Human Graphics Pipeline

As a group of people, you will walk through a simplified version of the programmable graphics pipeline (*fig 1*) to draw a triangle “on-screen” through paper-based resources by following the instructions below. The instructions refer to “Person A”, “Person B” etc., but you may alter who acts as which part of the pipeline depending on the amount of people available.

Person A - Vertex Processor
Person B - Primitive Assembler
Person C - Edge Clipper (optional)
Person D - Rasteriser
Person E - Fragment Shader

You will need:
A group of up to 5 people
Paper resources (provided by lecturer, or available as a download from the website)
Washable marker pens if using acetate resources, or pens for paper resources

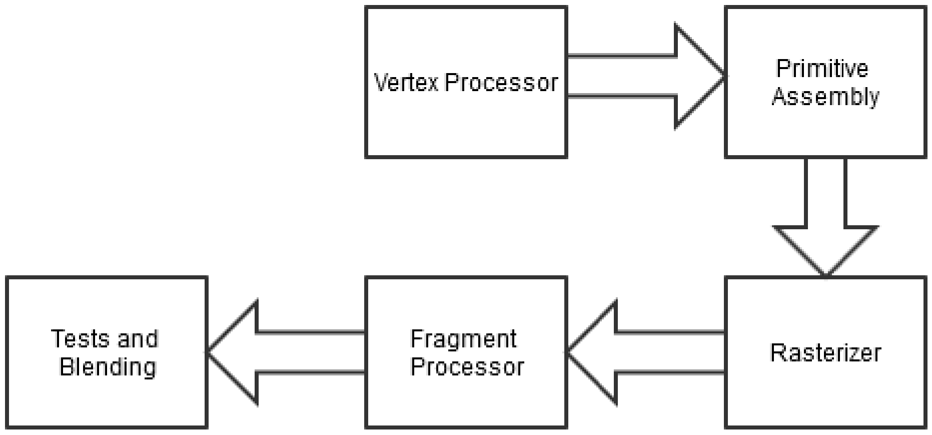

Figure 1 - A simplified version of the programmable graphics pipeline

Step 1 - Vertex Processing

Have Person A define three points for the primitive to be drawn. Note down the coordinates in 2D space on the window space grid (-1 to 1) and the order in which they are to be drawn.

Step 2 - Primitive Assembly

Have Person A (vertex processor) pass their vertices on to Person B (primitive assembler) in the order that the points should be drawn. The primitive assembler will plot the points on the window space grid and connects them with lines in the order that they were received.

Step 3 - Clipping (optional)

If you are using the clipping world space grid, you may have placed vertices outside of the screen space. If this is the case, clipping can be performed so that data outside of the screen space is not unnecessarily drawn.

Person C will create a new list of vertices for the clipped primitive. Start with the first vertex in your list of primitive vertices. If the vertex lies within world space, add it to your list. Follow the lines and mark any intersection points that occur between the edge of the world space and the primitive lines. Add these intersection points to your new list in the order that they occur (from nearest to furthest from the point you started with). Continue this process until you have covered all vertices for your original primitive.

From your new list of points, draw them as a triangle fan starting from the first vertex. For instance, in *fig 2* the first triangle is vertices 1, 2, 3, and the second is 1, 3, 4. Continue this process until all faces are drawn (you will have n-2 faces).

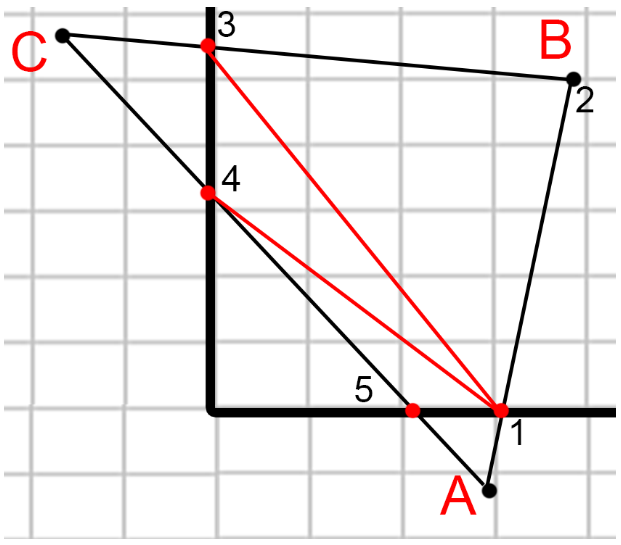

Figure 2 - An example of clipping across the bottom left corner of world space where the black triangle is the original primitive and the red lines represent the face edges

Step 4 - Rasterise to screen space

Refer to the screen space grid (pixel grid). If using acetate sheets, lay it over the window space grid (otherwise you may have to re-draw the primitive assembly results).

Person D will act as the rasteriser, who will decide whether a point resides within the triangle or not. This can be performed either::

- by eye (make an intuitive decision for simplicity)
- with a rasterising algorithm. The website provides calculators for Barycentric and Half-Space rasterisation. Refer to page 3 for instructions to perform this on paper

Step 5 - Fragment Shader

Person E will play the role of the fragment shader. The fragment shader will contain a colour that is to be applied for each pixel to be drawn. The fragment shader can use a coloured marker pen to emulate drawing pixels.

Rasterisation

Rasterisation algorithms are used to find if a point resides within a primitive to decide whether or not it should be sent to the fragment shader to be drawn to screen. Instead of checking every single pixel in screen space, you can create a bounding box around your primitive so that you only have to make decisions within it (*fig 3*)

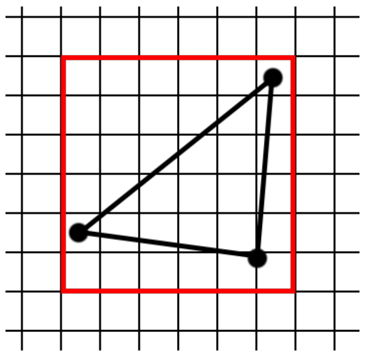

Figure 3 - A bounding box is created around a triangle, which is used to check points to rasterise

You will perform mathematical rasterisation of each pixel within the box. The centre point of each pixel will be used as the reference point (*fig 4*). To perform the rasterisation you will need the vertices of your primitive (in the order provided by the vertex processor), along with the reference point.

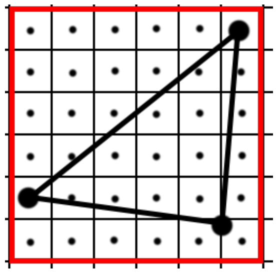

Figure 4 - Centre points of pixels within bounding box

Below are two different algorithms that rasterise with slightly different results.

Half Space

This algorithm checks which side of a line a point is on. If the point to check is on the side inside the triangle for all triangle sides, the pixel is within the triangle.

For each edge, take the points of your line (A and B) and the reference point of the pixel (P) and perform the following calculation (where X and Y refer to the X and Y position in world space):

(B.X - a.X) * (P.Y - A.Y) * (P.X - A.X)

You will now have three values (one for each line - AB, BC, and CA). If all of these values are greater than or equal to zero, or all values are less than or equal to zero, the point is within the triangle. Otherwise, it is not.

Additionally, you can perform backface culling with this. If the triangle is drawn counter-clockwise, you can cull all back-facing pixels by only checking if your resulting values are greater than or equal to zero.

Barycentric

For this method of rasterisation, you will use all of your primitive points (A, B, and C). Your pixel reference point, the centre of the pixel, is P. Perform the following calculations:

Vertex vs1 = (B.X - A.X) , (B.Y - A.Y) --- the spanning vector of edge AB
Vertex vs2 = (C.X - A.X) , (C.Y - A.Y) -- the spanning vector of edge AC

Vertex q = (P.X - A.X) , (P.Y - A.Y)

s = Cross Product(q, vs2) / Cross Product (vs1, vs2)
t = Cross Product(vs1, q) / CrossProduct (vs1, vs2)

If s and t are both greater than or equal to zero, and s+t is less than or equal to 1, the point is within the triangle. Otherwise, it is not.