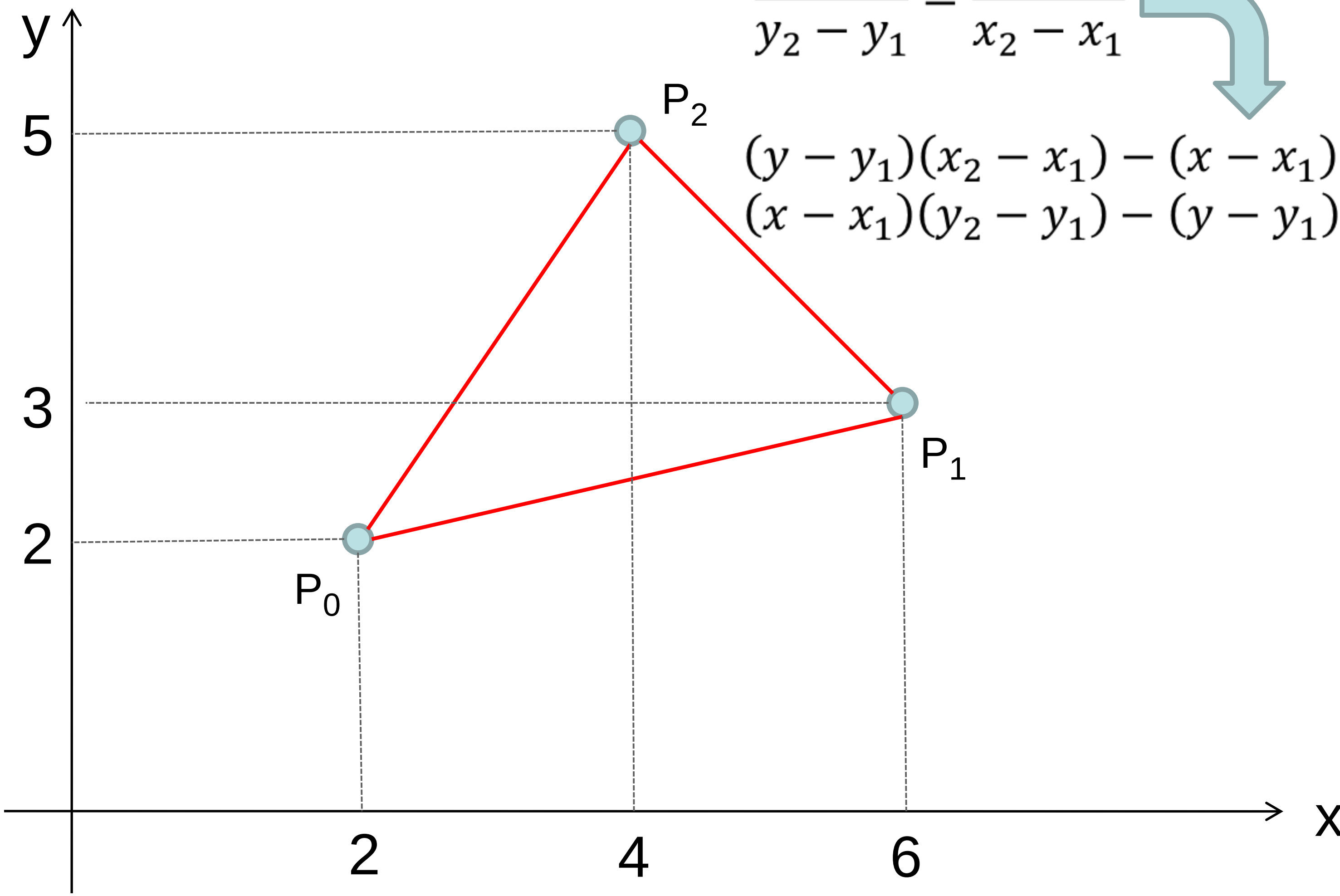
Queen Mary University of London

Edge Equation of 2 Vertices

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

$$(y - y_1)(x_2 - x_1) - (x - x_1)(y_2 - y_1) = 0$$

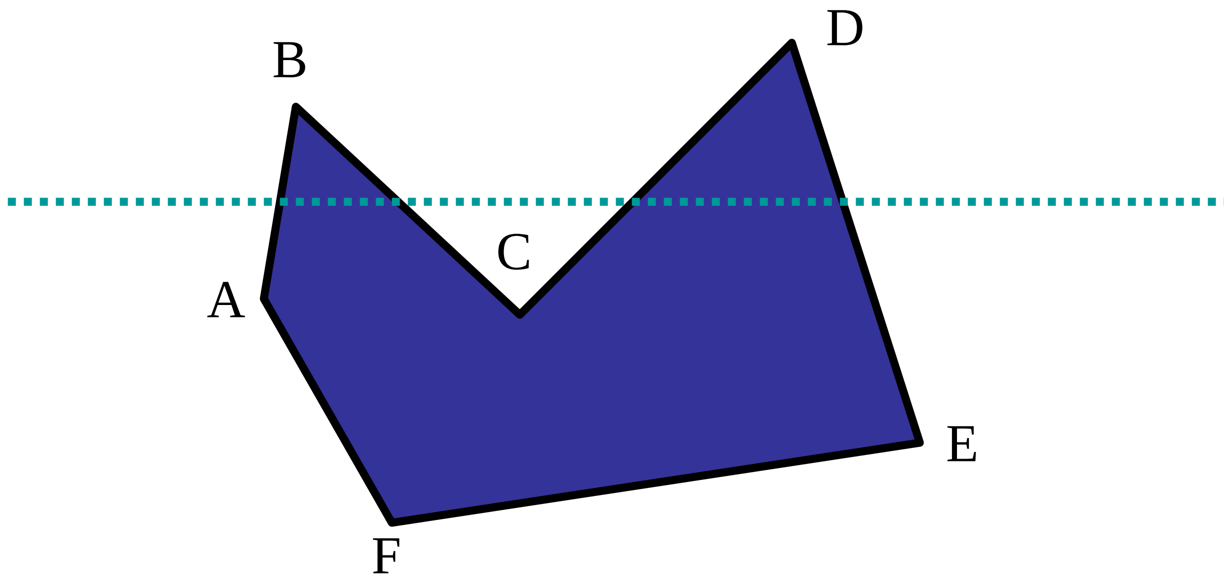
$$(x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1) = 0$$



Queen Mary University of London

General polygon rasterisation

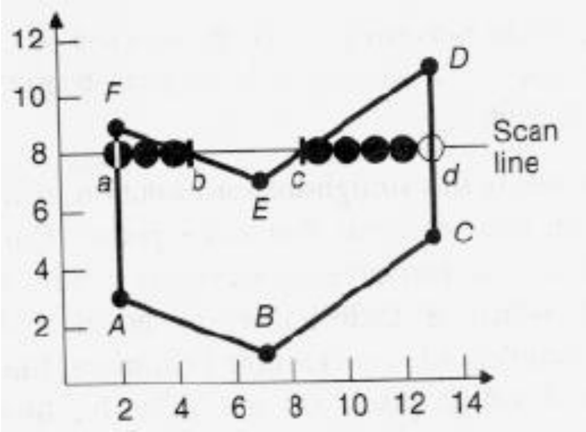
- Consider the following polygon:



- 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

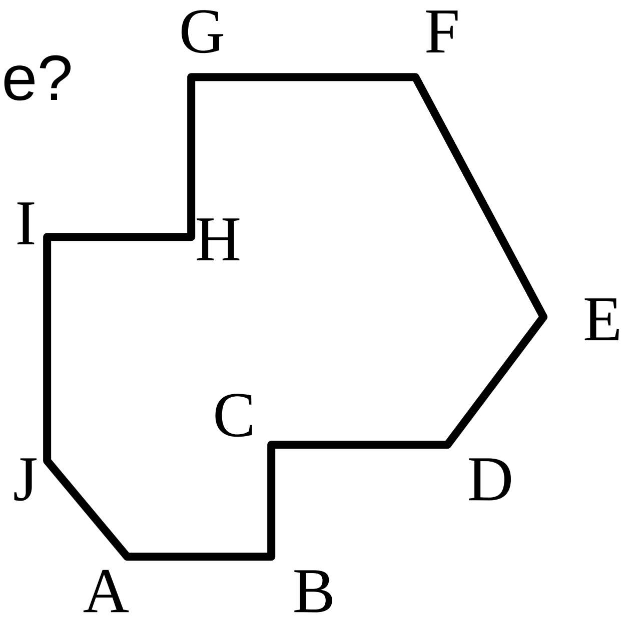



```
for each scanline
    edgeCnt = 0;
    for each pixel on scanline left to right
        if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
        // draw the pixel if edgeCnt odd
        if (edgeCnt % 2)
            setPixel(pixel);
```

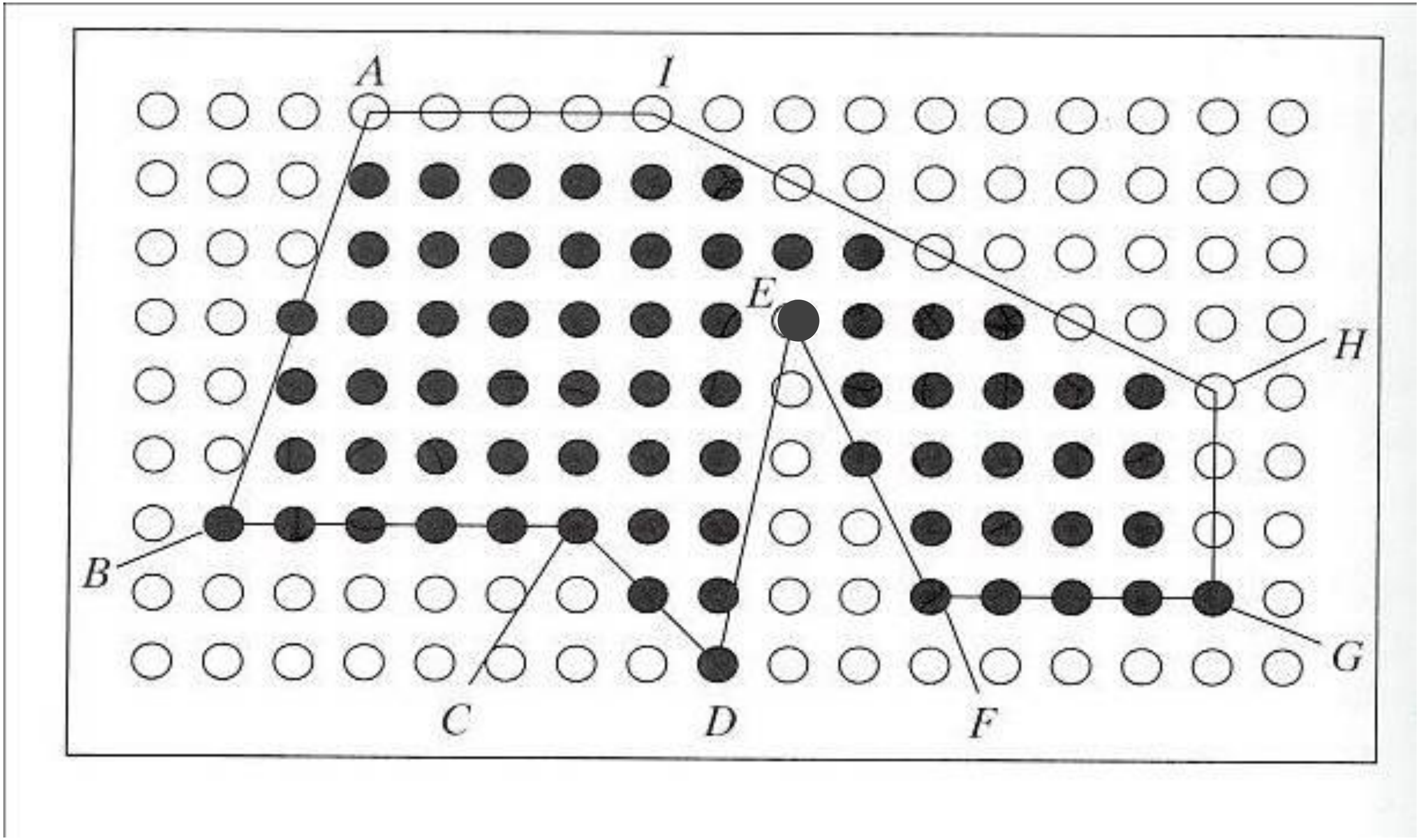
Queen Mary University of London

General polygon rasterisation

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



An idea: edge walking with polygon…



Queen Mary
University of London

Faster polygon rasterisation

- Problem: how can we optimise the parity test code?

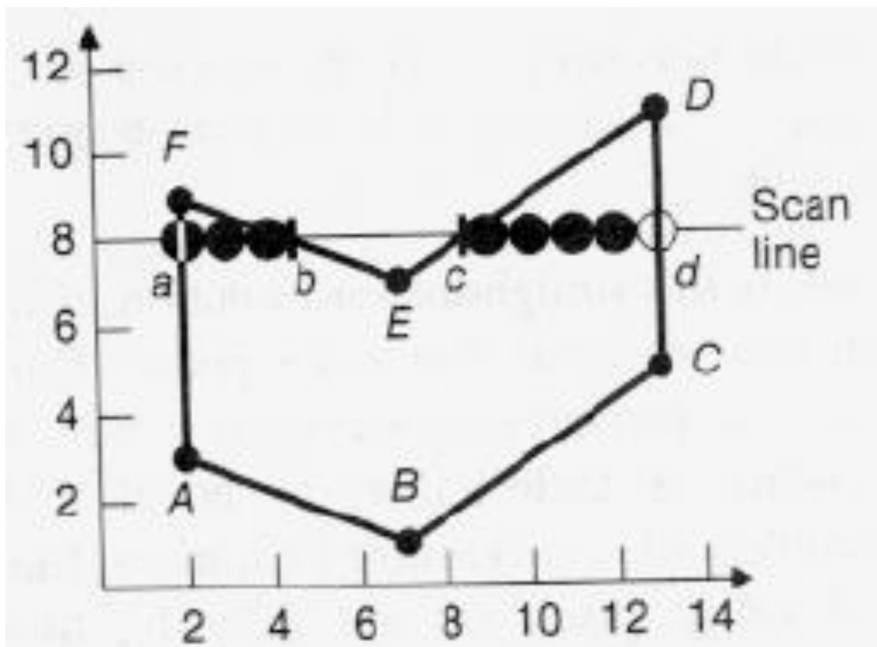
```
for each scanline
    edgeCnt = 0;
    for each pixel on scanline left to right
        if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
        // draw the pixel if edgeCnt odd
        if (edgeCnt % 2)
            setPixel(pixel);
```

Note: big cost → testing pixels against each edge

- Solution: active edge table *(AET)*

Active edge table: edge coherence

- Observation: edge coherence
  - 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

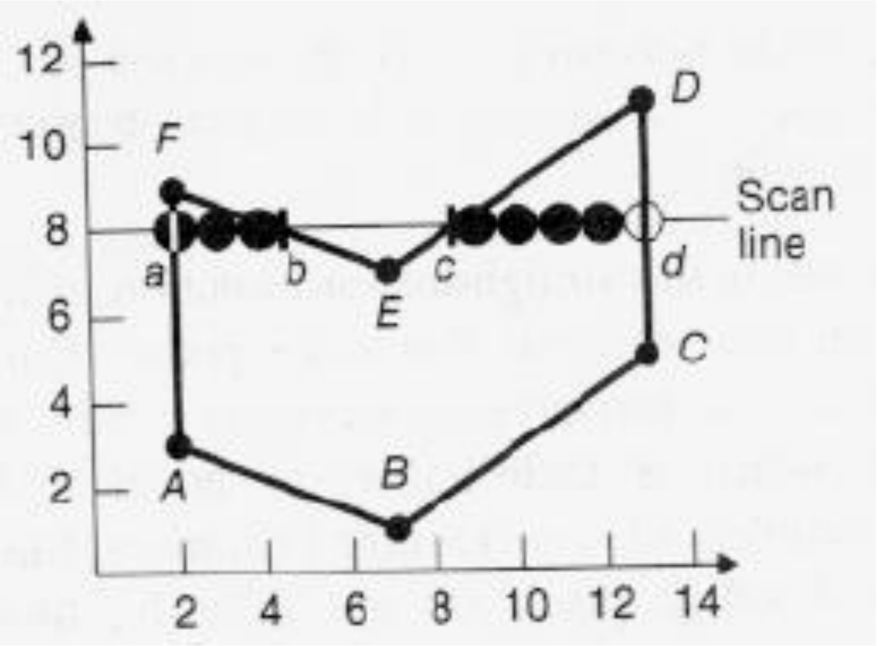
Active edge table

- 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 from left to right. 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 new scanline.

Active edge table: Algorithm

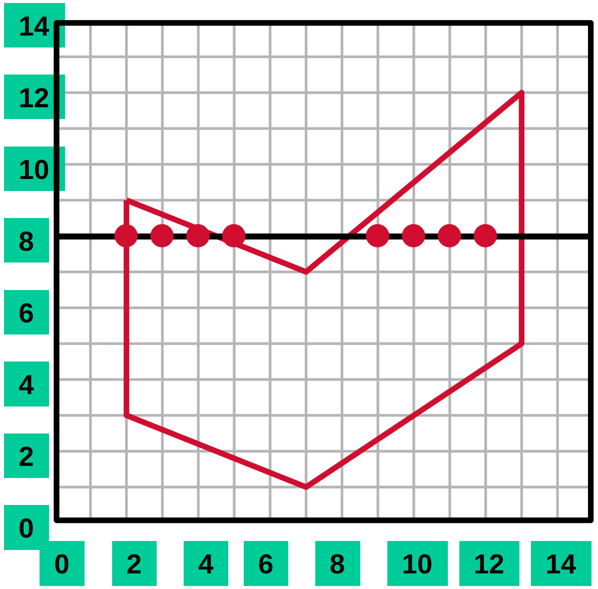
for scanline from bottom Y_{min} to top Y_{max} …

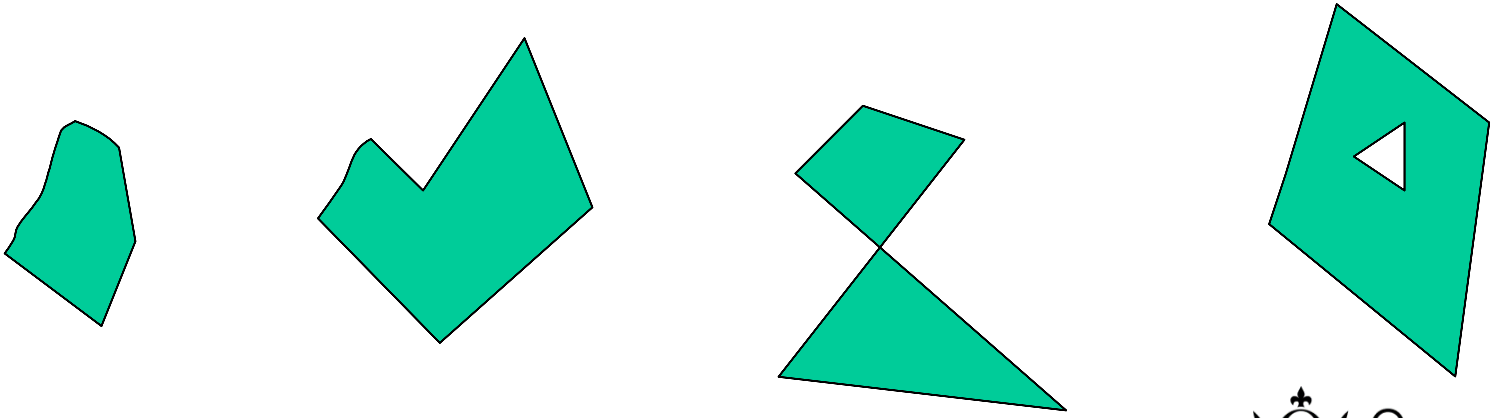
- Sort all edges by their minimum y coordinate
- Starting at bottom, add edges with $Y_{min} = 0$ to AET
- For each scanline:
  - Sort edges in AET by X intersection
  - Walk from left to right, setting pixels by parity rule
  - Increment scanline
  - Retire edges with $Y_{max} < Y$
  - Add edges with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



Summary: filling polygons

- Use scan lines
- Edge coherence
- Finding intersections

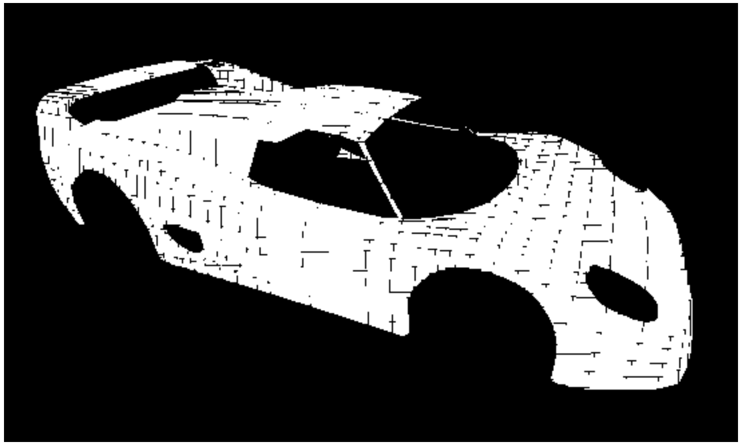




Questions

- What are the properties of a polygon that make it easy to rasterise? Why is it the case?

- The Figure shows an object that has been imperfectly rasterised (the small black segments that appear on the body of the car should not be there). In your opinion, what has gone wrong?



What did we learn today?

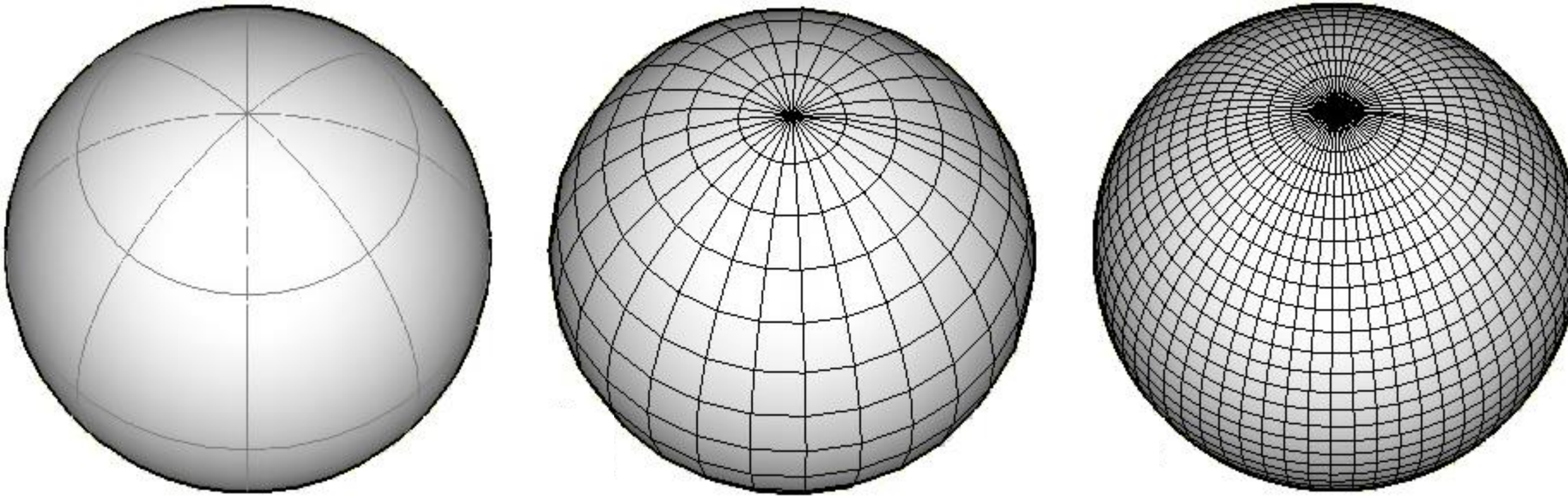
- Polygons
- Polygon rasterisation
- Triangulation
- Edge walking
- Edge equations
- Active edge table

Question …

In computer graphics, objects such as spheres are usually approximated by simpler objects constructed from flat polygons (polyhedral).

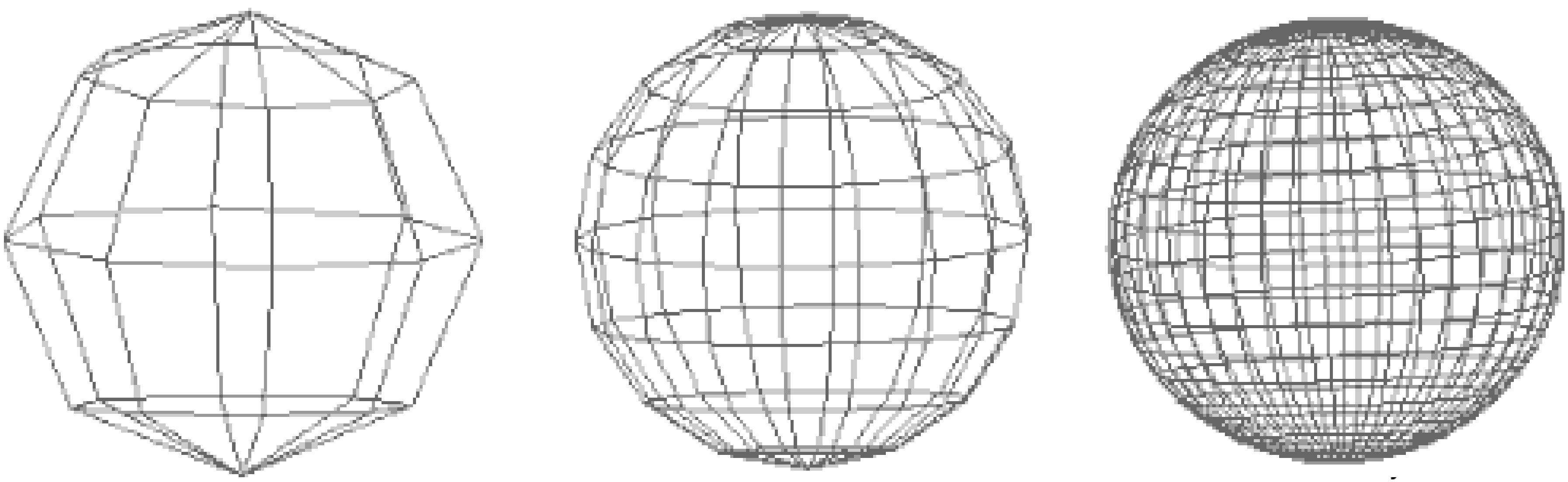
Using lines of longitude and latitude, define a set of simple polygons that approximate a sphere.

Can you use only quadrilaterals or only triangles?

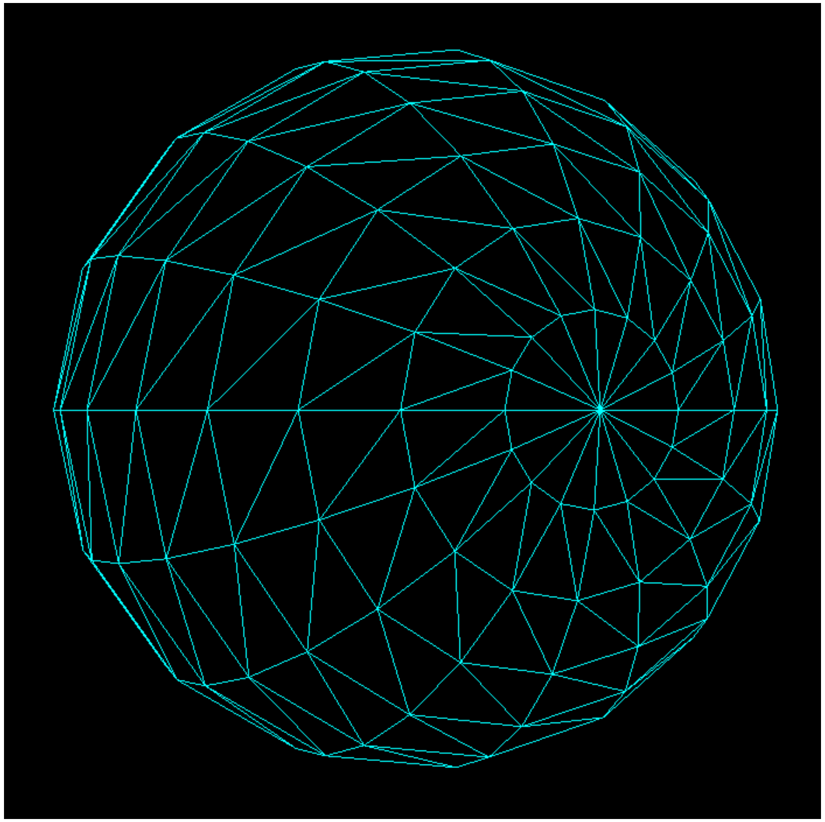


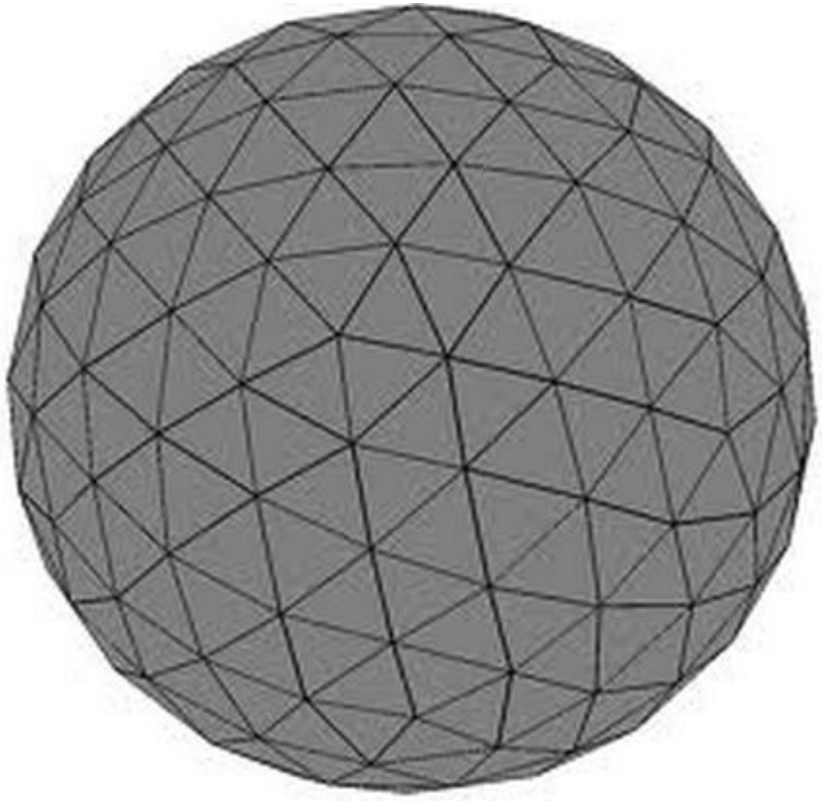
Surfaces of spheres with lines of longitude and latitude

Polygon mesh approximation



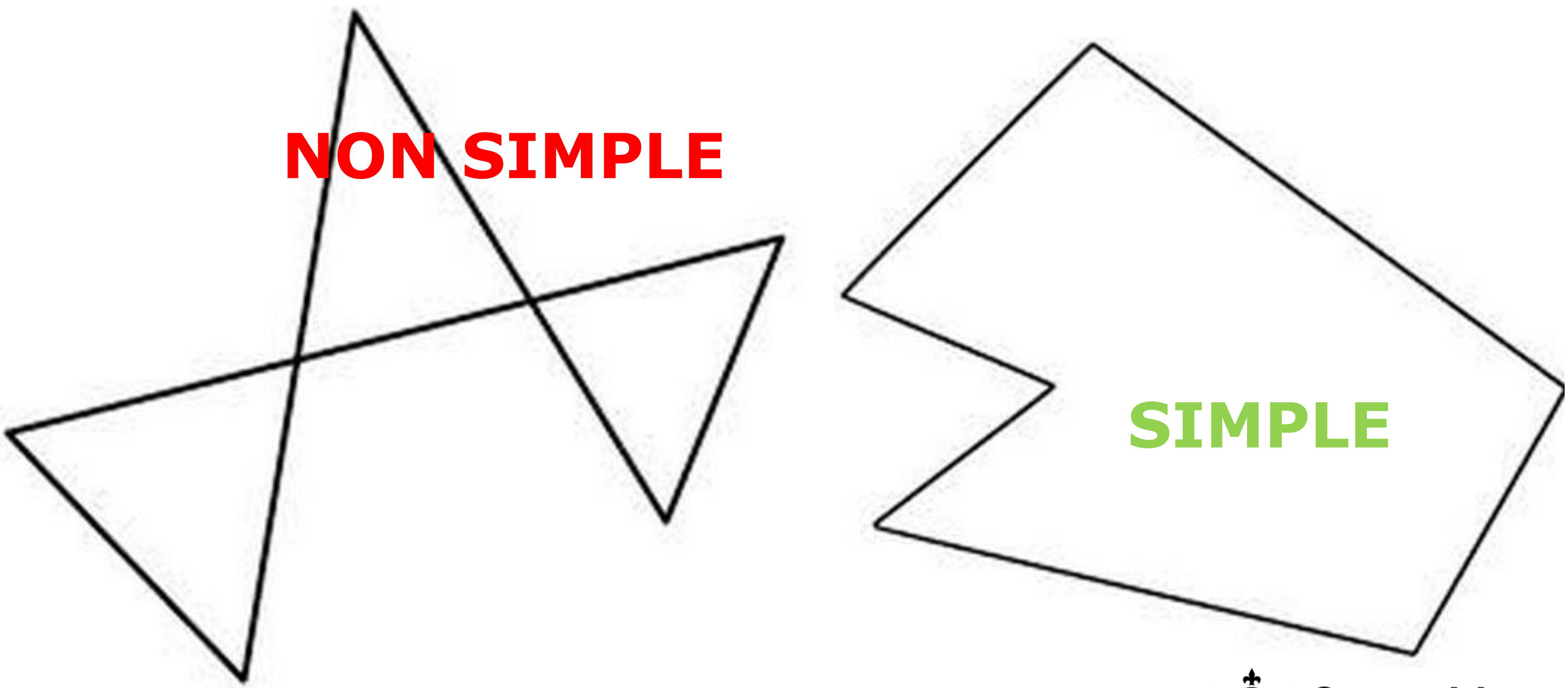
Triangulation





Question …

- Devise a test to determine whether a two-dimensional polygon is simple.

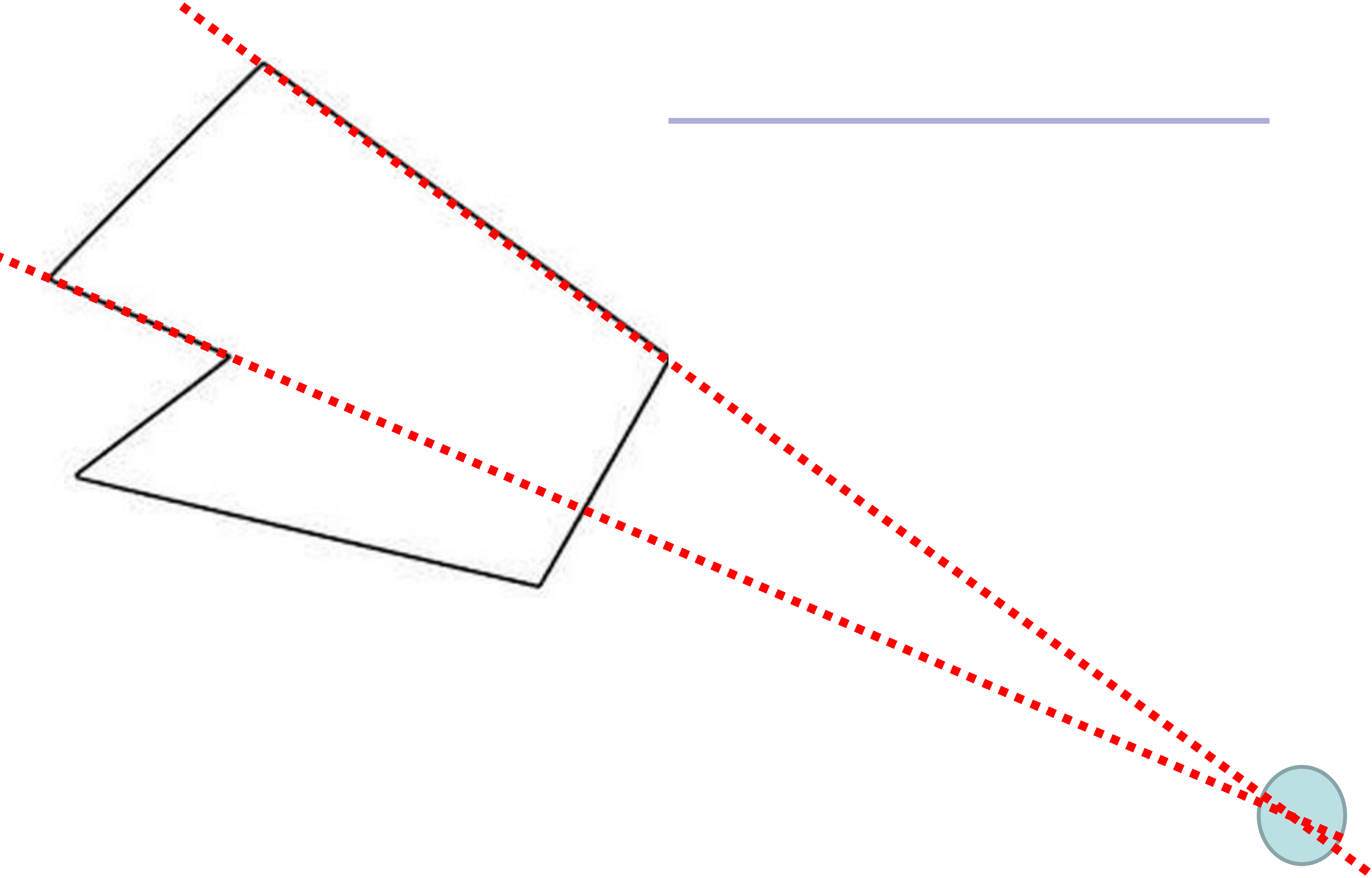


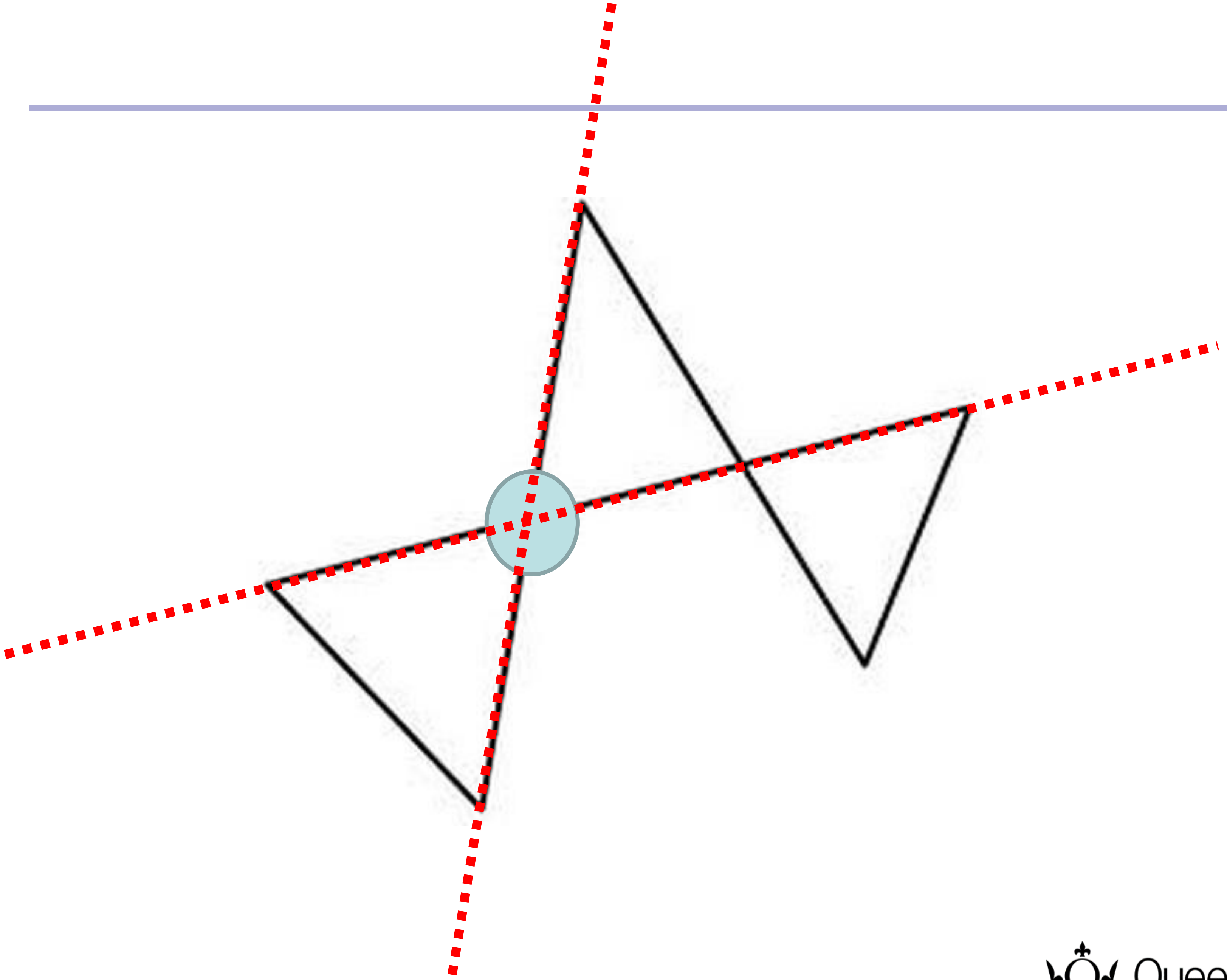
Queen Mary University of London

Question …

- Devise a test to determine whether a two-dimensional polygon is simple.

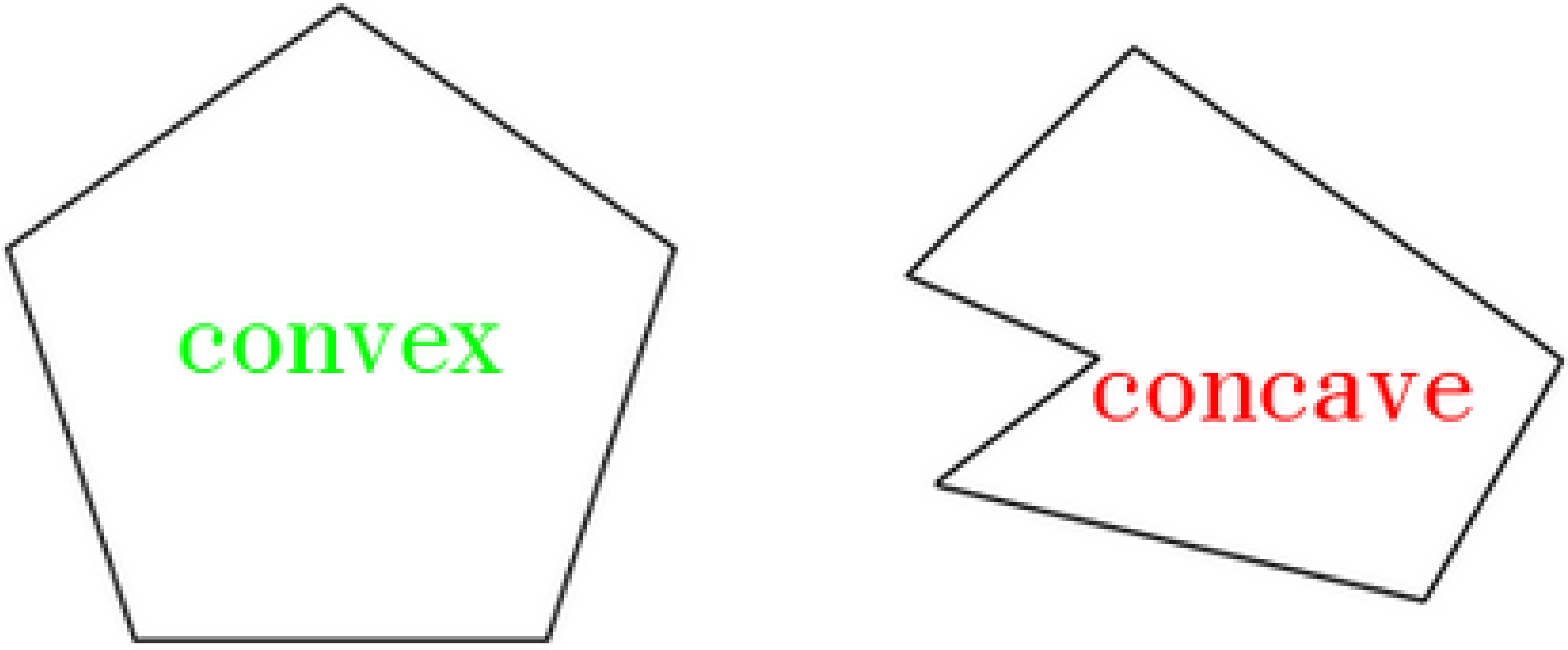
A simple test would be to compute the intersections of all pairs of lines that are determined by the edges of the polygons. We could then test if any of these intersections lie on the edges as opposed to the lines.





Question …

- Devise a test for the convexity of a two-dimensional polygon.

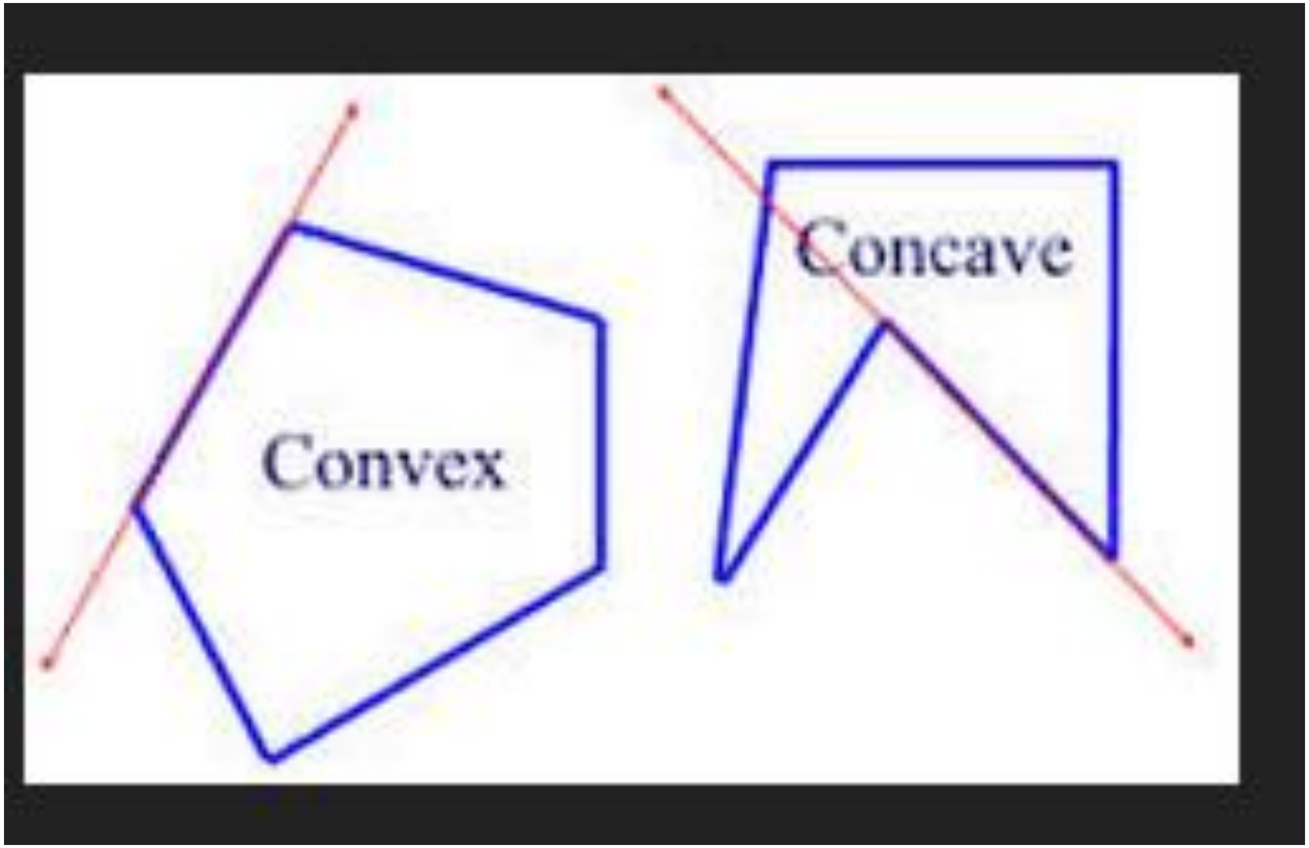


Question …

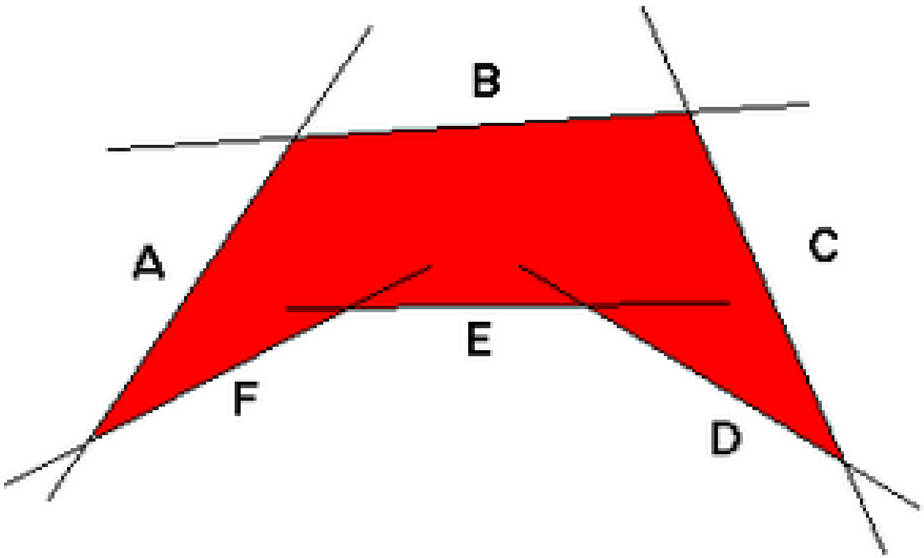
- Devise a test for the convexity of a two-dimensional Polygon

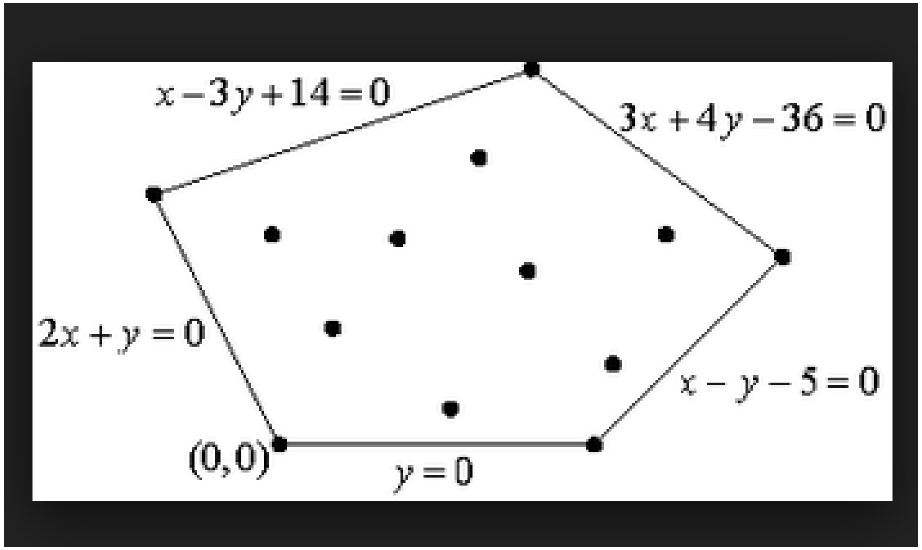
Consider the lines defined by the sides of the polygon.
We can assign a direction for each of these lines by traversing the vertices in a counter-clockwise order.
One very simple test is obtained by noting that any point inside the object is on the same side of each of these lines. Thus, if we substitute the point into the equation for each of the lines ($ax+by+c$), we should always get the same sign.

Queen Mary
University of London



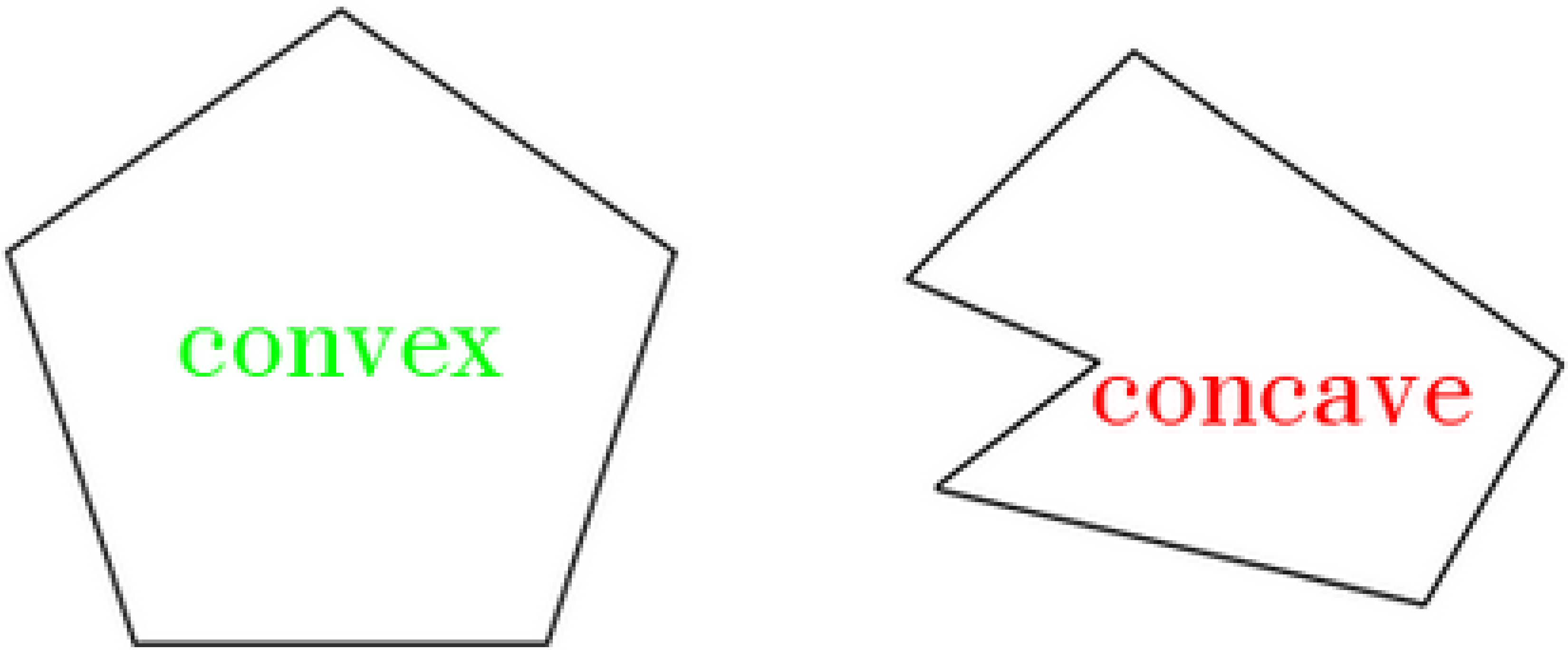
Queen Mary
University of London


B
A
C
E
F
D


$x-3y+14=0$
$3x+4y-36=0$
$2x+y=0$
$x-y-5=0$
$(0,0)$
$y=0$

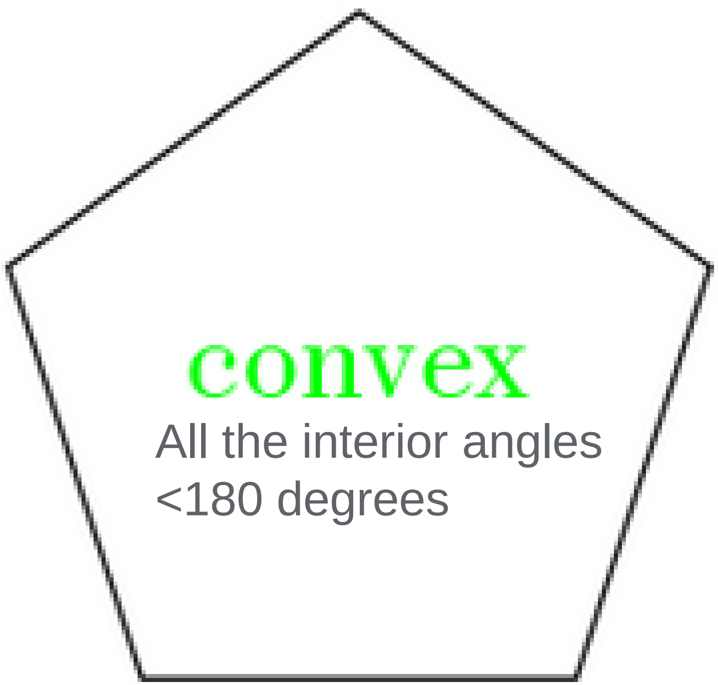
Question …

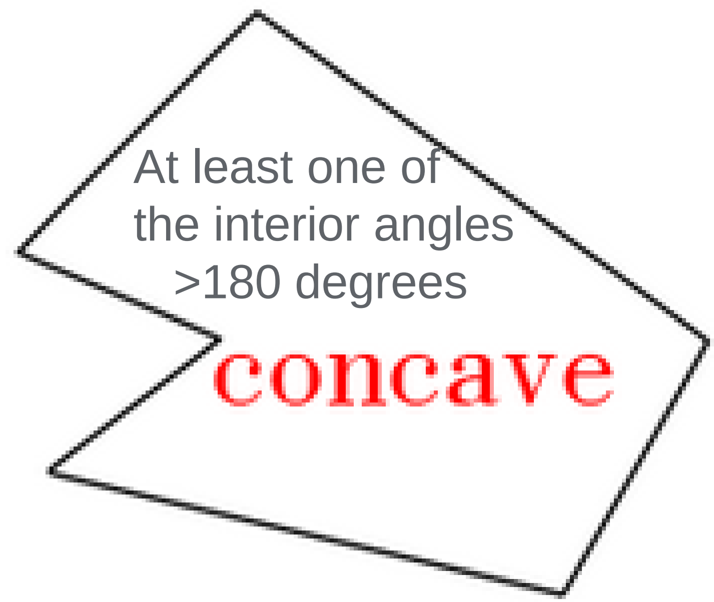
- How can we use angles to know if a polygon is convex or concave?



Question …

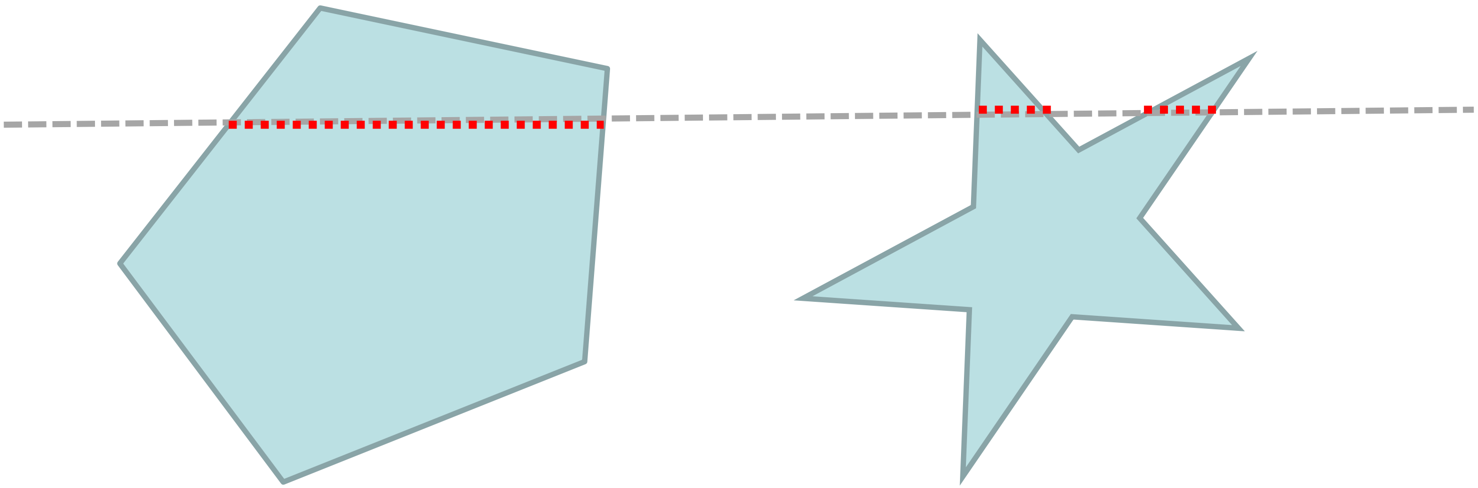
- How can we use angles to know if a polygon is convex or concave?





Question …

- For polygon filling, why do we need to know if it is convex or concave?



convex shape **concave shape**

Queen Mary University of London

Question …

- How can we convert a concave polygon into convex polygons?