Application and 3d API

The application initiations the rendering operation that triggers the rendering pipeline with a pre-prepared Vertex specification. This includes the vertex array object, which holds the data to be rendered, vertex positions, drawing order and vertex colours. Along with this, is an object which provides the Vertex Shader and Fragment shader used later in the pipeline.

Vertex Shader

The Vertex Shader applies some arbitrary processing to each vertex specified in the specification and produces some output vertex, this is done in parallel on the GPU, to produce some output vertex. Aside from the vertex information provided in the vertex array object, other attributes might be assigned to use inside the shaders. These attributes are commonly used for attributes like the projection model and uniform colour values.

[Flere eksempler på matematikken og andre uniform værdier, plus deres fordele, 1 værdi mange vertex]

Primitive Setup

Primitive setup refers to the interpretation of the original stream of vertices provided, into a stream of individual "base primitives". This is done by looking at the provided interpretation scheme provided along with the vertex stream itself. These interpretation schemes (Primitives) tell OpenGL how to read the data. In a stream of GL_LINES, every other point is one end of a line and so on for points and triangles.

[Eksempel med convserion af vertices til lines, point, triangles]

Clipping and culling

Clipping and culling takes the primitives interpreted from the previous step in the pipeline, and then determines if they can be seen by the camera, thus determining if they should be rendered or not.

Clipping refers to the determination of whether or not the primitive is within or partially within the clipping volume. If a primitive has no vertices within this volume, then it is discarded or *culled*, if it is fully within the volume, it is rendered as normal, but if it is only partially within the volume, then new appropriate vertices are computed. For lines this is usually the intersection between the volume's edges and the primitive, but triangles can clip the view volume in such way that new triangles must be generated for the final rendering to look as it should.

[Back face culling]

Rasterization

Rasterrization refers to the conversion of individual primitives into discrete fragments which have a size related to the pixel size. This conversion also makes use of interpolation, as the conversion must estimate the values between vertices to generate proper fragments in between these vertices.

Fragment Shader

Similarly to the Vertex Shader, the fragment Shader applies some arbitrary processing, but to fragments instead of vertices. This also means that it is similarly parallelized and run on the GPU. The output of this shader is the depth of the fragment, its stencil value and the colour of that fragment.

Per-Sample Operations/ Raster Operations

The step in the pipeline is the last, and the output of this step is final buffer displayed on screen. Some of the operations are tests like depth tests, where a fragment can be culled based on the values stored in a depth buffer, hence making closer geometry appear in front of other geometry. Another test is the stencil test, where a bitwise operation between the fragments stencil value and a stencil value in a stencil buffer, is used to determine whether or not to cull a fragment.

Other operations are used to blend colors, or dithering which can be used to avoid banding in images.