CG Programming - S0006E - 2018 Week 7

Software Rasterization

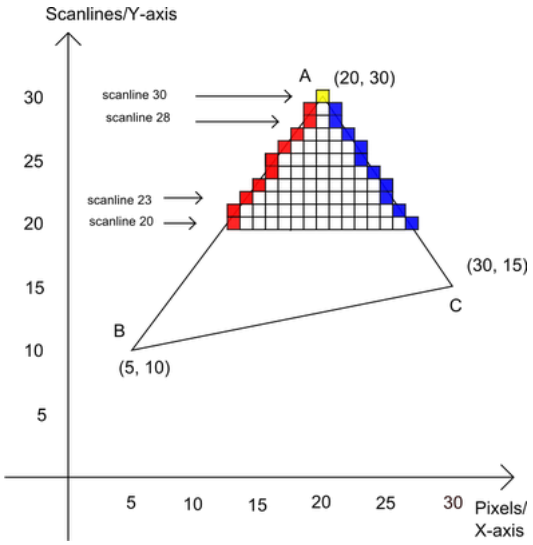
The goal is to create your own software rasterizer that can draw textured and lit triangles into an image buffer. You should upload your image buffer as a texture in OpenGL and render it to a quad.

1. Create a software renderer class. This should contain functions for:

   - Adding a vertex buffer and index buffer.
     This should return a handle that can be used with the draw function.
   - Setting up a framebuffer with arbitrary dimensions.
   - Retrieving a pointer to the framebuffer.
   - Retrieving the size of the framebuffer.
   - Setting a vertex shader by providing a lambda function as an argument.
   - Setting a pixel shader by providing a lambda function as an argument.
   - Setting a model-view-projection matrix to transform the triangles with.
   - Setting a texture resource to texture the mesh with.
   - Rasterizing a triangle.
   - Drawing an index buffer to the framebuffer by passing the buffer handle as an argument.
     This calls the *RasterizeTriangle* for each triangle in the index buffer.
     This should use the vertex and pixel shaders that has previously been set up.

2. The *RasterizeTriangle* function should take 3 vertices with (at least) 3D position, normal and texture coordinates as argument. It should interpolate the normals and texture coordinates smoothly over the surface of the triangle inbetween the vertex and pixel shader (barycentric coordinates).

   Implement the triangle rasterizer by using a scanline technique. Start at the vertex with the lowest y value and proceed line by line downward (assuming [0,0] is top left corner of the image buffer). Use the Bresenham algorithm for finding the left and right pixels on every line and linearly fill the line.



   The *RasterizeTriangle* function call the vertex shader for each vertex and pixel shader for each pixel, storing the resulting color in the framebuffer.

3. The vertex shader lambda function should take a single vertex with (at least) 3D position, normal and texture coordinates as argument. The vertex shader should transform the vertex attributes using the model-view-projection matrix. It should return any necessary data that needs to be passed along to the pixel shader.

4. The pixel shader lambda function should take uv coordinates, normals and a texture as argument and should return a pixel color. Shade the pixel using Blinn-phong shading and texture it using your uv coordinates and texture.

5. The frame buffer should have a accompanying depth buffer that can be used to perform depth tests to discard pixels early.

6. Add clipping

   - The *RasterizeTriangle* function should discard any pixels "outside" the view frustum.
   - (Optional) The draw function should discard any triangles that are entirely outside the view frustum.

7. Add functionality so that you can move and rotate the object with mouse and keyboard, like in the previous assignment.

8. Render the resulting framebuffer/image onto a quad as a texture using OpenGL.

Delivery

Present your completed assignment to Fredrik or Johannes. Place your completed assignment in the corresponding folder in your repository, commit, push and then upload the number of the revision to canvas rooms submission folder. *Remember to comment your code thoroughly.*

Deadline 2018-11-02 12:00