Ray-Tracing

DD1354 Modeller och Simulering

Emil Wallgren ewallgre@kth.se
David Peilitz dpeilitz@kth.se

1. Introduction

Ray Tracing is a technique used for modelling how light travels commonly used in rendering of digital images. It is usually used in order to render realistic lighting on 3D models. There exists a wide variety of ray-tracing based rendering techniques that vary in the degree of accuracy and realism. There is usually a trade-off between the detail, thereby accuracy, of the simulation and the computational power required. Therefore, different algorithms are used case by case depending what aspect is prioritised.

2. Method

Where going to use a technique called path tracing as it is relatively simple to implement and creates pretty good results. The basic principle is building an image by extending rays from the camera and recursively generating new rays when interacting with surfaces. Altering the colour and intensity of the rays as they interact with different surfaces. As we are representing a 3D space from a 2D perspective we will have to use quite a bit of linear algebra.

Generating Images

As we are doing the 3D calculations in our code, the only thing that is needed in the way of generating images is the ability to colour pixels on a canvas or window.

Vectors and classes

It is enormously beneficial to approach path tracing with an object oriented mindset as it revolves around simulating things with behaviour that can be easily abstracted such as rays of light. As such one can use three dimensional vectors to represent the movement of light rays. An object that holds 3 values representing the 3 dimensions can both be used to describe vectors and points in 3D space if an origin point is decided.

Light rays

A ray of light can be described using the following formula
$f(x) = A * vx$

Where A is a 3 dimensional point, v is a vector that describes the ray's direction, x is the length of the ray. This behaviour is surmised in the ray class.

Representing Spheres

A sphere centered at the origin that is of radius R can be described using the follow equation:

$$x^2 + y^2 + z^2 = R^2$$

One can therefore describe a sphere centered on a given point C a:

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 = R^2$$

A vector from the spheres center C to any given point P is (P - C) which is an expression of a radius. Therefore we can write the formula as the following

$$R^2 = (P - C) * (P - C)$$

This shows that if the P satisfies this formula, the point lies somewhere on the sphere. If P is replaced with a formula for a ray $F(x) = A * vx$ we can see that if there exists an x for which F satisfies the expression, the ray intersects with the sphere. This can be written as:

$$(F(t) - C)\cdot(F(t) - C) = R^2 \Rightarrow (A + vx - C)\cdot(A + vx - C) = R^2$$

From which we can create the following quadratic equation

$$x^2 v * v + 2tv\cdot(A - C) + (A - C)\cdot(A - C) - R^2 = 0$$

Shading

Shading can be implemented by using the vector that is perpendicular to the point that the ray intersects with a surface. This is called surface normals.

Antialiasing

Antialiasing is a technique used to reduce jagged edges in digital imagery by smoothing out edges by blending the colours of adjacent pixels. This is required as in our ray calculations we only use the average positions of the spheres surfaces which results in jagged edges. Multisampling is an antialiasing technique which involves checking multiple points within each pixel and calculating the average colour, resulting in the edges becoming blended.

Diffuse materials

Diffusion of light can be simulated by using a property called Lambertian reflectance . This will result in the object having a matte appearance. This property can be implemented by taking a random point within the unit sphere that is tangent to the hit point then sending the ray in the direction of this point.

Reflective materials

Perfectly reflective materials will reflect an incoming ray at a right angle. As such a simple expression for a reflective material(such as a metallic surface) was possible to find using some fairly simple vector math which gives that the reflected ray is simply equal to v + 2b, where v is the vector hitting the surface and b is the vector going perpendicular to the surface, from the end point of v, to the extended surface.
It's also possible to make the reflections slightly fuzzy by allowing the reflected ray to not be perfectly perpendicular to the surface, but rather within a sphere with a certain size. The bigger the sphere, the fuzzier the reflection.

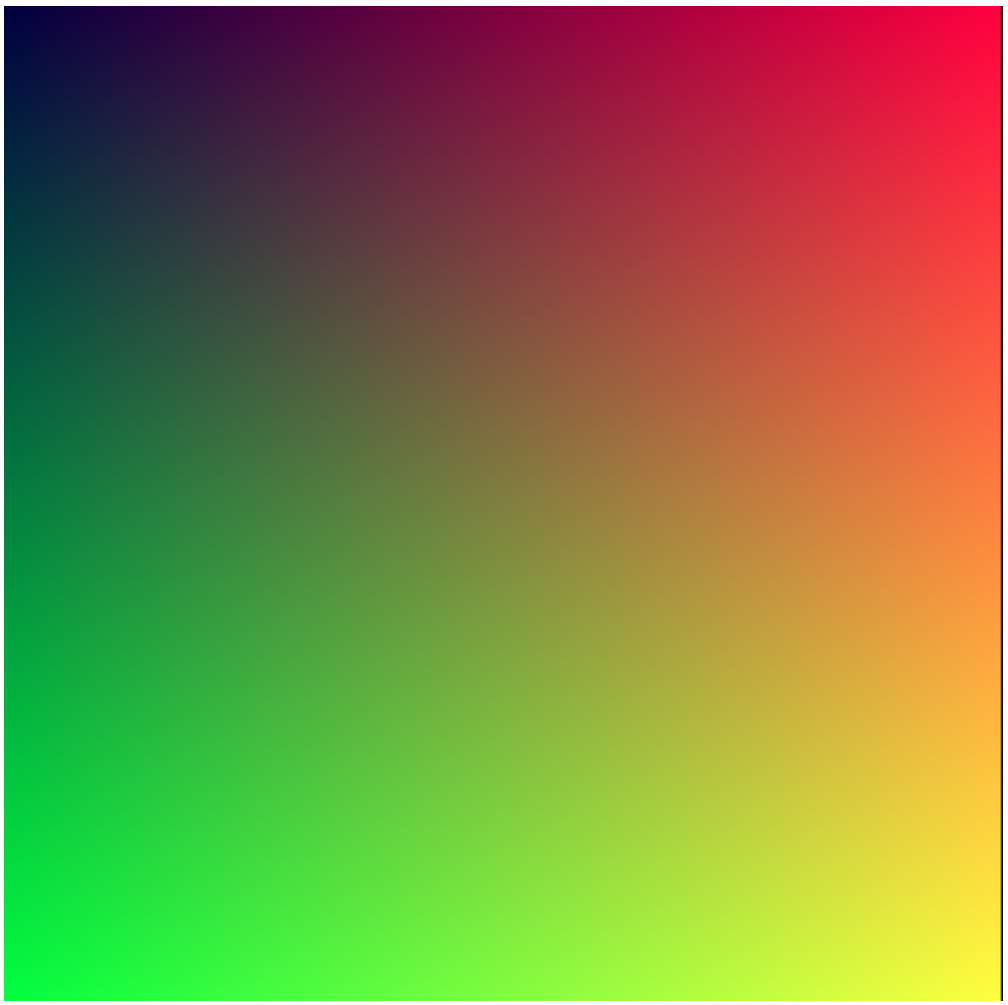
Dielectric materials

Dialectric materials handle both some reflections, but more importantly also has some refraction that needs implementing. Fortunately, refraction can be handled using something known as Snells law. However, there is a quirk to Snells law when the ray is inside the refractive material, since the law doesn't have a solution, which means that the material must then reflect. This is also represented in the code for the dielectric materials hit() function.

A further issue when trying to simulate glass spheres is the fact that reflectivity will vary with an angle. This can be solved with something known as Schlick approximation, which is implemented in the Dialectric Struct, as the reflectance() function.

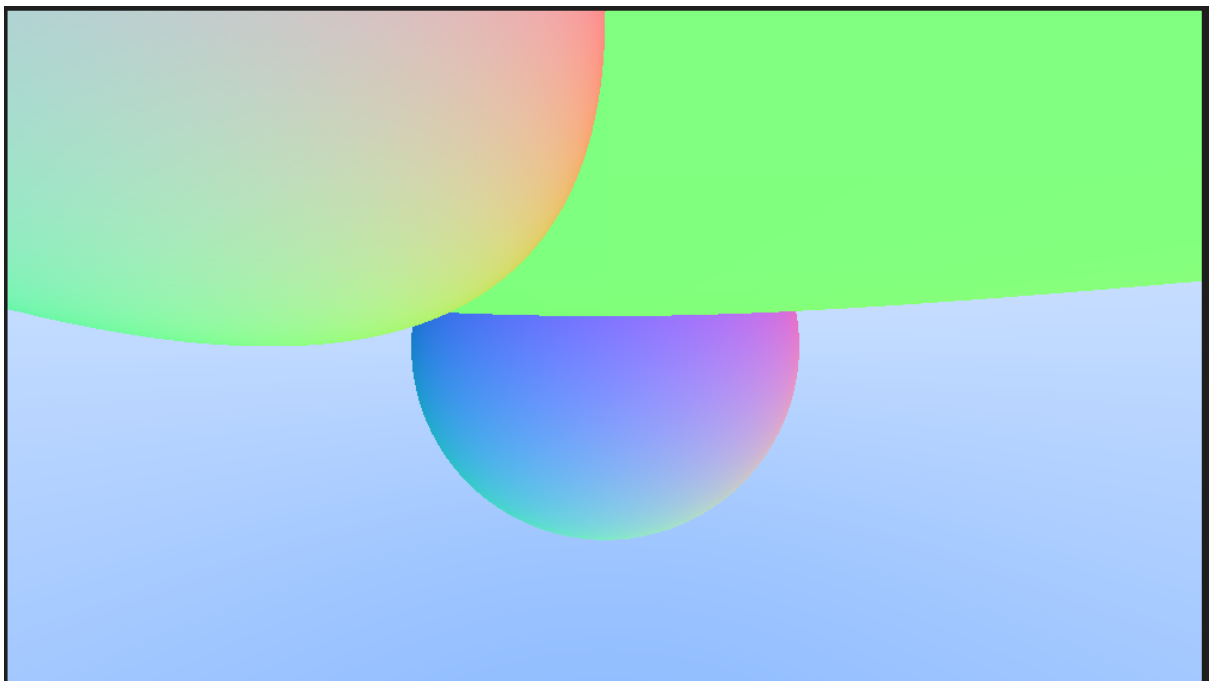
3 Results

At the end result we have an extremely slow but functioning raytracer! Yay! Here are some screenshots of our struggles to getting where we are currently:

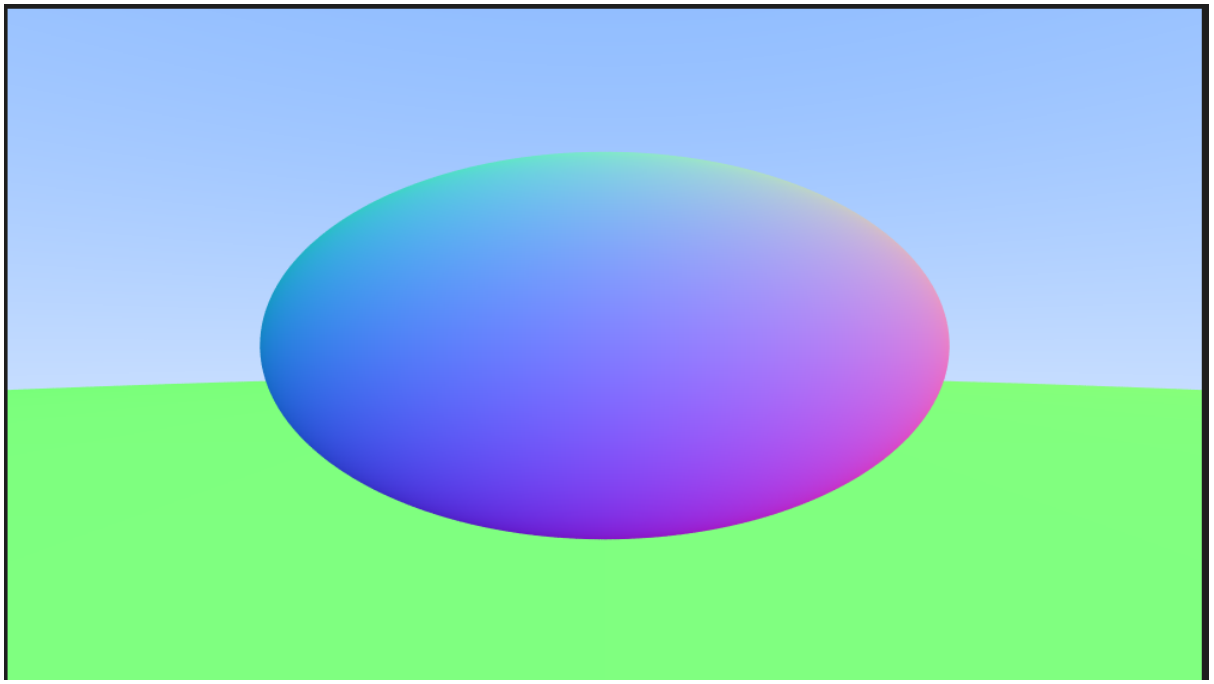


Starting off small, but at least we know we can do colours right!

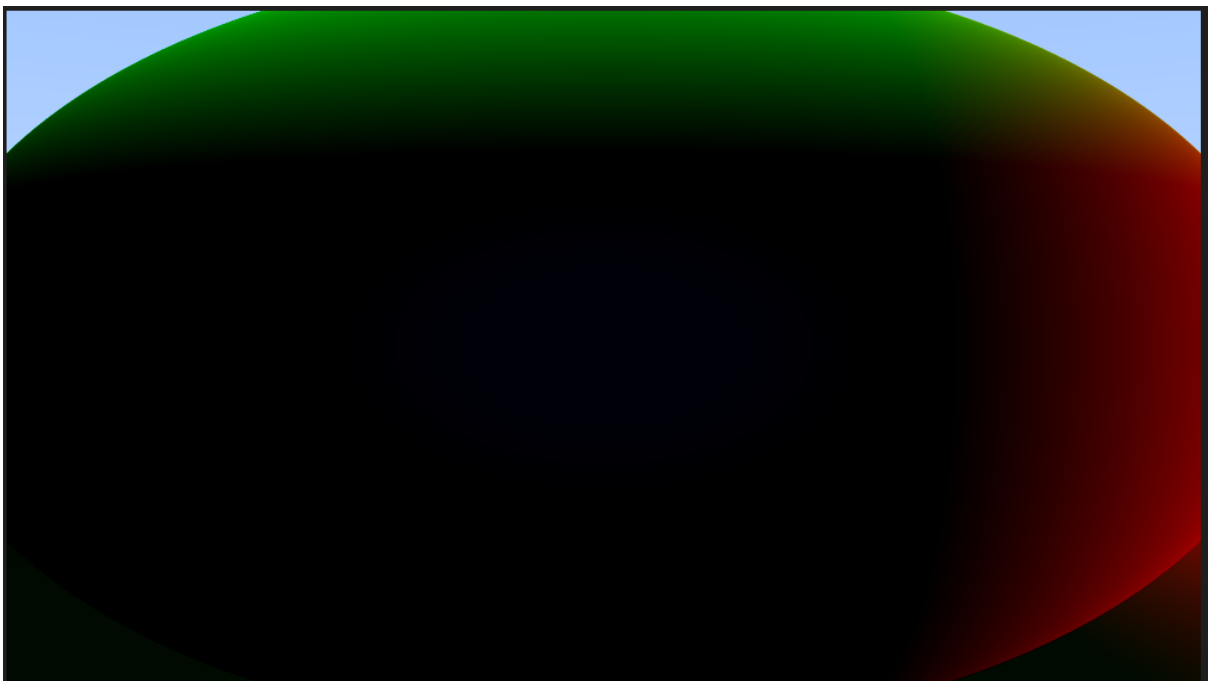
I mean this looks a little like the sky, right?
All right, time to create some spheres! Multicolored sounds good!



I did only intend on inputting the big one and the one in the middle. And why are they inverted

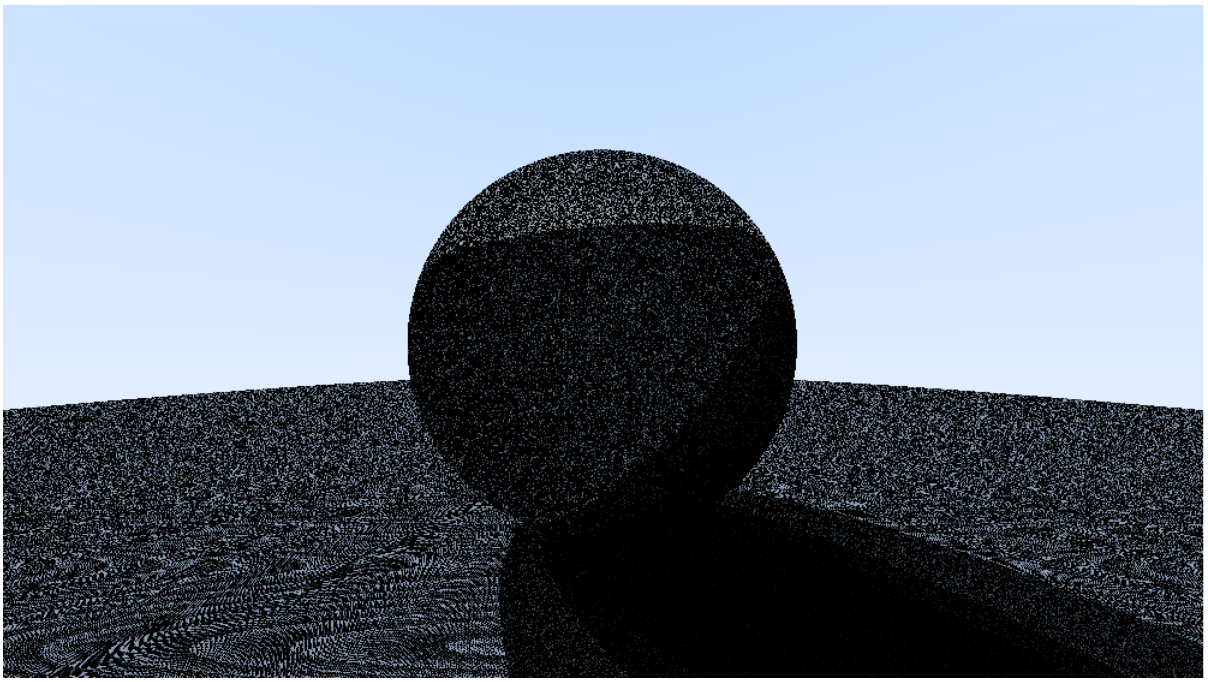


Fixed it! But why is it squished now?

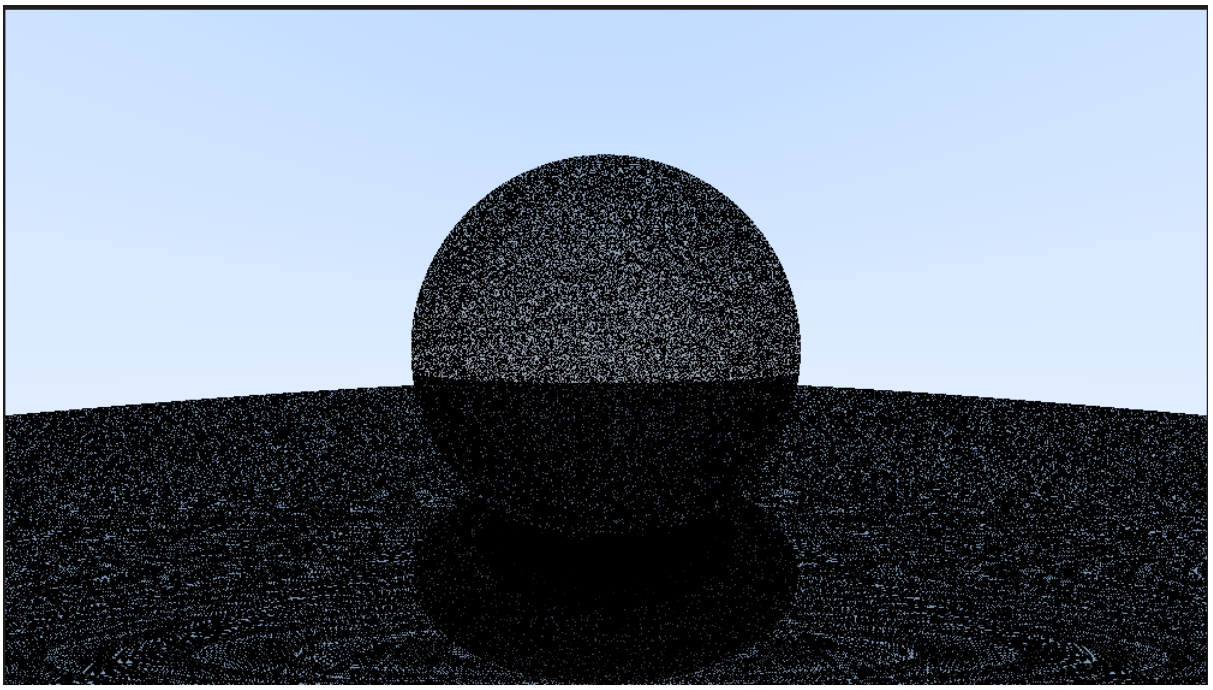


Oh god!

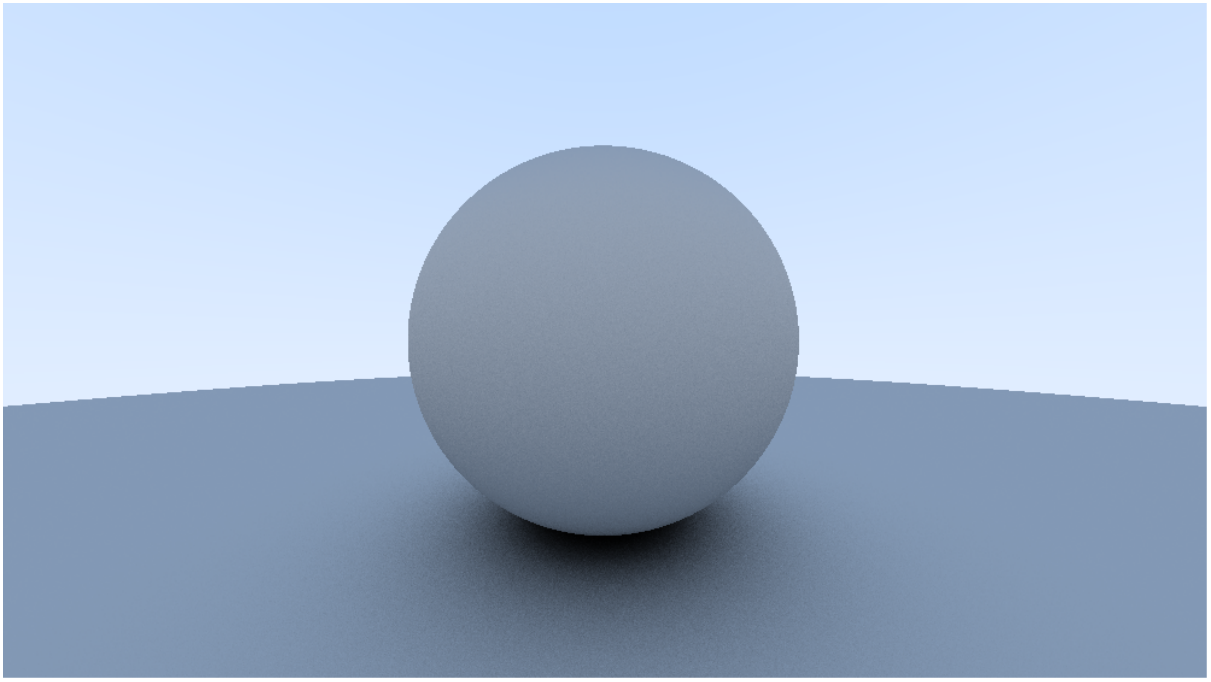
Some troubleshooting, and a first attempt at implementing lambertian materials later:



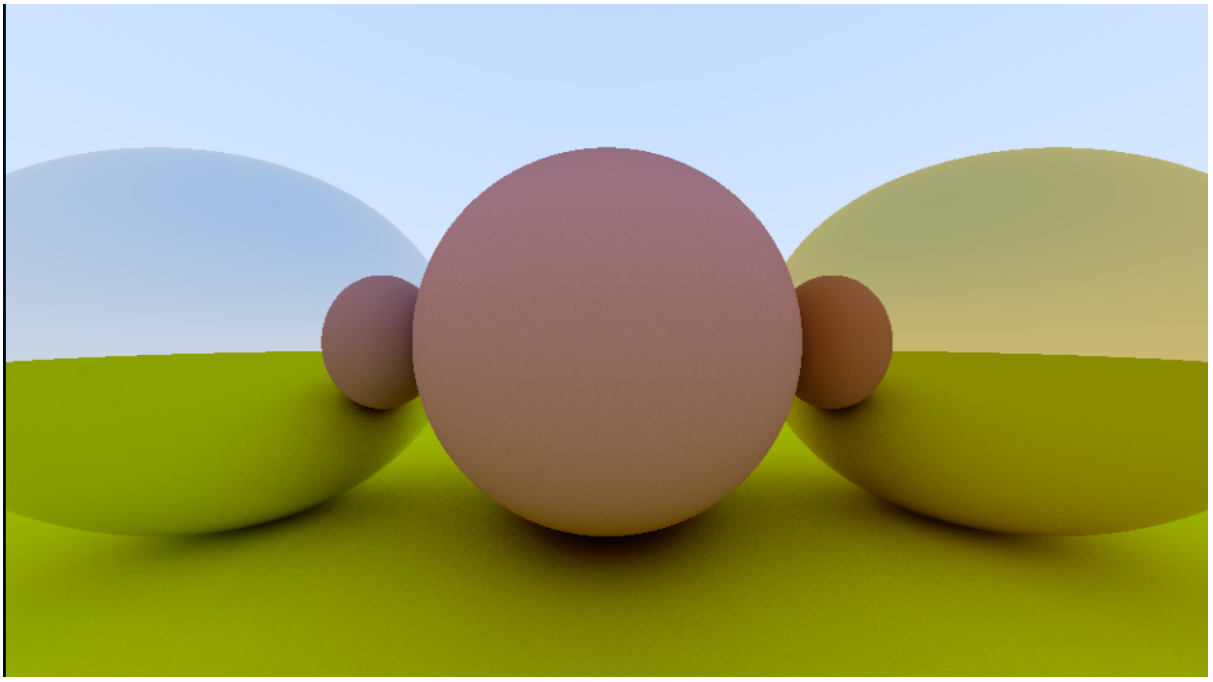
Very dramatic ball staring into the nonexistent sunset. Entirely not on purpose.



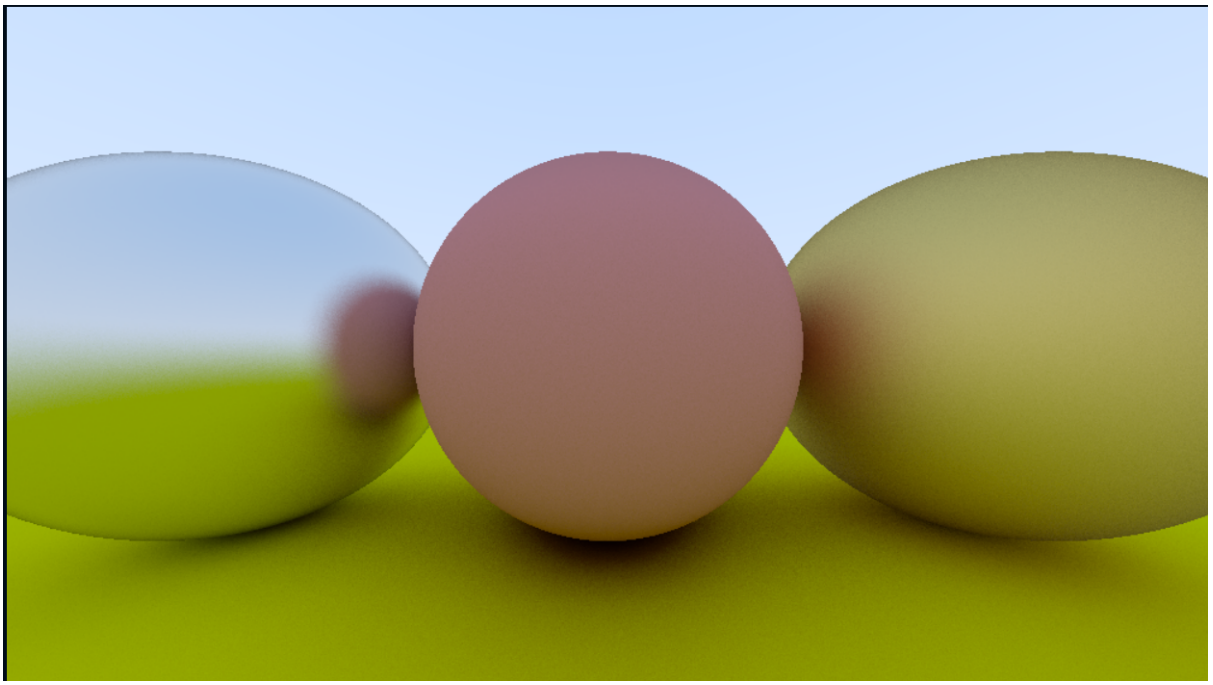
Also why is it seethrough??!?!!



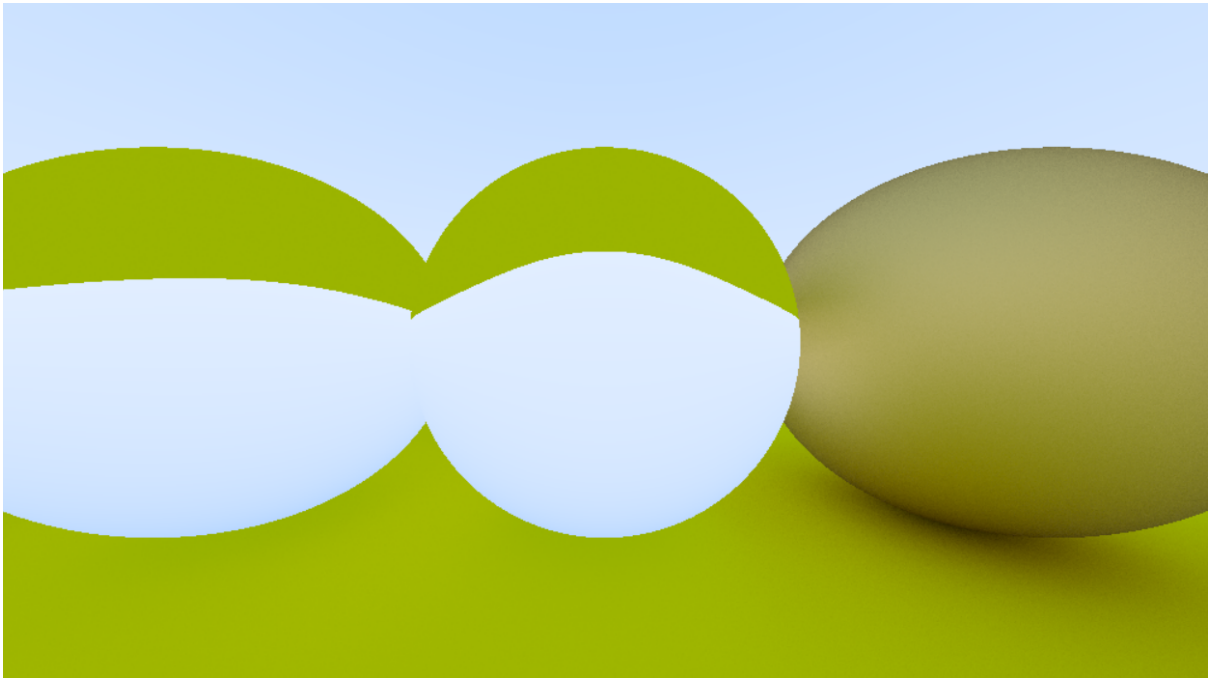
Some struggles and troubleshooting later and we have a lambertian sphere! Woo!



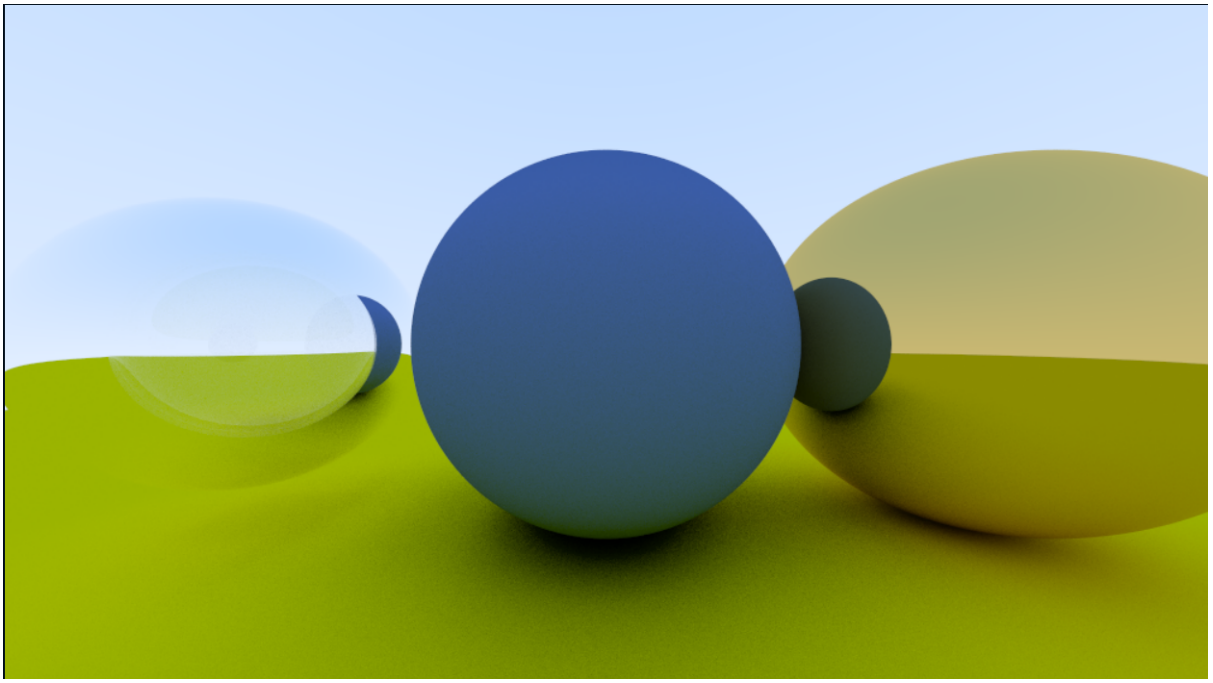
Metallic surfaces implemented!



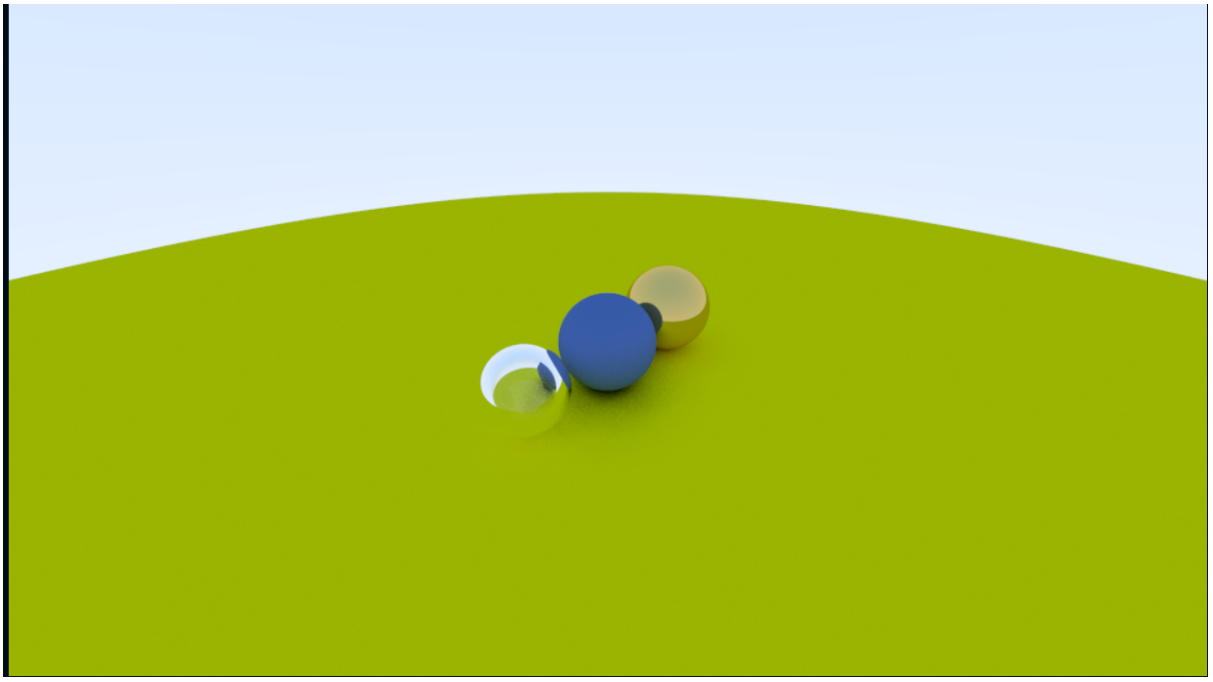
Fuzzy metallics even!



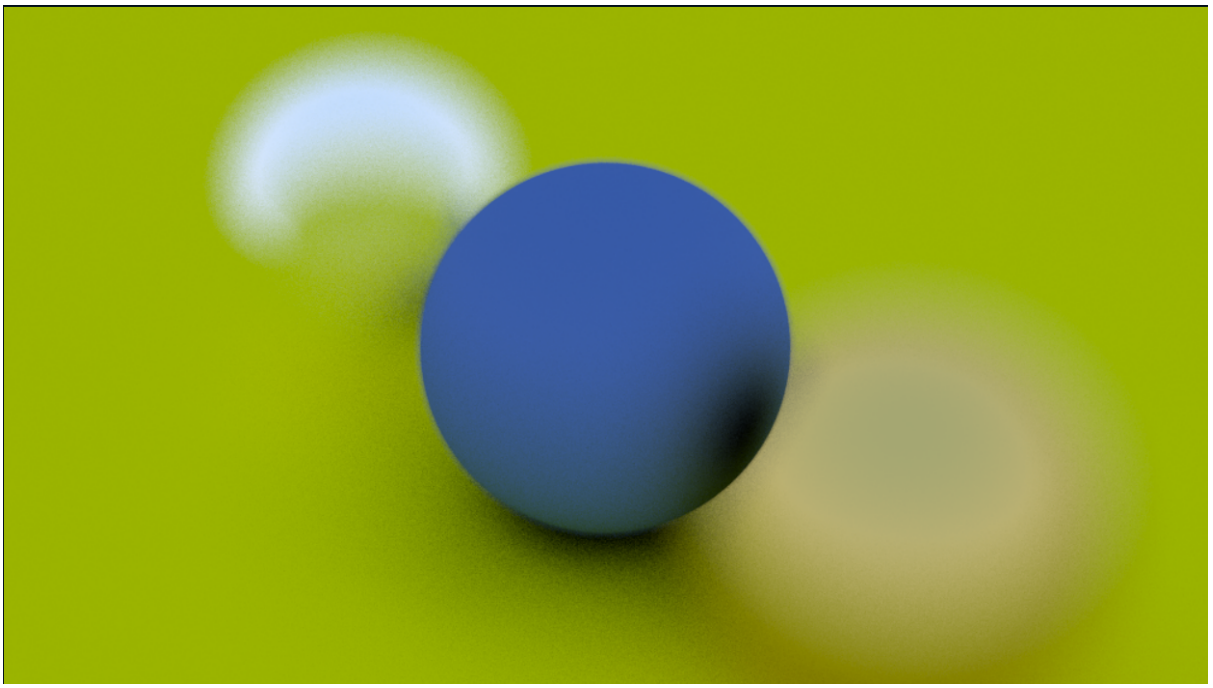
Dielectric surfaces are pretty cool also



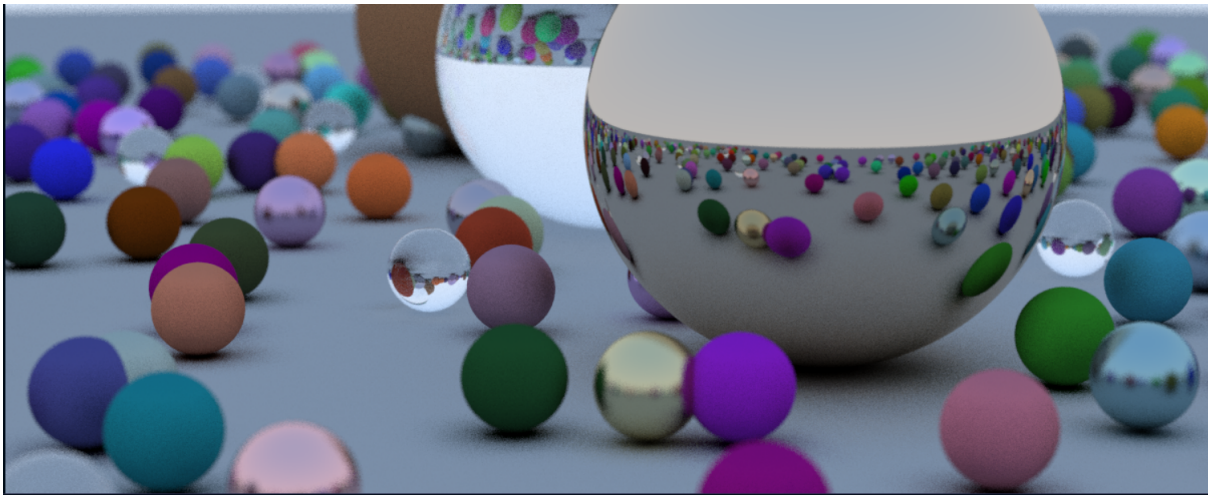
I raise you inverted dialiectrics!



Different camera angles



Close up, and with implementation of lens focus(“Depth of field”).



And a really big environment with randomly placed balls. There are some interesting problems we have not elected to fix just yet, mainly regarding how we store the balls internally causing some to be hit even though they shouldn't .

4 Discussion

Using some fancy linear algebra and similar we finally managed to implement a fairly slow but functioning raytracer. Which we are very happy about.
Note that the library we wanted to use for rendering the images in real time was impossible to compile to an executable file, so unfortunately we were unable to do so. As such the final solution submitted will not be using this window renderer but will simply put the results in a file known as output.ppm.

Two ubuntu-friendly executables will be included in the zip file provided, one much more detailed one and one that takes a lot less time to render. Note that the slow one has a rendering time of 6 hours on our computers.

To run the code provided you need the 0.11 release of Zig and simply run zig build run from the zig folder provided.

Appendix

All the code can be found on this repository: https://github.com/OwlPatrol/modsimrays

KRAV

1. Present the report in a well formatted manner - add a table of contents if the report is large, and appropriate sections and subsections to split text up and make it easier to comprehend and read.

2. Include the main details that you think are important. For example, if you are working as part of a group, a description of the contributions of each group member is a good idea.

3. Include some screenshots in the report. If you did some performance analyses or similar, include some graphs to complement the description.

4. Maybe reflect on the process that you used to complete the project and consider what you could do better/differently if you were starting the project again. What went right? What went wrong?

5. Since you've (hopefully) created a project specification and had some feedback on it, it might be a nice idea to include that in the report and comment on it.

6. Anything else that you think we might be interested to know about the project.