Simulating Focal Lenses Using Matlab Matrices

Ryan Kropp   
Electrical and Systems Engineering Department  
Washington UniversitySt. Louis, USA  
kropp.r@wustl.edu  
  
Matthew Walker   
Electrical and Systems Engineering Department  
Washington UniversitySt. Louis, USA  
matthew.r.walker@wustl.edu  
  
Robert Walsh   
Electrical and Systems Engineering Department  
Washington UniversitySt. Louis, USA  
r.j.walsh@wustl.edu

For this case study, we sought to use our knowledge of linear algebra to help predict the transmission of light through a lens. We were given base matrices and with that we plotted out rays in reference to the lens. We first created a model that formed rays that started from two different points in the and then generated a rather blurry rendering of the image we were given. For the next part we took our initial conditions and generated the rays as it hit through a lens with a given focal point of 150 mm, using matrix multiplication for each step of the way. We then rendered an even clearer image of the planet, which turned out to be Saturn. Finally, for our last part we wanted to see if we could basically reverse the process, by creating an inverse matrix, and so we created the inverse matrix and ended up with another extremely clear image of Saturn through reverse processing.

# Introduction

For this project, we had to use our base understanding of linear algebra and apply it to the properties of light and its transmission through a lens. When light travels, it splits into many different rays before hitting an object and making it visible. In this instance, we wanted to get a clearer image, so we wanted to control the propagation of the light by essentially modeling it ourselves and sending it through a lens.

For the first part, we just used an initially given matrix to model the rays as they emerged from two different specified points. With that initial matrix we applied it with a given distance and we wished to get some sort of image returned when we propagated it.

For the next part, we essentially just put rays through different transformations with different matrices to get a clearer image through a lens. We had to model the rays as they went across distances of d1 and d2, as well as how it goes across the lens. We did all of this through matrix multiplication, so we could model a new image.

For the last part, we calculated an inverse for our initial M matrix, and then we used reverse processing to compute a resulting image without a need for a physical imaging system.

# Methods

## Modeling the rays through the initial matrices

To begin, we wanted to observe how focal matrices operated in **two dimensions**. Sample light rays were produced using matrices containing information on each individual ray’s x-coordinate, y-coordinate, and the angles corresponding to each dimension. The ray matrices equipped were of the following form:

Rays =

In this 2-dimensional example, only the y coordinates were relevant; the ray plots were linear functions of x, so the slope of each line was determined by y final, which depended on . The rays are the columns of the above array, and the transformed “final array” was determined using the following matrix:

M =

When the above is multiplied with the rays, each individual ray has a new x and y coordinate, calculated by adding d\*Ө. This is an approximation according to the assumption that every angle is very small, so sin(Ө) roughly equals Ө. and matrices were constructed as zeroes or d\*ones matrices so we could set all rays to begin at x = 0 and end at x = d. The ray matrices were constructed so one set of rays began at y = 0 and the other at a random point besides the origin on x = 0. The final y coordinates were calculated via matrix transformation, i.e., M\*Rays, and the lines representing the rays were plotted from initial y values to transformed y values, found in row three of the transformed matrix. The following is the result:



Figure 1: The initially modeled rays in the two different positions

Where the blue and red lines represent rays from different points propagating through our 2D space. By using the rays2img function, we received back this processed image.



Figure 2: The processed image using the initially modeled rays.

## Propogating the rays through a lens

A focal lens takes in rays like the above, and it changes the *angle* of each ray, so they converge on the other side. We simulated this using code by constructing a matrix that inverted Ө depending on its orientation with respect to the x-axis, i.e., a negative Ө would be made positive. The amount by which every angle is altered depends on its distance from the center of the lens, so with a lens centered on y = 0, the transformed y coordinate derived above would be this distance. Sin(Ө2) = -y/f, and once again we use the small-angle approximation to set Ө2 = -y/f. This is the transformation matrix for this process:

Mf =

We changed the ray angles according to position and focal length f, which we set according to the equation:

Where d1 is the object’s distance to the lens, and d2 is the image’s distance to the lens. We chose f with the goal of d2 equaling d1. However, setting f equal to d1/2 did not give us an image d1 away from the lens. Instead, guess and check near this value was used to get a properly oriented image. The following is the transformation equation *after* the same propagation technique from part 1 was performed:

And its result:



Figure 3: The newly modeled rays going through the lens in the two differing positions.

In which the lens is in the middle of our object and image at x = 0.2.

For this experiment, we were also given a much larger matrix of light rays, following the same format used for our sample rays, only where we had 5 rays with 4 descriptors each, this data contained 5,000,000 rays with 4 descriptors each. In the provided data, we are given the x and y coordinates of the light rays, which describe the rays’ positions after traveling from the object to earth, or *after initial d1 transformation*. This time, both sets of coordinates and angles are important as these rays travel along the z axis towards us, propagating through 3D space.

Our goal with this data was to equip the previously used focal transformation matrices in concert with a provided rays-to-image equation to form a proper image of the light source. Using both, the initial image in Figure 1 was transformed into this:



Figure 4: The processed image created by propogating the rays through a lens.

The astral body and its ring can now clearly be made out. The focal length of our “telescope’s lens” was initialized at 300 mm, then altered till a satisfactory image was produced. In the following equation:

d1 should be so large that 1/d2 is roughly equal to 1/f, though this is not the case. Once f was set, d2 was adjusted until image quality was satisfactory.

## Using an Inversible Matrix to Reverse Process

For our last part, the objective was to explore how an inverted matrix could affect the rays in the ray tracing algorithm. An inverted matrix is a special mathematical structure that when multiplied to its base matrix, produces an identity matrix

We were given a restriction on finding this inverted matrix, we could not use a single function to find it. To find the inverse of the matrix, we would have to do it by hand. We settled on using the Gauss-Jordan method. This method allows us to process our matrix and compare it to an identity matrix to find an inversion.

After doing some basic math, we found that all we had to do was eliminate the d values from our M matrix, which gave us a resulting inverse matrix with negative d values in the same position. Knowing this, we were then able to apply our inverted matrix against the ray data.

# Results and Discussion

In creating our rays for part 1, which are shown in Figure 1, we managed to create an image by using the rays2img function in MATLAB, and we ended up with the image in Figure 2. As we can see, it’s extremely blurry but it shows how our rays could still create an image, even without the lens.

So, by using the lens, we could create an exponentially clearer image and we can model the lens through matrices. We had to model not only the ray modifications along the distances of d1 and d2, but the ray modification that the lens accounts for as well.

We can send the rays along each pathway via matrix multiplication, and our rays get augmented in different ways when they begin at different positions, as seen in Figure 3. With our newly propagated rays, we can produce a much clearer picture of the same image shown in Figure 2, and this new image is shown in Figure 4.

Finally, we were able to prove that the inverted matrix, M-1 exists. Since M is a square matrix with a dimension size of 4 by 4, that means that it should be an invertible matrix. To find the inverted matrix, we used Gaussian elimination. Since M was nearly in an identity matrix form, all we had to do was deal with the d values. This gave us a resulting inverse matrix that was nearly identical to M but with the d values being negative rather than positive.

Comparing this matrix to an M-d matrix, where the d value is set to a negative value, we can see that M-1 is equivalent to M-d. This equivalence does make sense considering how they are used. In the equation provided to us, we are taking a resulting set of rays, presumably obtained by multiplying the original rays by M, and multiplying it by the inverse of M to re-obtain our original rays. The product in this case would effectively be going backwards by our selected distance. M-d would be the same process. It would be the rays processed through a backwards distance just like M-1 .

Taking our new matrix, we used it to process the light field dataset. By propagating the rays through our inverted matrix with a selected distance of 1, we managed to obtain a shockingly clear and detailed image, one far greater than the image of part 2. We can now see that the object we have been ray tracing from was a planet that appears to have been Saturn the whole time (Figure 5).

A picture containing text, monitor

Description automatically generated

Figure 5: The final, clear, processed image of the planet Saturn using our inverted matrix.

# Conclusion e

In conclusion, we ended up creating a completely clear image from an extremely blurry picture using reverse processing and a dynamical system. This shows one of the many real-world applications of matrix multiplication, and this case study was yet another great example of how linear algebra can be applied to real world scenarios. While we had initial challenges in correctly propagating the rays, our final result was perfect, showing the efficiency of our system of matrices and functions.

# REFERENCES

[1] Boyd, S., & Vandenberghe, L. (2018). Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares. Cambridge: Cambridge University Press. doi:10.1017/9781108583664

[2] “MATLAB Deep Learning Toolbox.” Deep Learning Toolbox Documentation, <https://www.mathworks.com/help/deeplearning/index.html>.

[3] “Physics Tutorial: Refraction and the Ray Model of Light.” The Physics Classroom, https://www.physicsclassroom.com/class/refrn/Lesson5/Refraction-by-Lenses