Lab 3

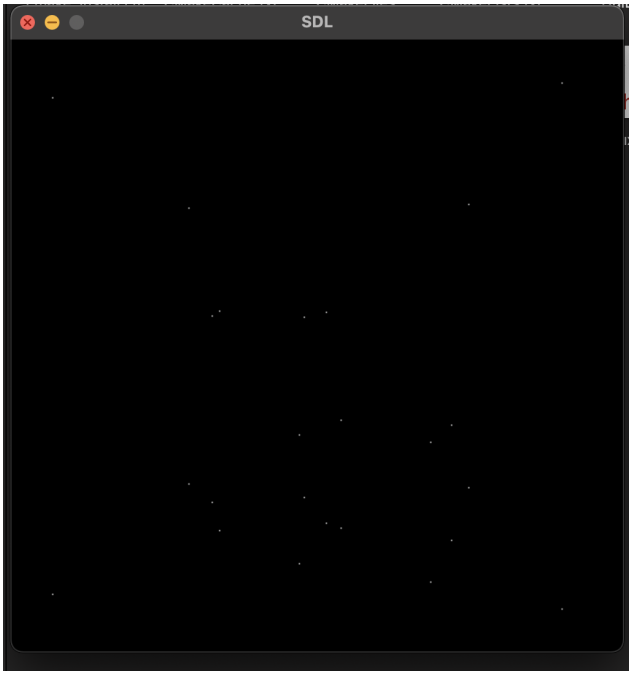
DH2323 DGI22

Josephine Kvarnberg
jkv@kth.se

13 May 2022

1. Drawing Points

The scene is 2 lengths in each direction, ranging from -1 to 1. If the camera is placed 3 steps in the negative z direction, there will be 2 lengths between the start of the scene and the camera. Meaning the scene will definitely fit into the view of the camera, covering the whole screen. If the aspect ratio is 1:1 then both horizontal and vertical fov will be 90 degrees.



Drawing of points of polygons.

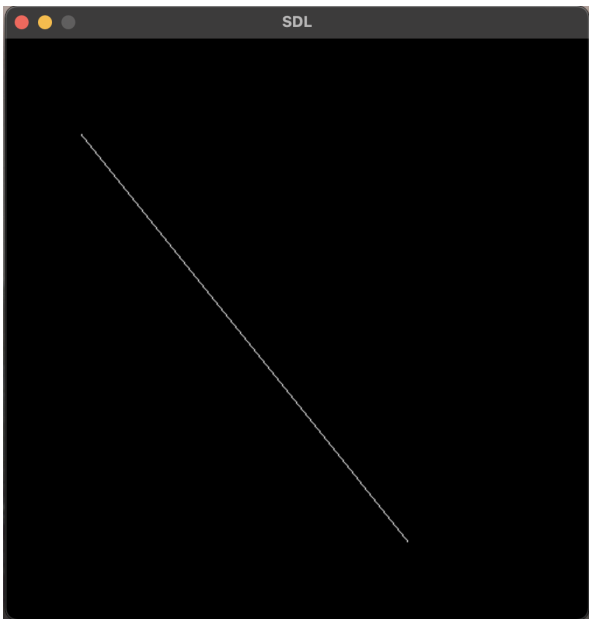
I experienced some issues with the executable not showing the scene without me rotating it first. It was hard to solve at first but then I tried changing the initialization of rotation matrix R to include (1 0 0, 0 1 0, 0 0 1) instead of being empty. That solved it.

2. Drawing Edges

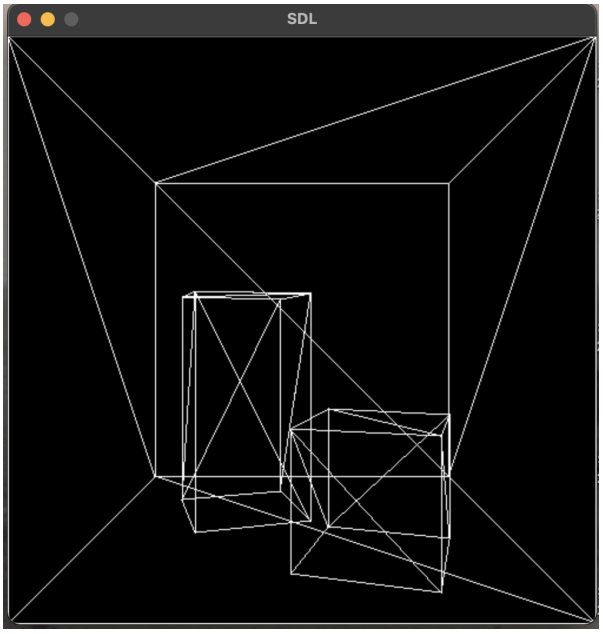
Some common functions that I found important to have the description for are these two.

glm::max - Returns y if x < y; otherwise, it returns x.
glm::abs - Returns x if x >= 0; otherwise, it returns -x.

DrawLineSDL was tested with some random pixels on the screen and it worked to output a simple 2D line.



Test with DrawLineSDL



Rendered edges for polygons.

I added the interaction code used in lab2, for rotation and moving back and forth. I notice that if i rotate so that the back of the camera faces the scene, different lines are appearing. According to instructions this is expected and okay.

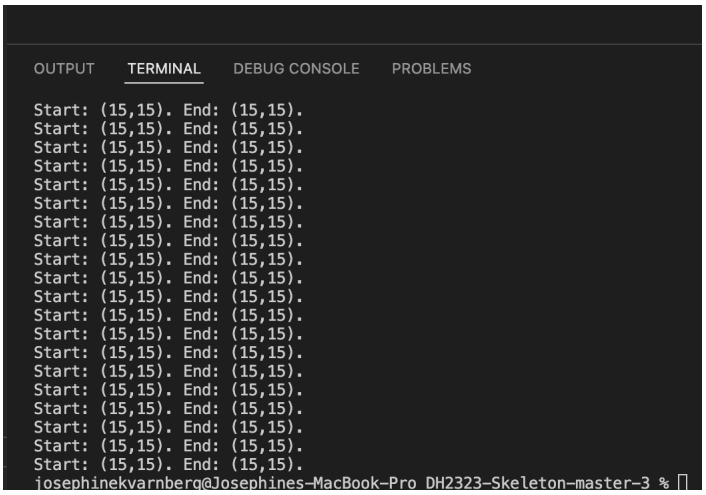
Despite that, I could not for the life of me get GLM::pi() to work so I ended up just inserting a constant pi=3.14159265359 for the rotation. It worked fine.

3. Filled Triangles

I implemented ComputePolygonRows according to the instructions, which were very descriptive in what was needed code-wise. When I had to find a max of three y-values. I used std::max and std::min

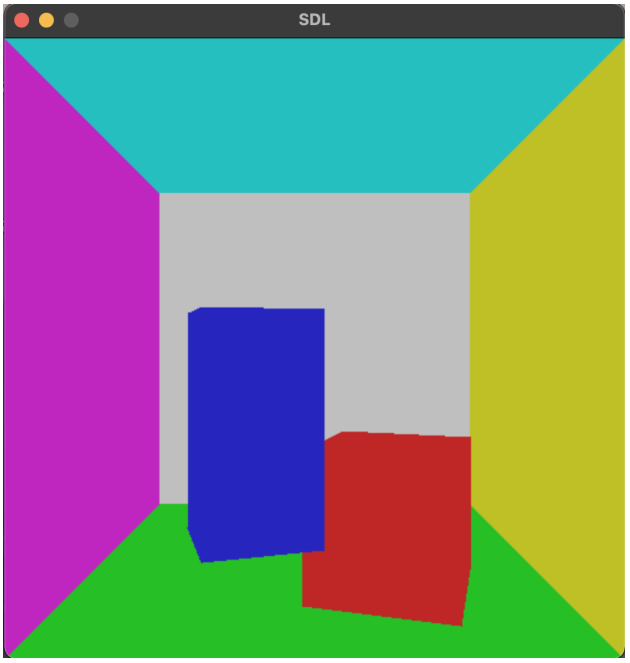
instead of GLM::max and GLM::min, since that can take a list of values instead of only two arguments.

Finding the number of rows is taking the maximum y-value minus the minimum y-value, and adding 1. I had trouble testing the function and it output the following instead of what was described as expected in the instructions.



Error in testing the function.

It turned out, after having tried changing most parts of the code, that I initialized the minimum y-value as a max value instead of min. A small error that got me stuck for a very long time.

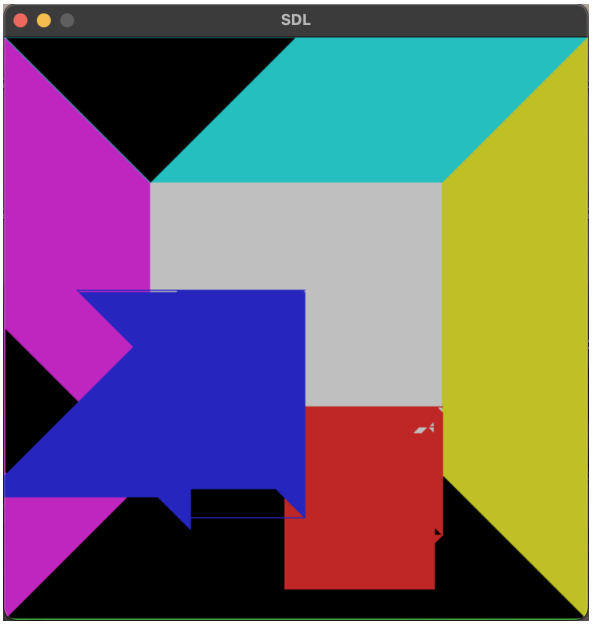


Rendering of the scene where depth is not taken into account.

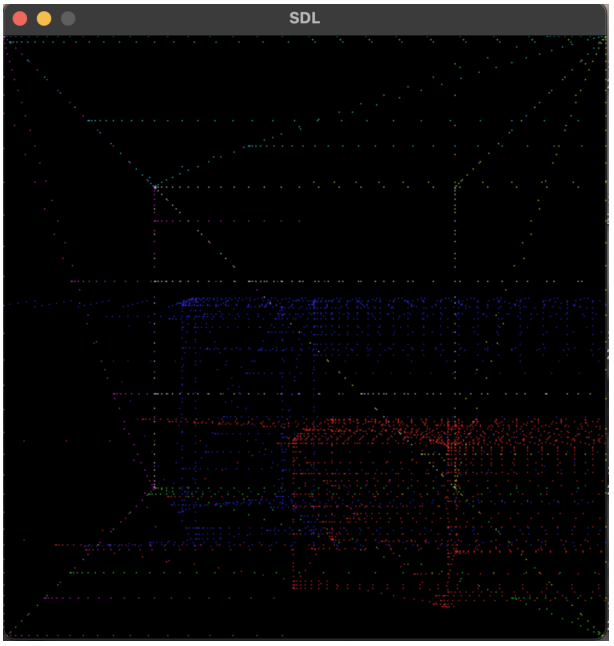
The speed of the program was compared with the speed of the raytracer, in the current stages of implementation (without illumination). Rasterization was done at about 16-17 ms. Raytracing (without DirectLight) was around 135-141 ms but it did vary a bit. These numbers are expected, since rasterization is supposed to be much faster than raytracing.

4. Depth Buffer

Changing every ivec2 to Pixel meant that some code had to be altered to fit the struct. However, something broke by doing this part and I could only render the dots from the beginning of the lab. By further inspection, I found that the first version of the pixel interpolation had errors in that I incorrectly assigned some values. When changing that to what I thought would work, the following scene was rendered. It looks pretty interesting.



Quite abstract.



Somewhat visible shapes.

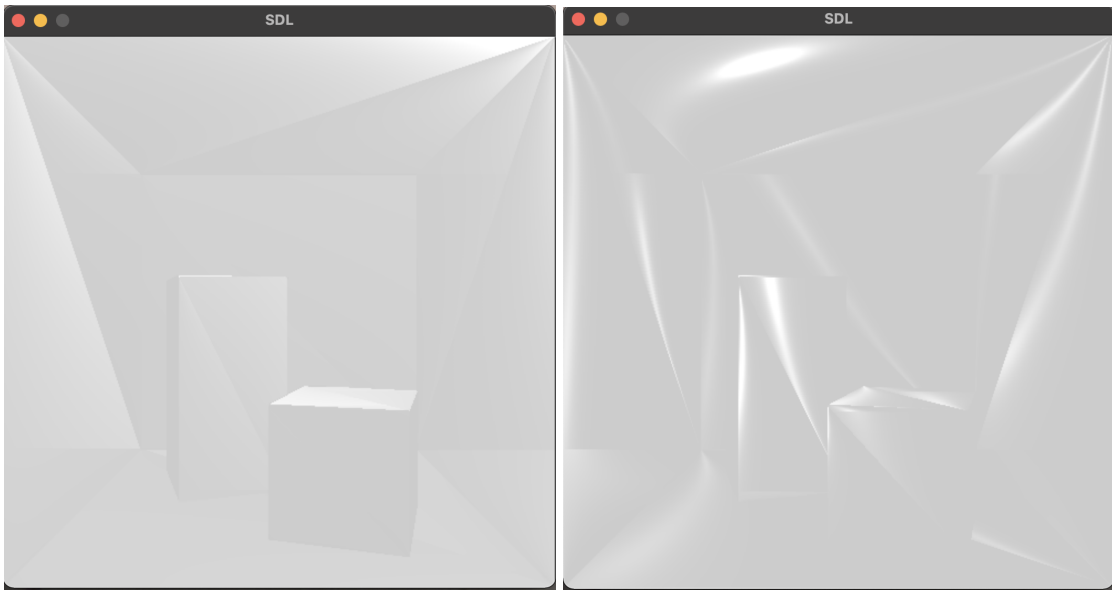
On another try, I got better results in that the shapes are somewhat visible as they should be. This was solved by fixing the interpolation function, as it was incorrect. I incorrectly handled the current pixel values as in the interpolation function from before. Instead, I changed it to the way that color

interpolation works in lab1, which is the formula of (1-t)a + t*b. In short, it works like startValue + fraction * (endValue - startValue); where fraction is a value between 0 and 1.

Scene with squares rendered in the right order.

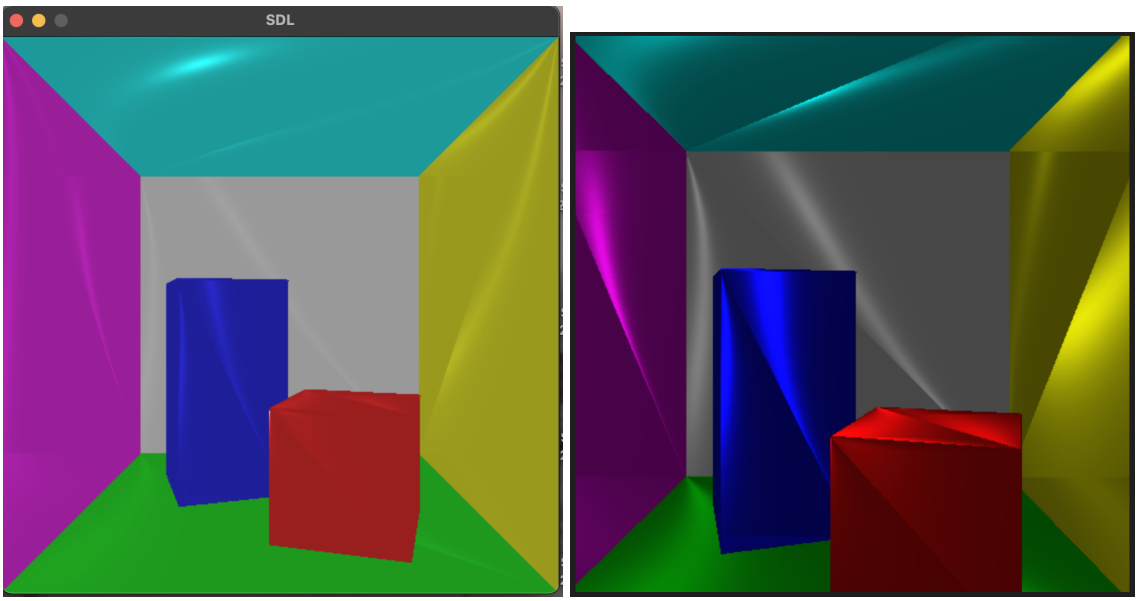
I then worked with including the depth buffer to compare zinv (depth value) to it so that the right pixels gets drawn occluded (or not occluded).

5. Illumination



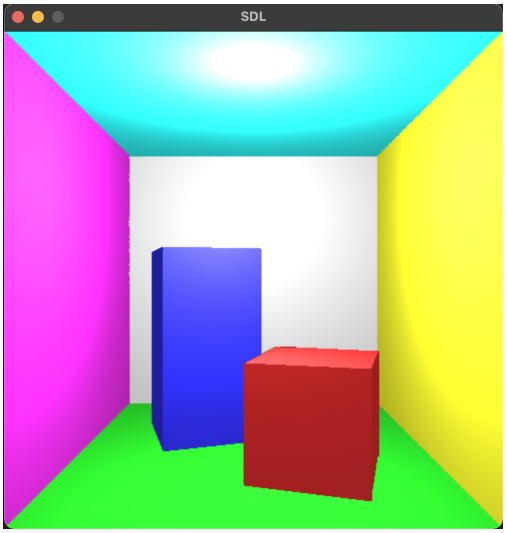
Interesting render where color has been left out and only wrong illumination is seen.

I had a very hard time implementing PixelShader and VertexShader to match the expected outcome. When I got a rendered image, it was not illuminated or looked like the above images. When I moved the camera it failed due to a segmentation fault or bus problem.



Wrongly interpolated scenes with illumination of different powers.

After changing around the code, it is getting better rendering wise, but the illumination is still not okay. To combat the segmentation fault, I tested a few different ways of checking the bounds. I had to ask around for tips on this. I ended up putting in a check in DrawLineSDL, before calling PixelShader. The lightPower variable was turned up since it was too weak to see any clear illuminations. In the right image, the scene gets illuminated incorrectly since the interpolation is not done correctly. After further discussions with a classmate on what I am doing wrong, I revised the interpolation function to get the correct illumination. Movement of the light source was copied from lab2 and worked as before.



Final scene.

References

- https://stackoverflow.com/questions/13488957/interpolate-from-one-color-to-another
- https://en.wikipedia.org/wiki/Linear_interpolation

- https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/visibility-problem-depth-buffer-depth-interpolation