rptcam: An Open-Source Implementation of a Realistic Camera Model for Computer Graphics

RAHUL KUMAR, ROHAN KUMAR, LUCY MENG, and MICHAEL YU, UC Berkeley, USA

In this paper, we present `rptcam`, an open-source implementation of a realistic camera model written in the Rust programming language. As of now, `rptcam` is capable of simulating custom aperture shapes and lens systems to render images as if they were taken by a real camera. The rendered images display artifacts that are characteristic of real cameras, such as bokeh, depth-of-field, and spherical aberration. Future work will involve simulating more complex lens systems and creating more compelling renders to highlight the capabilities of our camera model.

CCS Concepts: • **Computing methodologies** → **Ray tracing**.

Additional Key Words and Phrases: Ray tracing, Computer Graphics, Camera Model, Aperture, Lenses

ACM Reference Format:
Rahul Kumar, Rohan Kumar, Lucy Meng, and Michael Yu. 2024. rptcam: An Open-Source Implementation of a Realistic Camera Model for Computer Graphics. In *Proceedings of Computer Graphics 2024 (CS 184 '24).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Our goal is to produce an open-source implementation of physically-based camera models. We are building our implementation on top of `rpt`, a path tracing library written in Rust. The functionality of `rpt` is similar in scope and completeness to the path tracer we wrote in HW3.

2 CURRENT PROGRESS AND RESULTS

So far, we have implemented configurable camera aperture shapes and support for custom lens configurations.

2.1 Aperture Shapes

We allow users to configure the aperture shape and diameter for the camera in the scene. We currently support circular and square apertures; we are working on generalizing our implementation to support arbitrary polygon aperture shapes.

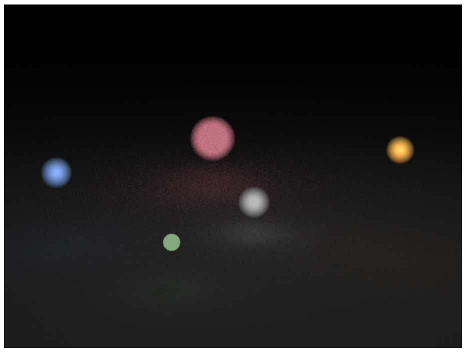
When imaging an out-of-focus point light source with a camera with a non-zero aperture size, the point source will appear on the sensor plane as the aperture shape. That is, a camera with a circular aperture will observe a circular circle of confusion when imaging a point source.

To implement custom aperture shapes, we adapted the ray sampler. Rather than picking a ray going straight through a particular camera pixel, we uniformly sample a point within the camera's aperture shape, and offset the origin of

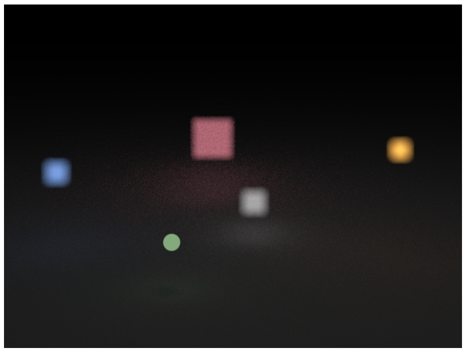
Authors' address: Rahul Kumar, rahulkumar@berkeley.edu; Rohan Kumar, rohankumar@berkeley.edu; Lucy Meng, lucymeng@berkeley.edu; Michael Yu, michaelyu15@berkeley.edu, UC Berkeley, USA.

the ray by that point. We also adjust the direction of the ray so that the ray still travels through the focal point of the camera.

Figure 1 shows results that demonstrate our implementation. We set up a scene (based on example scenes provided by the ‘rpt‘ authors) with several spheres having emissive materials. We rendered the scene twice: once with a circular camera aperture, and once with a square camera aperture. The resulting images are shown below.



(a) Circular aperture



(b) Square aperture

Fig. 1. A scene demonstrating bokeh due to different aperture shapes.

Note that the camera’s focal point is set to the green sphere, so it appears circular even when imaged with a square aperture camera.

2.2 Lens Configurations

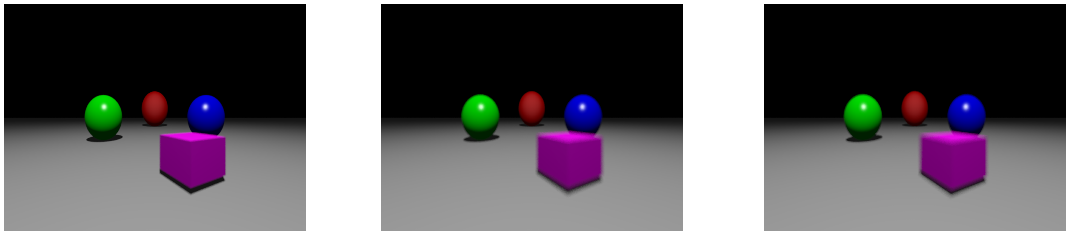
We additionally support custom lens configurations, which was implemented as in [1]. Table 1 gives an example lens configuration for a simple biconvex lens. Each type of lens configuration can specify how its focus point can be shifted (i.e. which optical elements can be moved).

Radius	Thickness	n	d
4	0.01	1.6	0.02
-4	3.995		0.02

Table 1. A sample configuration of a simple biconcave lens. Surfaces are listed from object-facing to image-facing. The first column gives the radius of curvature and is positive for lenses that curve away from object-space. The thickness gives the distance to the next surface along the camera’s axis. n is the index of refraction up to the next surface and d is the aperture diameter. In this case, the specified $n = 1.6$ lens is centered 4 from the image sensor and has two convex surfaces with radius of curvature 4.

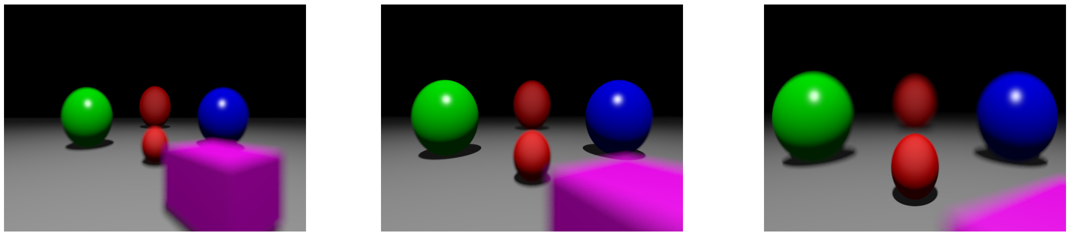
Using a simple biconvex lens, we are able to observe realistic artifacts such as depth-of-field changing with aperture size, spherical aberration, and an inverted image. A couple images rendered using this lens configuration are included below. Figure 2 demonstrates the effect of aperture size on the depth of field of the image. Figure 3 demonstrates the effect of the index of refraction of the lens on the focus point of the camera.

As expected, increasing the aperture size decreases the depth of field as the circle of confusion grows more quickly. Similarly, increasing the index of refraction brings the focus point closer to the camera as the lens refracts light more sharply. It can also be seen that off-axis geometry tends to be more blurry and stretched even when the object is in focus, which demonstrates spherical aberration.



(a) $d = 0.02$ (b) $d = 0.05$ (c) $d = 0.07$

Fig. 2. Images rendered using a biconcave lens configuration with increasing aperture diameter d.



(a) $n = 1.8$ (b) $n = 2.3$ (c) $n = 3.5$

Fig. 3. Images rendered using a biconcave lens configuration with increasing index of refraction n.

3 FUTURE WORK

We are roughly in line with the timeline we included on our proposal. We have completed our week 1 and 2 goals (implementing custom aperture shapes, allowing users to provide custom lens configurations, and implementing raytracing through a series of lenses). We have also read papers/articles about chromatic aberration (one of our stretch goals), but we have not yet started implementing it.

Our week 3 and 4 goals remain the same as what we listed on our proposal. We plan to set up a lens configuration that represents a more realistic camera, compare images generated with our path tracer to images taken by a real camera, and implement a simulated contrast-based autofocus algorithm. Our stretch goal is to implement a physically-based chromatic aberration simulation.

ACKNOWLEDGMENTS

To the CS 184 course staff for providing the idea for this project and feedback on the project's goals. To the authors of `rpt` for creating the base path tracer implementation for us to build our project off of.

REFERENCES

[1] Craig Kolb, Don Mitchell, and Pat Hanrahan. 1995. A realistic camera model for computer graphics. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. Association for Computing Machinery, New York, NY, USA, 317–324. https://doi.org/10.1145/218380.218463

Received 16 April 2024