Graphics Pipeline Paper

The graphics pipeline is the process that creates an image on your screen. This process renders a 3D image onto a 2D screen. The four major steps of the graphics pipeline are vertex processing, clipping and primitive assembly, rasterization, and fragment processing.

First, some vertices enter the vertex processor. These vertices are processed one at a time. According to the textbook *Interactive Computer Graphics*, “The major function of this block is to carry out coordinate transformations. It can also compute a color for each vertex and change any other attributes of the vertex.” One important variable in the code that runs the vertex processor is called gl\_Position. From the online article Getting Started in WebGL, Part 1: Introduction to Shaders, the author says that gl\_Position “is where the vertex shader should write the final vertex position.” This variable is a vector with four elements that includes the values x, y, z, and w. I remember you saying that for now the w value is not important and it is normally set to 1.0 in the code.

After the vertex processor comes the clipper and primitive assembler. Because imaging systems can’t see everything, clipping must occur. However, before any part of the image is clipped off, sets of vertices are turned into primitives. These primitives can be line segments or polygons. The primitive assembler then considers a clipping volume. Any projections outside this volume will be clipped and not shown in the final image. The projections inside the volume will be shown. New vertices are created when an object is clipped.

After the clipper and primitive assembler comes the rasterizer. Also from the textbook *Interactive Computer Graphics*, it says, “The primitives that emerge from the clipper are still represented in terms of their vertices and must be converted to pixels in the framebuffer.” The rasterizer outputs a set of fragments, which represent each primitive. These fragments make up one pixel in the framebuffer, which also makes up one pixel on a screen. According to another online article on the website khronos.org, “The state for a fragment includes its position in screen-space, the sample coverage if multisampling is enabled, and a list of arbitrary data that was output from the previous vertex or geometry shader.”

After the rasterizer comes the fragment processor. This is the final step of the imaging process. The fragment processor takes in the fragments created by the rasterizer and updates the pixels in the framebuffer. This is where the final pixel color is decided for any objects in your image. This is also where light or texture is added to the image. Two ways to change the color of a pixel includes texture mapping and bump mapping. If 3D information is processed, the final image still won’t show some fragments because they are hidden behind another surface. Like the vector processor, the fragment processor has an important variable in the code that runs the fragment processor. The variable is called gl\_FragColor. Also from the online article Getting Started in WebGL, Part 1: Introduction to Shaders, the author says that it gl\_FragColor is, “where the shader writes the final value of the fragment, if there's only one color buffer attached.” This variable is also a vector with four elements that includes the values r, g, b, and a. The ‘rgb’ is the color model and the ‘a’ is alpha. I also remember you saying that for now the ‘a’ value is not important. The process is finally over and the final product is an image that you can see on any screen.