DH2323 Computer Graphics and Interaction

Lab 3: Rasterization

Ramona Häuselmann

ramonaha@kth.se

May 18, 2021

1 Drawing Points

I used the given skeleton for the `Draw` function and implemented the `VertexShader` by using equations 3, 4 and 8 from the assignment instructions. This resulted in the following image. We can see the vertex positions of the scene projected on the image as white pixels. It looks the same as in the instructions.

Figure 1: Drawing Points

2 Drawing edges

I implemented the `Interpolate` function as given in the instructions and use it to draw a line as follows

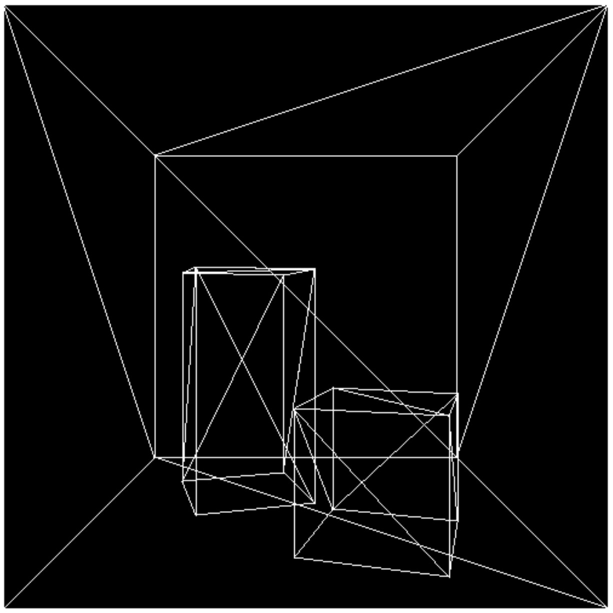
```
void DrawLineSDL(SDL_Surface* surface, glm::ivec2 a, glm::ivec2 b, glm::vec3 color) {
    glm::ivec2 delta = glm::abs(a - b);
    int pixels = glm::max(delta.x, delta.y) + 1;
    std::vector<glm::ivec2> line(pixels);
    Interpolate(a, b, line);
    for (int i = 0; i < line.size(); ++i) {
        PutPixelSDL(surface, line[i].x, line[i].y, color);
    }
}
```

Then I use the `DrawPolygonEdges` and `Draw` functions given in the instructions to create the image shown in Figure 2 a). I mapped W, A, S, D, Q, E to move the camera as follows:

- W: move camera forwards
- S: move camera backwards
- D: rotate camera around y axis to the left (yaw)
- A: rotate camera around y axis to the right (yaw)
- Q: rotate camera around x axis down (pitch)

- E: rotate camera around x axis up (pitch)

I also implemented camera movement via mouse but I deactivated it for the rest of the assignment since I found it a bit annoying.


(a) Drawing Points

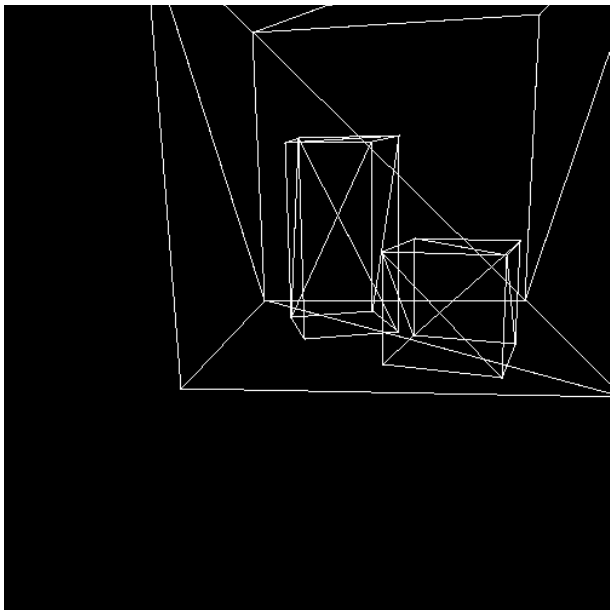

(b) Camera movement

Figure 2: Drawing Edges

3 Filled Triangles

I started with implementing `ComputePolygonRows` by implementing the given instructions in the comments. To test my implementation I wrote `TestComputePolygonRows` to compare my output with the output given in the instructions. While implementing this I had to debug some index out of bounds problems while accessing my line vector but I was able to solve this after some debugging. After I got that working I implemented the `DrawPolygonRows` function and finally used everything to draw a the scene by using `DrawPolygon` and `Draw` as given in the instructions. This resulted in the image shown in Figure 3, which looks the same as in the instructions. The blue box is overlapping the red box.

The rasterized scene in 500x500 pixel takes 5ms to render. The same scene raytraced takes 110ms. We can see that rasterizing is much faster than ray tracing.

4 Depth Buffer

I started by implementing an `Interpolate` function for `Pixels` and calculate the `zinv` value in the `VertexShader`. The `zinv` is interpolated over the triangle. In my `DrawLineSDL` function I then check the depth value of the pixel before drawing. This results in the image shown in Figure 4. Now the blue box does not overlap the red box anymore.

5 Illumination

I started by implementing and adjusting `PixelShader` and `VertexShader` as given in the instructions. After adding the calculations described in the instructions I get the result in Figure 5. Here we calculate the illumination per vertex and interpolate the illumination over the triangles. Figure 6 shows the result

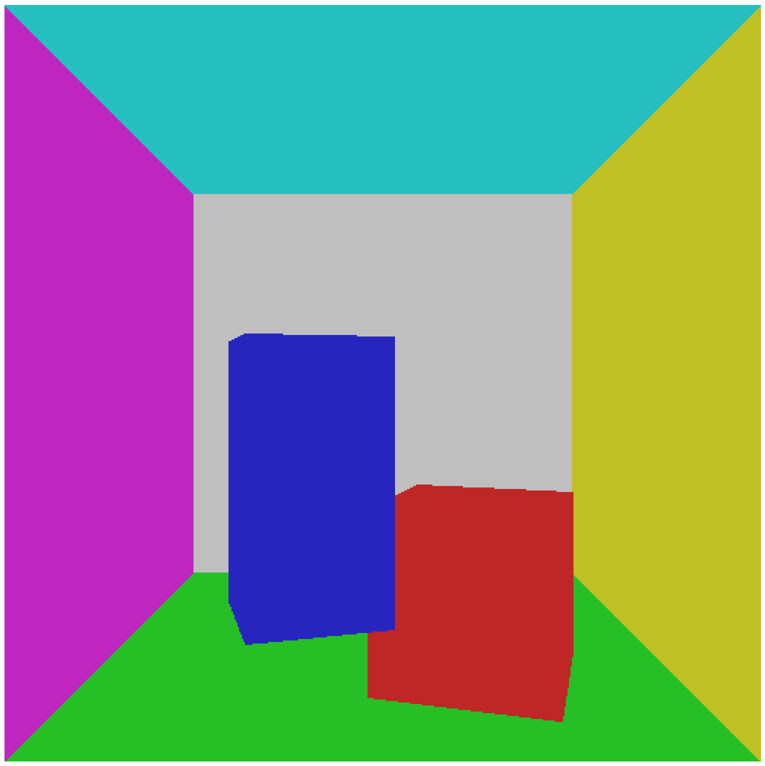


Figure 3: Filled Triangles

after adjusting the code to use per pixel illumination. Here we interpolate the 3D position over the triangle and use this interpolated position to calculate the illumination of each pixel. In Figure 6 the illumination looks skewy since we are not using perspective correct interpolation. By interpolating over p/z instead of 1/z we obtain the correct looking illumination shown in Figure 7. Finally I added light movement with the following mapping:

- UP: move light forwards
- DOWN: move light backwards
- LEFT: move light to the left
- RIGHT: move light to the right
- RIGHT_SHIFT: move light up
- RIGHT_CTRL: move light down

Figure 8 shows the scene with a different light position.

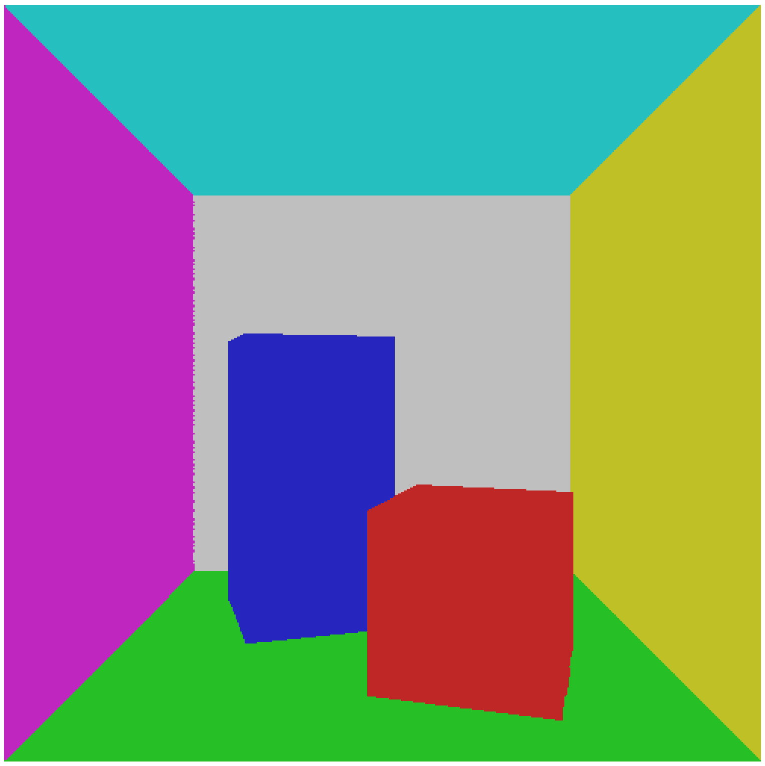


Figure 4: Depth Buffer

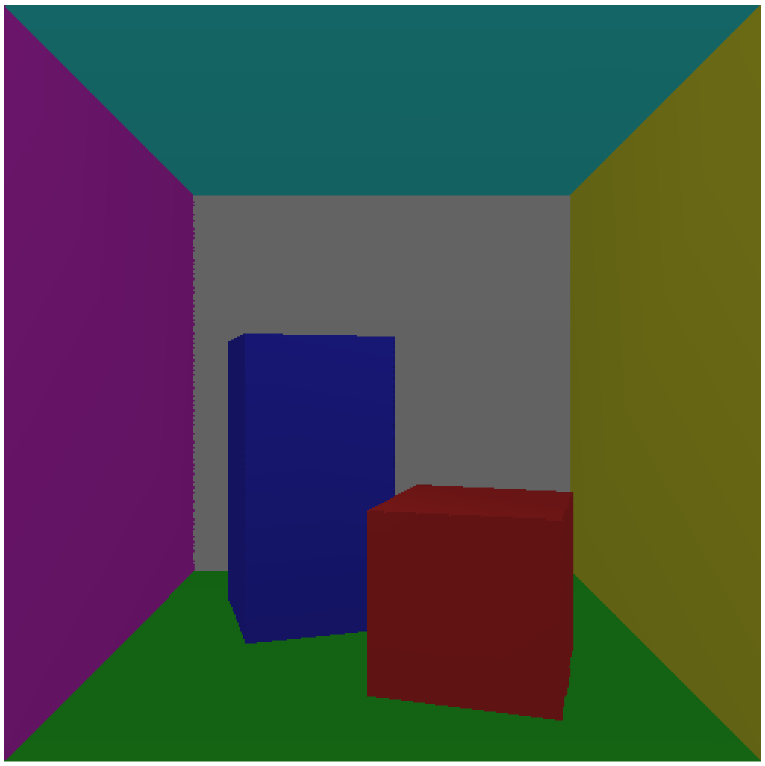


Figure 5: Per vertex illumination

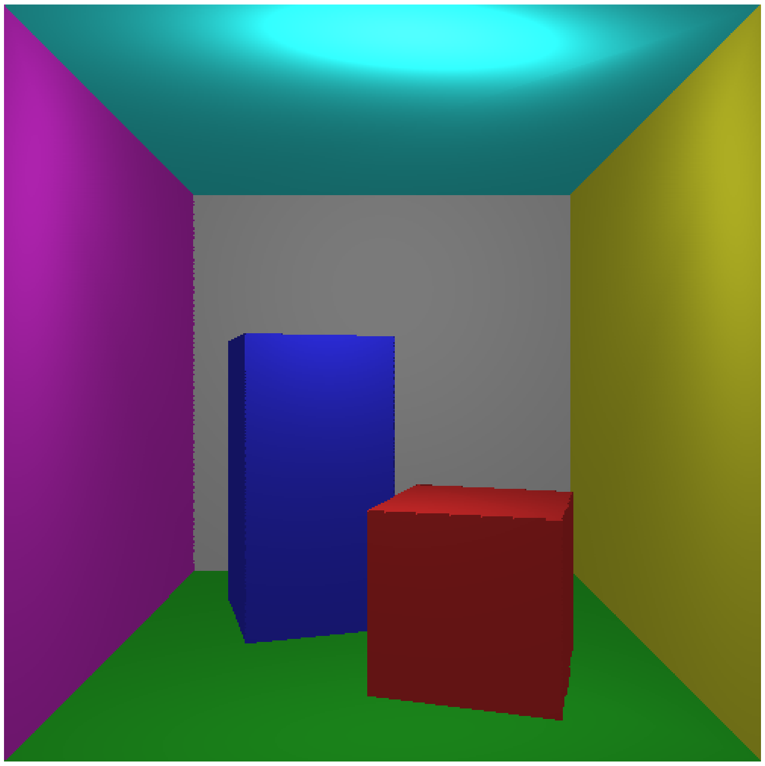


Figure 6: Per pixel illumination (skewy)

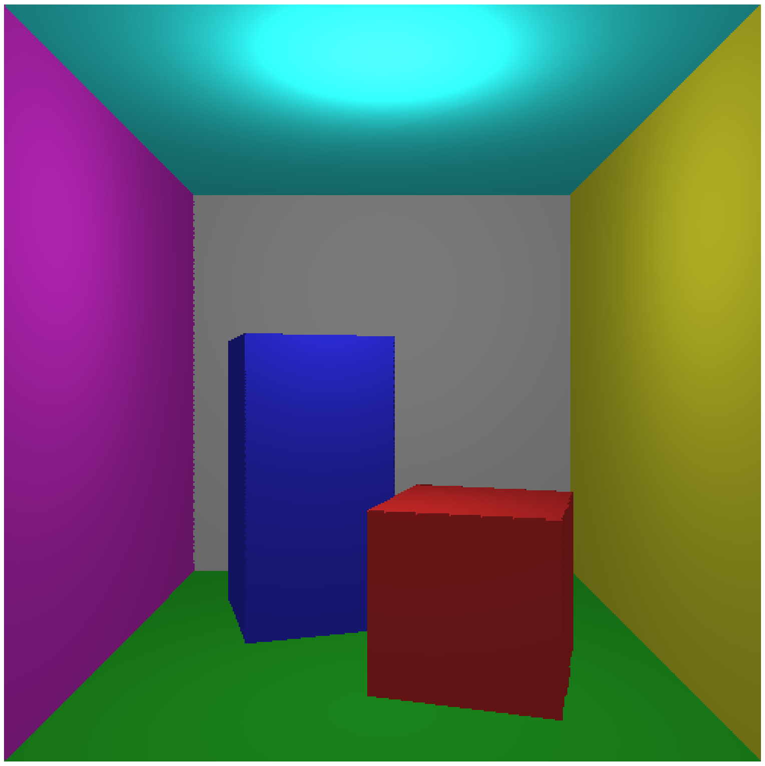


Figure 7: Per pixel illumination (not skewy)

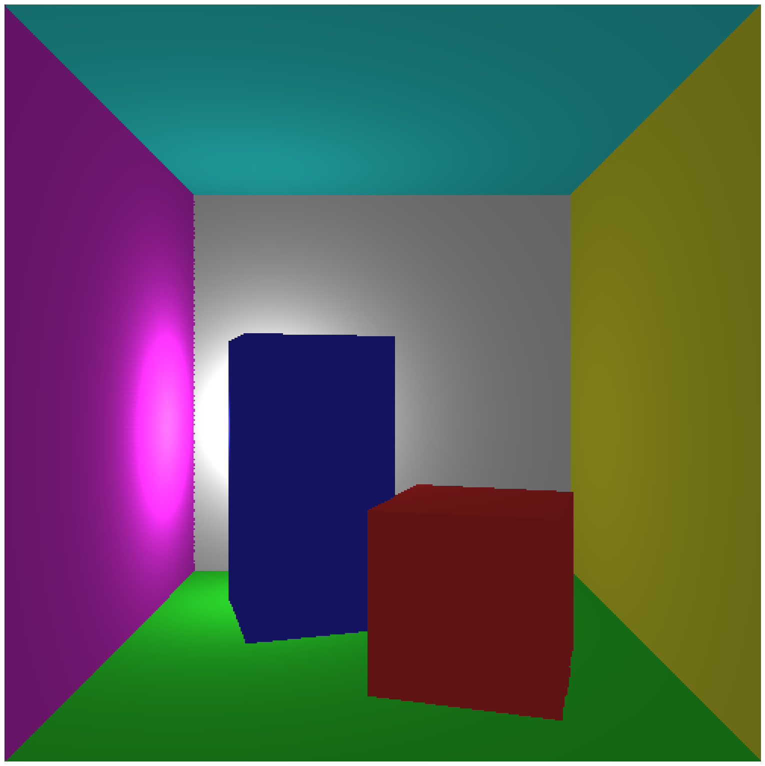


Figure 8: Move light