Rendering with a graphics pipeline is a method used to convert vector representations of images into actual images by using multiple steps in order to improve speed and performance of rasterization. The first step is data setup, which involves establishing a link between attribute variables and the GPU buffers. It also copies the data that shaders need for rendering into the shader program. The second step executes a vertex shader program on each vertex defined in the world coordinates and transforms them into normalized device coordinates. This process uses a transformation matrix to transform the inputs and move in front of a virtual camera. The third step is clipping, or getting rid of the extraneous data points that are not in the virtual camera's field of view. The fourth step is mapping the new data points from normalized device coordinates into actual pixels on a screen.The fifth step is rasterization, where geometric primitives are rasterized by determining which pixels are inside its boundaries.The sixth step executes a fragment shader on each of the pixels in order to determine each pixel's output color value. The seventh step is called compositing because it combines the color that fragment shader assigned each pixel with the pixel's previously assigned color and outputs it to the draw buffer. As far as I could tell, the main purpose of the pipeline method is to make the process of rendering graphics more efficient and less likely to take too long to execute. One of the sites I read about it mentioned finding a bottleneck, or a step in the process that is taking longer than the others and suggested that you reduce the workload in those areas to improve performance. It then described how to go about the process of optimization, or repeatedly locating and removing bottlenecks until the entire system is performing more or less at the same speed. It also talked about other methods to improve performance while rendering graphics, such as maximizing the sizes of the batches of primitives and reducing resource locking, as well as several tips for improving vertex processing and fragment shading.

Sources:

https://runestone.academy/runestone/books/published/learnwebgl2/01_the_big_picture/3_3d_graphics_pipeline.html

https://developer.nvidia.com/gpugems/gpugems/part-v-performance-and-practicalities/chapter-28-graphics-pipeline-performance