1. A pipeline can speed up processing by getting provided data to be processed by the pipeline. It helps set things up, then the pipeline can start executing in its order. The speed comes from Hardware design, software considerations, and geometry restrictions. Rendering also helps enable safe parallel processing which allows different parts of images to be processed independently. The processing steps can be divided among the processes which allows them to overlap further.
2. Data Flow
   1. Vertex shader can receive buffer data
   2. Fragment Shader and Vertex shader can receive uniform buffer data
   3. in layout(location = 5) vec3 bNormal;
   4. In Vertex shader -> out vec2 vTxtCrd;
   5. In Fragment Shader -> in vec2 vTxtCrd;
   6. In fragment shader -> in vec4 frag\_color;
   7. uniform layout(location = 6) float specular;
3. Stacked Triangles
   1. If it isn’t enabled, then a fragment processed after another fragment lands on the same pixel will replace the pixel’s color. Painter’s algorithm says that the last rendered pixel will take the color. The first triangle that would cover all others would be the black triangle. The triangle behind the black one but would cover the other two would be the orange triangle. The third triangle behind the first two but covering the third would be the blue triangle. The gray triangle would be covered by all other triangles.
      1. Black
      2. Orange
      3. Blue
      4. Gray
   2. Depth testing would make it so that the fragment closest to the camera gets its color shown. That would mean that certain triangles would be displayed first based on its distance from the camera. So, the order in which its color is shown/overlapped is as follows.
      1. Black
      2. Gray
      3. Orange
      4. Blue

auto const A = 27;

auto offsetA = 0;

auto colorA = orange;

auto const B = 105;

auto offsetB = A + offsetA;

auto colorB = + blue;

auto const C = 90;

auto offsetC = B + offsetB;

auto colorC = green;

glUniform4fv(color\_loc, 1, colorA.as\_float().get());

glDrawArrays(GL\_TRIANGLES, offsetA, A);

glUniform4fv(color\_loc, 1, colorB.as\_float().get());

glDrawArrays(GL\_TRIANGLES, offsetB, B);

glUniform4fv(color\_loc, 1, colorC.as\_float().get());

glDrawArrays(GL\_TRIANGLES, offsetC.as\_float().get());

1. Data Memory
   1. 3
   2. 651
   3. 7812
   4. 2604
2. Fragment Collision
   1. Fragment colors can be blended in certain circumstances. If there is an opaque surface, you can blend the colors together making one seem translucent.
   2. Two final pixel color strategies
      1. Painter’s Algorithm – The last fragment that was processed uses its color for the pixel.
      2. Depth Testing – The fragment closest to the camera gets its color shown in the end.
3. Computing Colors
   1. Red
   2. It is not getting the gold value with the given vec4 variable in a) because it is applying integer division and as a result, getting a value of 0 for everything but the first one. 255/255 will yield 1 but something like 215/255 will yield 0 according to the CLion program.
   3. Adding a .0 at the end of everything will get you the correct value. So, it could be vec4(255.0/255.0, 215.0/255.0, 0.0/255.0, 1.0);
4. Buffer Parameters
   1. glBufferData(GL\_ARRAY\_BUFFER, 2\*83, vector\_field, GL\_STATIC\_DRAW);
   2. glVertexAttribPointer(bVectorField, 2, 2\*GL\_BYTE, GL\_TRUE, 0, nullptr);
5. Order of transformations
   1. Order from buffer
      1. World scale
      2. World rotation
      3. World translation
      4. Animation scale
      5. Animation rotation
      6. Animation translation
   2. **Ignore for now**