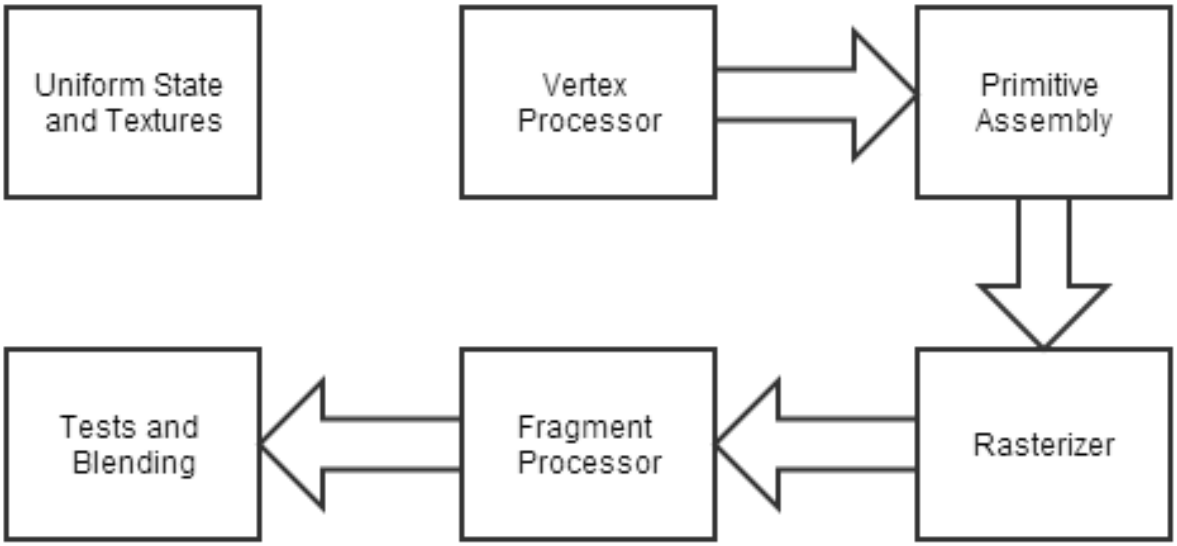
Human Graphics Pipeline documentation

Version 1 - Simple Pipeline



Stage 1 - Decide what primitive is to be drawn

Have someone begin the drawing process by setting what type of primitive to draw.

The easiest example to use is GL_TRIANGLE

Stage 2 - Provide vertices for the primitive

Have a group of people input a number of vertices, have them organised in a first in, first out way so that they can be picked up later in the pipeline in that order.

The vertices should be three dimensional, that means they should have an X,Y,Z layout. For simplicity the range of coordinates in all three directions should be limited to 1 to -1.

Stage 3 - Assemble the primitive

At this stage someone should build the primitive you want to draw from the vertices put in to the list in the previous stage. The number of vertices that are taken will depend on the primitive that is being drawn.

Stage 4 - Plotting to world space

The assembled primitive should then the taken by another person who plots the points into world space.

Once the points are plotted they should be joined up to create the primitive.

Stage 5 - Rasterizing

Here the shape should be filled. Overlay a pixel grid on the shape drawn in world space and colour in the pixels inside the shape, this is done in a purely visual way.

At this stage the shape should be a solid colour, this simulates the fragment shading stage of the pipeline where a display colour is assigned to each pixel.

Stage 6 - Buffering

The final stage is to move all the data for the completed shape out of the work space and into a storage area, this is to simulate the data produced being put in to the display buffer ready to be drawn.

Version 2 - Simple pipeline with mathematical rasterizing

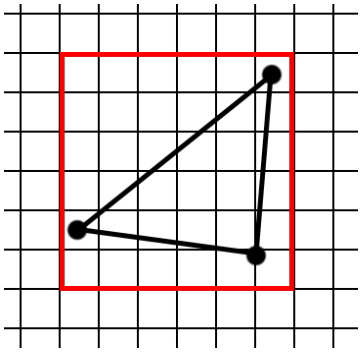
This version follows the same flow as version 1, the changes are solely in Stage 5 - Rasterizing.

Stage 5 - Mathematical Rasterizing

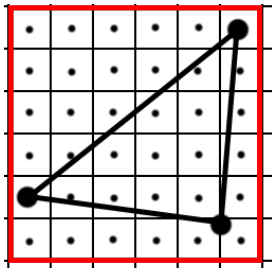
At this stage there are two mathematical methods that can be used to calculate what pixels are to be rendered, Half Space and Barycentric.

First overlay the pixel grid on the world space grid containing the plotted points and drawn primitive.

Create a bounding box around the primitive, this should be padded out to be to the nearest pixel that does not intersect the primitive. An example is below.



Place a point roughly in the center of each of the pixels within the bounding box and estimate their coordinates in world space.



You should now be ready to apply an algorithm to the shape to find out what pixels it will occupy on screen.

You need to know the three world space coordinates for the triangle and the world space coordinate for the point you wish to check, you also need to know the order the vertices were placed into the scene, these should be labeled a, b and c respectively.

Below are two algorithms, Half-space and Barycentric, they will produce a true or false value for estimating if the point is within the triangle.

Half-space

To check if the point is within the triangle you run it through an algorithm that checks its position to the left or right of the lines connecting the primitives points together.

Point check algorithm - *(2.x - 1.x) * (p.y - 1.y) - (2.y - 1.y) * (p.x - 1.x)*

Algorithm steps :-

1 - Take triangle points a and b, a is point 1, b is point 2.

2 - Get the point you wish to check for, this is known as p.

3 - Put these points through the point check algorithm, .x is the first number and .y is the second in the vertex.

4 - Record the result.

5 - Repeat for b, c and c, a. Pay attention to the order of the vertices, this has an effect on the outcome.

6 - Collect all results, if they are all positive the pixel is within the triangle, if any or all of the numbers are negative then the pixel is not within the triangle.

Barycentric

Here you are using the vector of the lines to check if the point is within the triangle.

First you must calculate the vectors, to do this the method is…

Vector 1 = (b.X - a.X, b.Y-a.Y)
Vector 2 = (c.X - a.X, c.Y-a.Y)
Point vector = (p.X - a.X, p.Y - a.Y)

These should produce three vectors with two values, one for X and one for Y.

Then you calculate two variables using these vectors.

s = (Point vector * Vector 2) / (Vector 1 * Vector 2)
t = (Vector 1 * Point vector) / (Vector 1 * Vector 2)

Once you have these two variables you can work out if the point is within the triangle.

Only if all these conditions are true will the point be considered as in the triangle.

s is greater than or equal to 0

t is greater than or equal to 0

s+t less than or equal to 1

An online calculator for both types of algorithm can be found at
http://shearer12345.github.io/humanGraphicsPipeline/Calculator.html