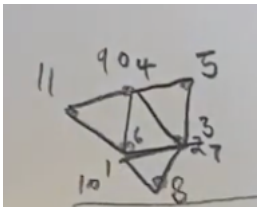
Ruikang Lin

Graphics Pipeline Explained by GetIntoGameDev
https://www.youtube.com/watch?v=ycLJ2UrKLdw

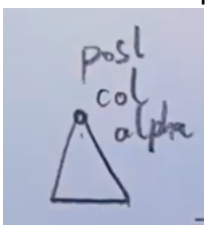
1. Vertex input
   binding description: information about the buffer that is being through at the graphics pipeline
   - binding number
   - stride
   - input rate

   It's a good idea to group everything into 1 buffer – bind number, to get consistency
   Attribute description:
   - location number
   - binding
   - format
   - offset

2. Input assembly
   We have a bunch of points, grabs a set of points, and sends it down to line.
   Make sense of the data. Points to triangles

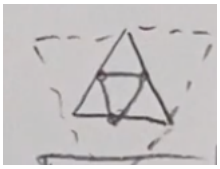
   

   Topology type
3. Vertex shader
   For vertex data in a triangle, find position, color, and alpha
   *Color is not applied until fragment shader

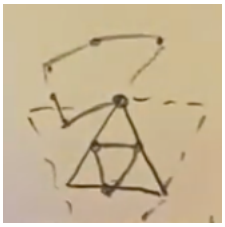
   

   shader stage:
   - stage - vertex
   - module
   - entry point – main (traditional called)
4. Tessellation
   For a triangle, subdivide it.

   
5. Geometry shader
   Reference nearby triangle to get information on the current triangle

   Could generate more vertex and pass it down the line.

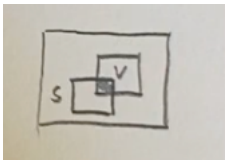


shader stage:

- stage – geometry
- module
- entry point – main (traditional called)

6. Viewport and Scissor

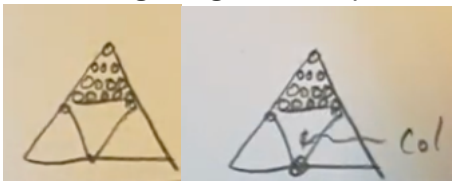
   Rasterization need these bounds in order to rasterize, clip the view region, speed things up a bit.
   Final viewable region, will be the intersection of them.



Here, only the intersection region (in black) is viewable.

7. Rasterization

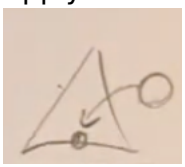
   measuring things in a shape, and interpolates (modify, insert) the attributes



   Depth test & back face is set in the rasterizer.

8. Fragment shader

   Apply the color to the fragment

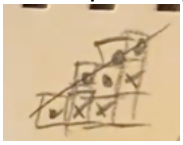


   shader stage:

   - stage – fragment
   - module
   - entry point – main (traditional called)

9. Multisampling (option)

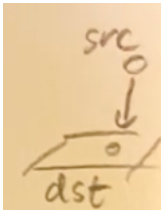
   Sets up a loop and algorithm for rasterizing.



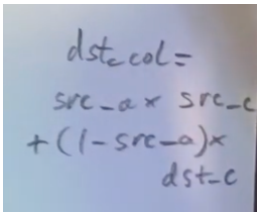
   Ex. when you draw a line, you can have a choice of pixels and depending on some sort of internal biases/parameters, you might choose a set of pixels. And in a loop, you can adjust the biases to get slightly varied positions and averages them out.

10. Color blend

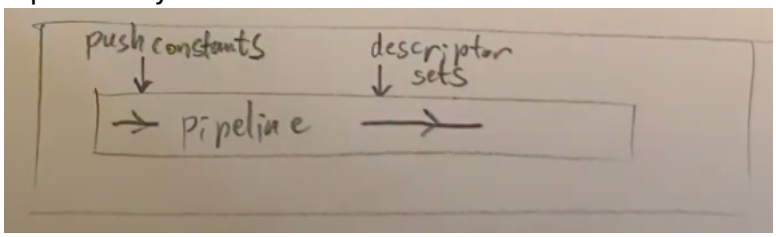
    A fragment already set as a destination (dst) and an incoming fragment (source/src),
    We have information on these fragment colors, and we can modify these colors, for example, by multiplying/adding the colors or traditional color blending.



traditional color blending:

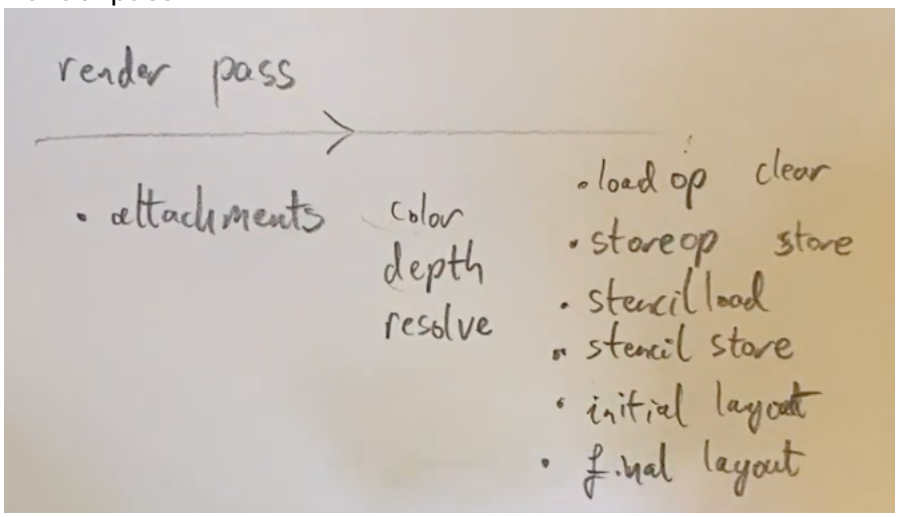


Pipeline Layout



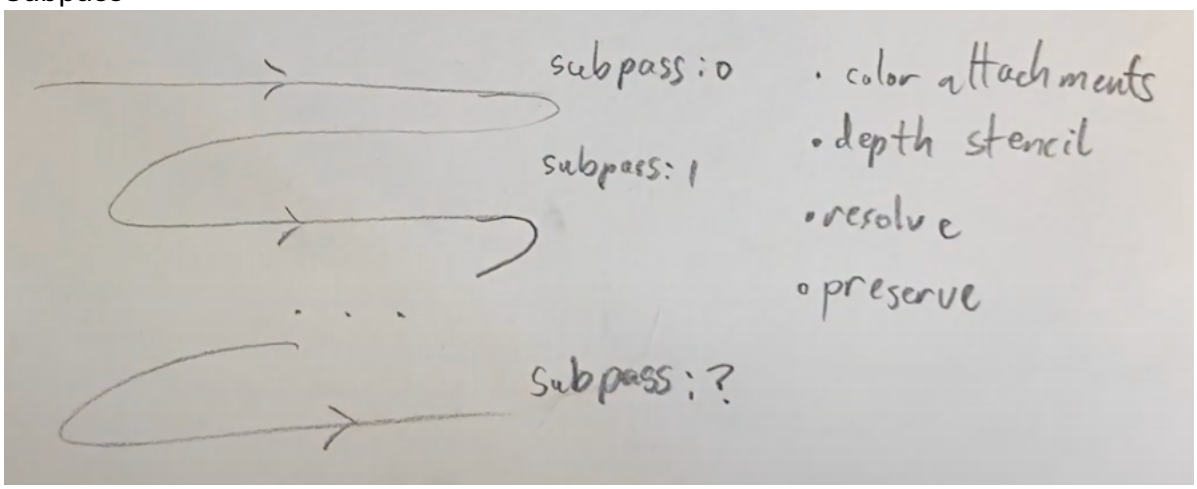
push constants: for example 1 or 2 matrices
descriptor sets: set of descriptors, a descriptor describes a resource of arbitrary type – b/c vulkan can be used in high/low level device with scientific notation

Render pass



In between each step in the graphics pipeline, we need to specify the render pass
What attachments are we working with? For example: color, depth, resolve (when a multisampled buffer sample turns into a single sample buffer)
Each attachments has operations: load op, store op, stencil load, stencil store, initial layout, final layout

Subpass



Think of running the pipeline, the shader again.
Use case: deferred shading
We always need at least 1 subpass
Attachments: color, depth, resolve, preserve