Set-up and Introduction to 2D and 3D Graphics

Author: Oliver Eriksson
KTH-id: olieri
Email: olieri@kth.se
Repository: https://github.com/raaavioli/DH2323-Computer-Graphics

I was not able to make SDL work on my hardware, so I used SDL2 together with the following skeleton linked in canvas: https://github.com/lemonad/DH2323-Skeleton

Dependencies

- SDL2
- CMake

Build

- git clone https://github.com/raaavioli/DH2323-Computer-Graphics
- cd DH2323-Computer-graphics-and-interaction/Lab-3-Rasterization
- mkdir build && cd build
- cmake ..
- make

Notice that CMakeLists are configured to compile for arm64 if running Apple (MacOS), since I'm working on Apple Silicone.

So if you're running **MacOS** on an **x86** processor and you're having problems, **comment out** the following lines in CMakeLists:

```
IF(APPLE)
  SET(CMAKE_OSX_ARCHITECTURES "arm64" CACHE STRING "Build architectures for Mac OS X" FORCE)
ENDIF(APPLE)
```

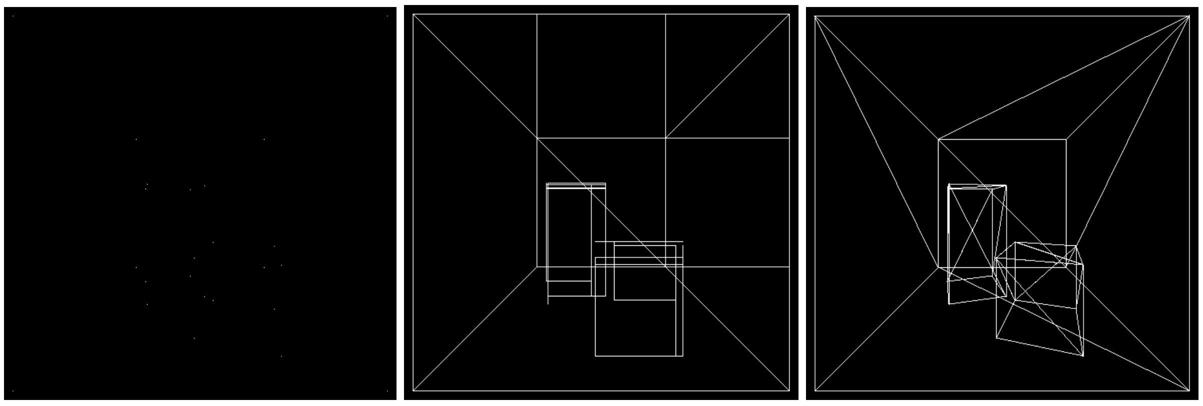
Run

Lab on rasterization is compiled into *Rasterization*

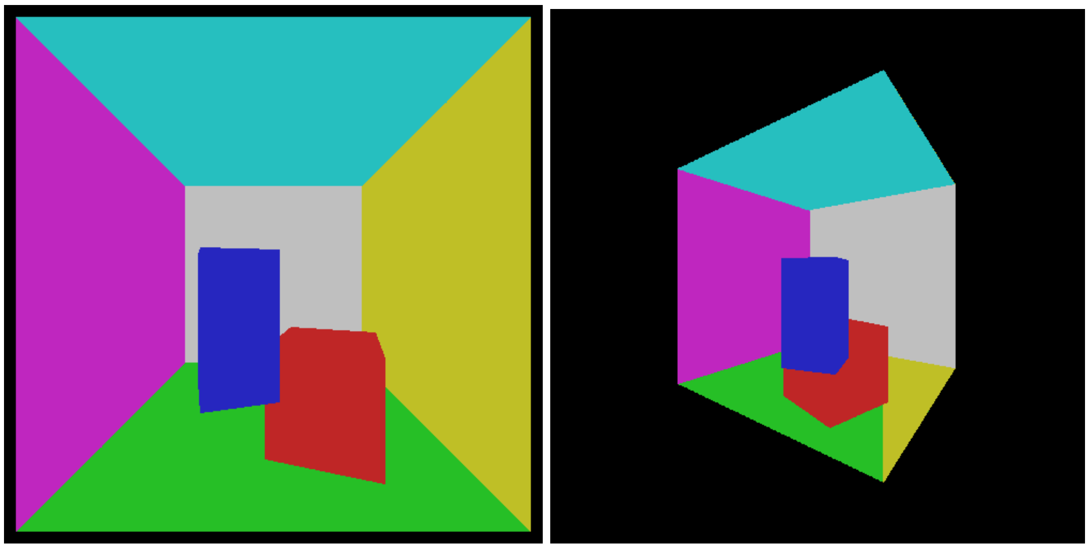
- *./Rasterization*

Implementation - Rasterization

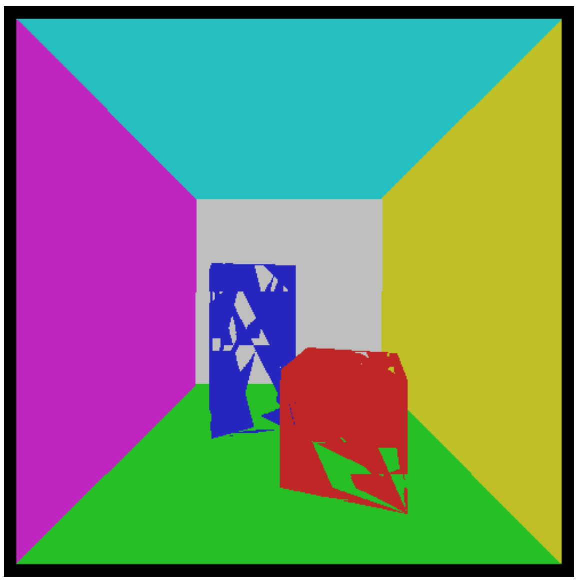
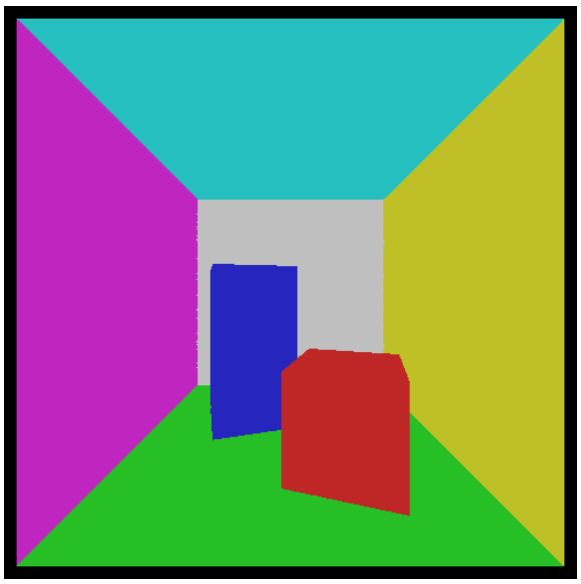
When implementing the rasterization assignment, I did not have any problems putting together the first part displaying the vertices of the model. However when trying to link the vertices into lines, there initially was some problem with which vertices were linked. Edges were only drawn parallel or perfectly diagonal at a 45 degree angle. The problem was due to rounding error when interpolating. I had mistakenly used ivec2 instead of vec2 for the step size so the smaller of dx and dy would remain 0 when divided by "N - 1", making the step size in x and y either 1, 0 or -1.


Left: plotted projected points of model scene, Middle: Plotted model with interpolated lines with broken interpolation function, Right: Properly interpolated lines for model scene.

The next major problem was mainly due to me not reading the instructions properly and not testing my Interpolation function properly and relying on the course instructions to be correct. This caused my application to crash due to a segmentation fault. The seg-fault occurred rather non-deterministically and was pretty hard to understand. After a while of debugging, I found that the error was due to a rounding error in my interpolation function. Rather than rounding to the closest value, and thus to the closest pixel, the value was floored to the closest smaller integer, which caused out of bounds reads in the left_pixels array. Lesson: Don't trust other people's implementations, build things yourself.

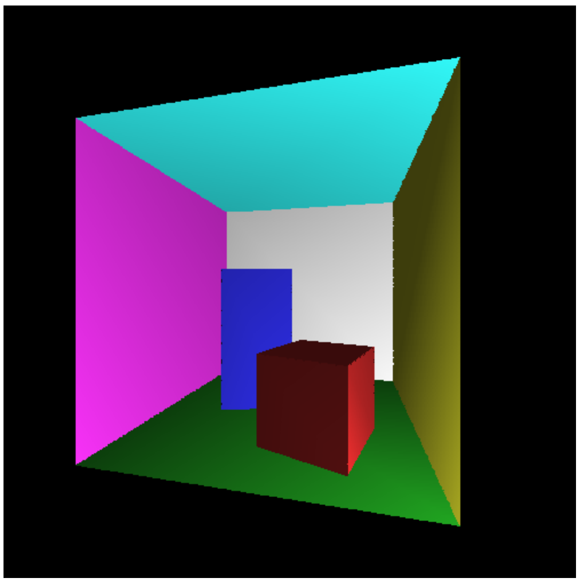
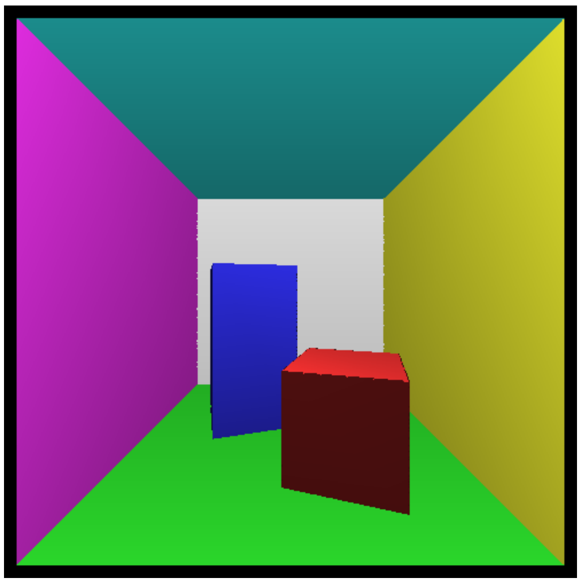

Left: Rasterized model scene without depth test, Right: Same as left, except viewed from an angle to the right

Implementing the depth buffer finally went rather smoothly. I had no real issues, however I had to rewrite the DrawRows function. Previously I had not used the Interpolation function, but rather just looped through all x-values between left_pixels.x and right_pixels.x for each row. Now that I implemented the new Pixel-struct with its corresponding Interpolation implementation, I could use that to get both the x-values and the z-values interpolated simultaneously. A small miss when creating the pixel-interpolation function, where I accidently rounded (1-p) from the formula (1 - p) * a + p * b, when interpolating, caused a graphical bug illustrated below.

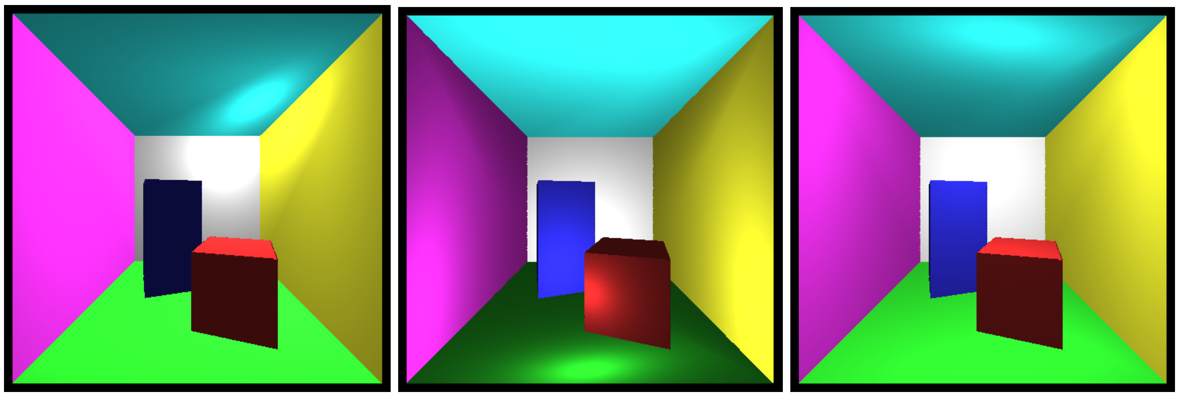
Left: Scene with graphical depth buffer bug caused by rounding error when interpolating z-values. Right: Proper scene rendered with rasterization using depth buffering.

Implementing diffuse per vertex shading was no problem as I have done that before in rasterized scenes. The following images illustrate diffuse shading with light sources positioned at different positions in the scene.

Left: Light position in bottom/right/near corner. We can observe the right side of the red cube being illuminated due to all vertices facing the light source, while the front facing side is dark. The yellow right wall is interpolated due to the position of the light source. Right: Light positioned as suggested in the assignment.

Finally, the last part was about implementing the per pixel illumination, which also went without any issues. For the result to look interesting, I tried to tweak the light position and intensity values. I also found that squaring the cosine-value, calculated from the dot-product between the normal and the light direction, created a more drastic and interesting looking shade.



Left: Light positioned in the upper/right/far corner with high intensity. Middle: Light positioned close to left/near edge of red box with medium intensity. Right: Suggested settings, light positioned above middle vertically with rather low intensity.