Raytracing

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-2-Raytracing
- 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)
```

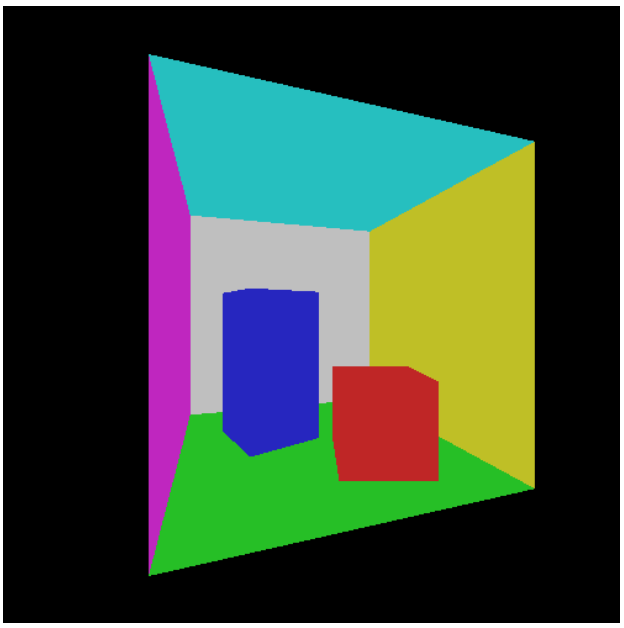
Run

Lab 2 is compiled into *Raytracing*

- *./Raytracing*

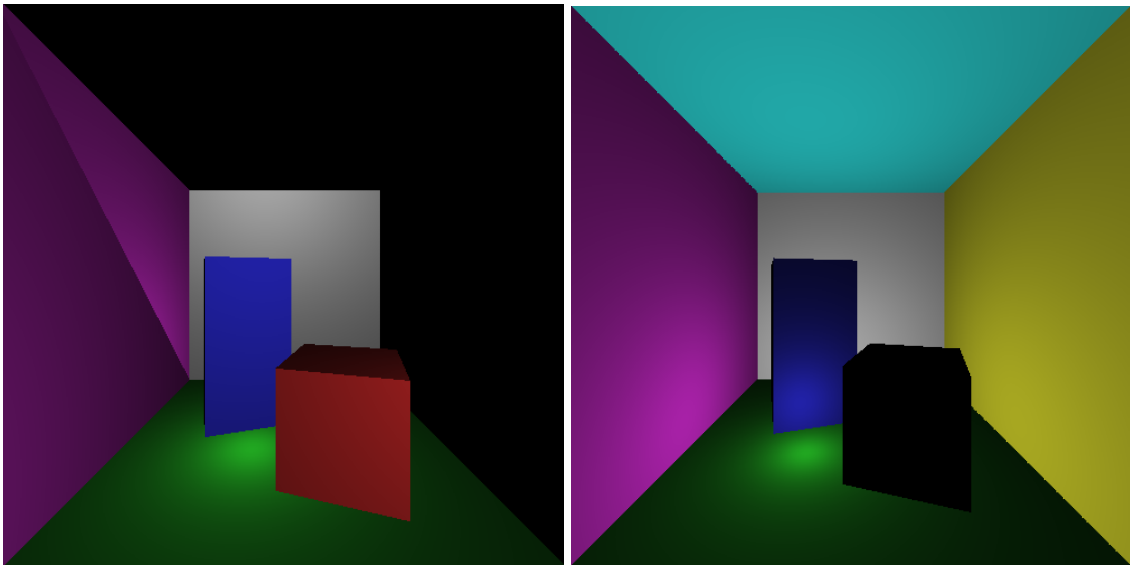
Implementation - Raytracing

The first part of the assignment went smoothly without any issues. I have implemented similar ray tracing scenes before, and have built scenes with cameras several times, so I did not encounter any particular bugs on part 3 and 4. The resulting scene looks something like the following image.



The camera is positioned a little further back and rotated slightly to the right.

When setting up the illumination, the first few visual artefacts were shown. I happened to have miscalculated the position of the intersection in previous stages writing “position = u * e1 + v * e2” without adding v0. This bug did not show up until I enabled illumination, as the position of intersection is first used when calculating the light direction. The first result looked something like the following image.



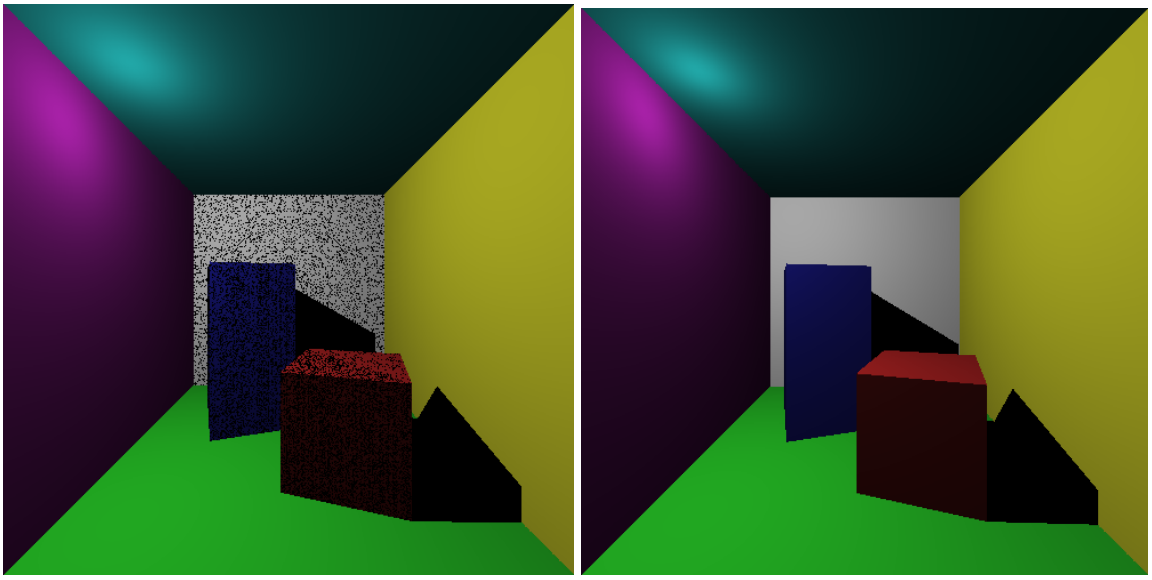
Left: *Light source positioned close to the floor between the two boxes. Neither the roof nor the right wall is illuminated with diffuse lighting. The light on the left wall is also not properly interpolated. Right: Proper look when correct position was set*

Next major graphical bug encountered was when implementing shadows. As we know, floating point numbers do not have infinite precision, and this is likely to cause graphical bugs if not considered properly. When calculating shadow rays, a common problem is self-intersection. Self-intersection basically means that the shadow ray hits the object that

the ray is casted from.

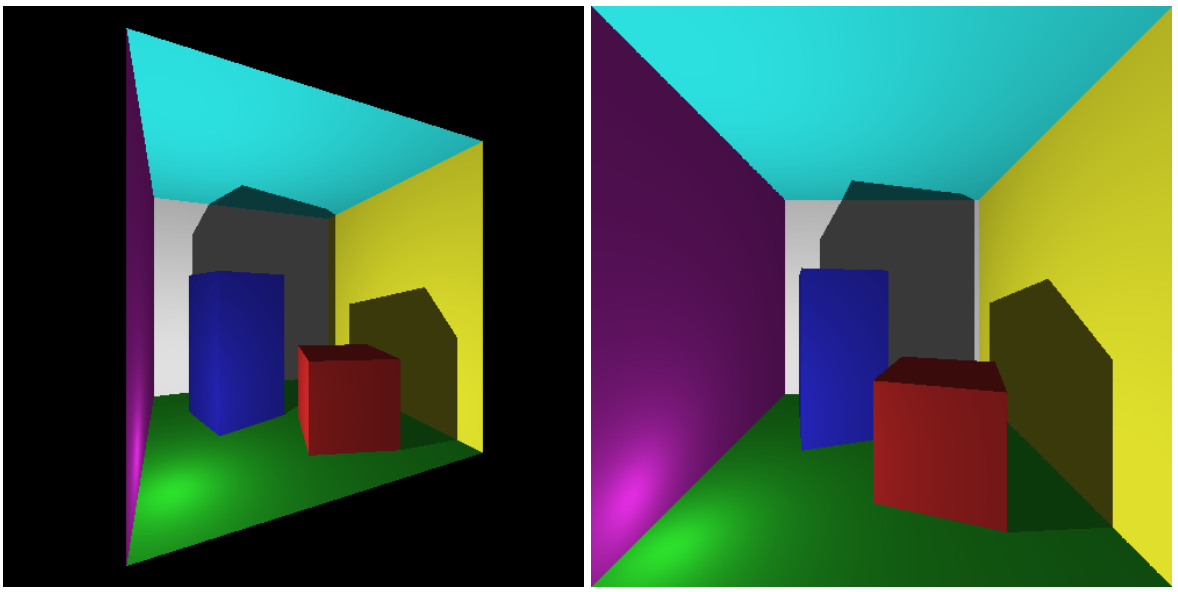
In practice, if we send a light ray into the scene and it hits an object, but the position of intersection is rounded upwards, so that the calculated intersection point is “behind” the object it hit, then when we send a shadow ray from that same position, it may intersect with the same object, telling the algorithm to shade the intersection point, even if the point is not supposed to be in shadow. This causes some objects to be cluttered with black shaded spots.

This can be fixed by introducing a bias, that is a small value which scales the start of the shadow ray towards the light source so that the object to be shaded is not between the shadow’s starting position and the light source.



Left: Self-intersection of shadow rays can be seen on the boxes as well as in the background. Right: Proper shadow calculations, bias introduced to move the start of the shadow ray towards the light source.

Finally, when calculating the constant approximation of indirect illumination, I decided to reduce the indirect lighting factor from 0.5 to 0.3, because I simply found it looking better. The final result when combining direct illumination, shadows and constant approximation of indirect illumination looks like the following image.



Left: *Camera positioned a little further back and rotated right. Light source in bottom left corner. Image illustrates raytraced diffuse lighting, indirect lighting and shadows. Right: Same lighting techniques and similar light position as left but camera is centered.*