The graphics pipeline starts with the application supplying vertex data to the vertex shader. It's the vertex shader's job to take this data and to perform any transformations that are necessary to take the vertices from the application's internal coordinate system to the coordinate system of the viewer. The vertex shader is programmable so it is not simply limited to these specific tasks. At the very least the vertex shader MUST output vertices for the next step in the pipeline, which is not a big limitation. As Omar El Sayyed pointed out, it can be used for wave animations on water which would be much more computationally expensive if implemented on the cpu. After the vertex shader it is important to get rid of all of the verteces that are no longer in the field of view of the camera, and to generate the primitives from the vertex data after this is complete. These functions are fixed and are generally reffered to as the clipper and primative assembly respectively. After these fixed functions are carried out, another fixed process takes place called rasterization. Rasterization takes the primitives from the earlier processes and generates a buffer of potential pixels called fragments. Rasterization is one of the most important processes because it translates the data from the language of positions in space to the language of the final 2D display. It can be thought of as a function that takes a 3D scene and produces a projection of this scene onto the final screen. Not only that, it must also cut the image into separate small squares that will eventually become pixels. The data from the rasterizer is transported in a data structure called a pixel buffer. The pixel buffer can have a variable depth, which is a measurement of how much data each fragment contains. Possible things that this data might contain include color, pixel depth, and other data pertinent to the fragments themselves. After the rasterizer has generated the pixel buffer, processing then moves on to the fragment shader. The fragment shader's job is to take the fragments generated by the rasterizer and transform them into the final pixel color values to be displayed on the screen. The fragment shader is also a programmable part of this pipeline. Typically, a fragment shader will use information passed from earlier in the pipeline to determine which pixels will be displayed, what color those pixels will have based on vertex color information, depth information, and texture information. Anti-aliasing is typically performed at this stage. After the fragment processor is done, the image is displayed on the screen and the pipeline has done its job.