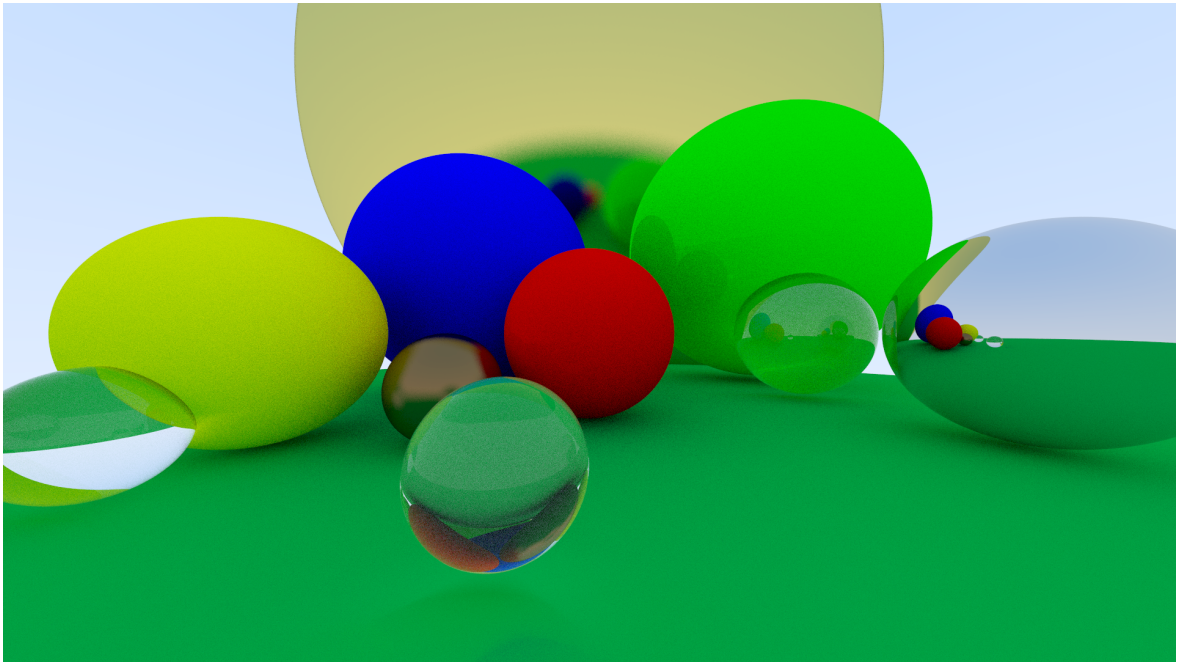


Ray-tracing based renderer from scratch



Timo Salisch

Contents

1	Rendering loop	1
2	Camera	3
3	Objects: shape	4
4	Enhancing camera and rendering loop	5
5	Objects material: diffuse	6
6	Objects material: specular	7
7	Objects material: specular transmission	8
8	Lights	9
9	Positioning and orienting camera	10
10	Animation	11
11	Render parallelization	12
12	Config files	13

1 Rendering loop

The rendering process consists of two loops. One outer loop which iterates through the rows of the image and the inner loop iterating through the columns of the image. This way, every pixel will be visited once and the color of the pixel can be calculated. In this basic version of the rendering loop the color of every pixel will be set to (128, 64, 255). The resulting image can be seen in figure 1.1.

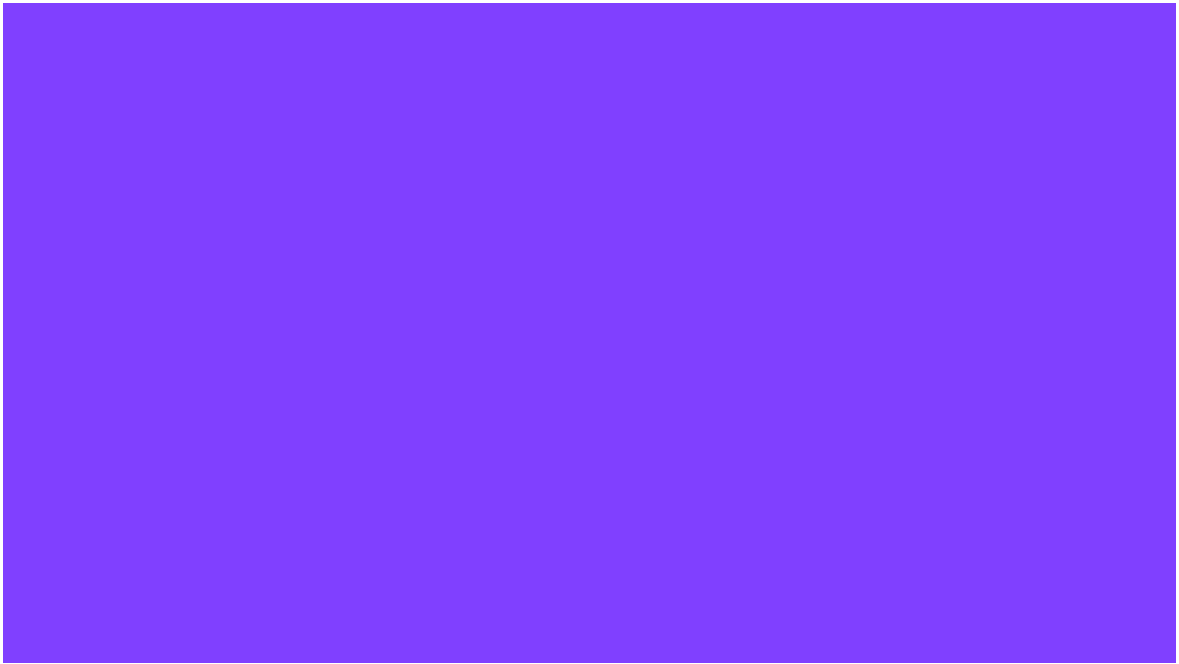


Figure 1.1: Result after running simple rendering loop

To store the already rendered information about the image, an *Image* class is implemented. An instance of this class represents one image and saves the color information of every pixel using a two dimensional list. The color is represented as an instance of the *Vector* class. This *Vector* class saves three floating point values and offers simple

arithmetic operations of (three-dimensional) vectors. These three values can either be interpreted as the world coordinates x , y and z or as the color properties red, green and blue. To change the color value of a specific pixel of the image, the *Image* class provides an *update* method to do so. In order for the image to be viewed, it has to be saved. The image is saved in the "PPM" format. This is done by the *save_image* method which expects a path and then saves the image to that path using the "PPM" format. For the file to be readable as "PPM", the first three lines have to include: "P3", image width and height, max color value. The actual pixel information are added by the loop iterating through the pixels. It is import to iterate through the rows beginning with the top row, otherwise the image is flipped. The saving process can be seen in source code 1.1.

Source code 1.1: Saving an image

```
1 def save_image(self, path: str):
2     image_str = f'P3\n{self.width} {self.height}\n255'
3
4     for j in range(self.height)[::-1]:
5         for i in range(self.width):
6             red, green, blue = self.matrix[i][j].to_tuple()
7             image_str = image_str + f'\n{int(red)} {int(
                green)} {int(blue)}'
8
9     with open(path, mode='w+') as f:
10        f.write(image_str)
```

2 Camera

jhfgjkfgjfhjjfg

3 Objects: shape

jhfgjkfgjfhjjfg

4 Enhancing camera and rendering loop

jhfgjkfgjfhjjfg

5 Objects material: diffuse

jhfgjkfgjfhjjfg

6 Objects material: specular

jhfgjkfgjfhjjfg

7 Objects material: specular transmission

jhfgjkfgjfhjjfg

8 Lights

jhfgjkfgjfhjjfg

9 Positioning and orienting camera

jhfgjkfgjfhjjfg

10 Animation

jhfgjkfgjfhjjfg

11 Render parallelization

jhfgjkfgjfhjjfg

12 Config files

jhfgjkfgjfhjjfg