

# Case Study 3: Optical Imaging as a Linear System

ESE 105, Fall 2022

DUE: Tuesday, November 22, 2022, at 5 PM to [Canvas](#)

In this case study you will use the skills and methods you have learned in linear algebra and MATLAB to simulate how an optical imaging system, such as a digital camera, a microscope, or your eye, forms images in free space. The methodology you will use is called *ray tracing*.

*Ray optics* is the simplest theory of light and comprises a set of geometrical rules that describe how rays travel. Therefore, ray optics is also called *geometrical optics*. These geometrical rules are simple to implement in the language of linear algebra.

Your goals are threefold:

1. Simulate rays leaving an object. Explore how they travel through space, and how a camera sensor visualizes light.
2. Explore how a lens bends rays and forms images.
3. Explore an application of computational imaging: digitally refocus an image without using a physical lens.

## 1 Part 1: Ray propagation in free space

### 1.1 Background physics

Light travels in the form of rays. They are emitted by light sources, reflect off of objects, and can be observed when they reach an optical detector (e.g., your eyes or a camera sensor). A ray is described by its position  $(x, y)$  and its direction (angles  $\theta_x$  and  $\theta_y$ ) with respect to the primary axis  $z$  of travel, called the optical axis.

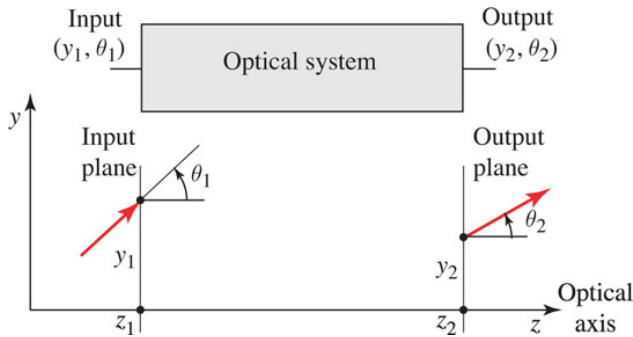


Figure 1: A ray enters an optical system at position  $y_1$  and angle  $\theta_1$  and leaves the system at position  $y_2$  and angle  $\theta_2$ . Figure from Ref. [1].

To simplify the situation, let's first assume that a ray begins at position  $y_1$  and travels within the  $yz$  plane at angle  $\theta_1$  with respect to the  $z$  axis (Figure 1). We represent an optical system using a ray-transfer matrix  $\mathbf{M}$ , which describes how the incoming ray  $(y_1, \theta_1)^T$  is transformed by  $\mathbf{M}$  into an outgoing ray  $(y_2, \theta_2)^T$ , such that

$$\begin{bmatrix} y_2 \\ \theta_2 \end{bmatrix} = \mathbf{M} \begin{bmatrix} y_1 \\ \theta_1 \end{bmatrix}. \quad (1)$$

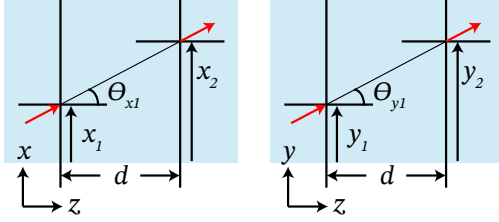


Figure 2: Free-space propagation in the (left)  $xz$  and (right)  $yz$  planes. Figure adapted from Ref. [1].

Rays travel in straight lines when they are not disturbed by objects. We also need to generalize the ray-transfer matrix concept to three dimensions, not just the  $yz$  plane. Therefore, after the ray traverses a distance  $d$  along the optical axis (Figure 2), we know

$$x_2 = x_1 + d \tan(\theta_{x1}) \approx x_1 + d\theta_{x1} \quad (2)$$

$$\theta_{x2} = \theta_{x1} \quad (3)$$

$$y_2 \approx y_1 + d\theta_{y1} \quad (4)$$

$$\theta_{y2} = \theta_{y1}. \quad (5)$$

To make this process linear (so we can use the linear algebra used in this class!), we have made the approximation that  $\tan \theta \approx \theta$ . Here,  $(x_1, y_1)$  represents the location of the ray in the input plane;  $(x_2, y_2)$  is the location of the ray in the output plane. The angle  $\theta_x$  represents ray's direction of travel in the  $xz$  plane, while  $\theta_y$  represents the angle the ray makes with the  $z$  axis in the  $yz$  plane.

Therefore, we can now define a 3D ray-transfer matrix  $\mathbf{M}_d$  that represents propagation in free space, given by

$$\mathbf{M}_d = \begin{bmatrix} 1 & d & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (6)$$

where

$$\begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix} = \mathbf{M}_d \begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} \quad (7)$$

can be used to compute the new position and angle of any ray after traveling a distance  $d$ .

## 1.2 Supporting data and code

The case study provides data in `lightField.mat` and code in `rays2img.m`:

1. You are given a simulated dataset containing the locations and propagation directions for  $N = 5 \times 10^6$  rays, stored in `lightField.mat`. (Be sure to use a broadband internet connection to download the case study .zip file; the light field dataset alone is ~72 MB!) This file contains rays, a  $4 \times N$  matrix, with each row describing:
  - the  $x$  position of  $N$  rays
  - the propagation direction (in radians) in the  $xz$  plane of  $N$  rays
  - the  $y$  position of  $N$  rays
  - the propagation direction (in radians) in the  $yz$  plane of  $N$  rays

2. You are also given a function `rays2img(rays_x,rays_y,width,Npixels)` that simulates the operation of a camera sensor:

---

```

1 function [img,x,y] = rays2img(rays_x,rays_y,width,Npixels)
2 % rays2img - Simulates the operation of a camera sensor, where each pixel
3 % simply collects (i.e., counts) all of the rays that intersect it. The
4 % image sensor is assumed to be square with 100% fill factor (no dead
5 % areas) and 100% quantum efficiency (each ray intersecting the sensor is
6 % collected).
7 %
8 % inputs:
9 % rays_x: A 1 x N vector representing the x position of each ray in meters.
10 % rays_y: A 1 x N vector representing the y position of each ray in meters.
11 % width: A scalar that specifies the total width of the image sensor in
12 % meters.
13 % Npixels: A scalar that specifies the number of pixels along one side of
14 % the square image sensor.
15 %
16 % outputs:
17 % img: An Npixels x Npixels matrix representing a grayscale image captured
18 % by an image sensor with a total Npixels^2 pixels.
19 % x: A 1 x 2 vector that specifies the x positions of the left and right
20 % edges of the imaging sensor in meters.
21 % y: A 1 x 2 vector that specifies the y positions of the bottom and top
22 % edges of the imaging sensor in meters.

```

---

### 1.3 Tasks

*Milestone 1: You should complete all items within this task by the end of class on Wednesday 11/16.*

1. *Explore ray tracing.* Simulate the rays originating from two points on an object: one at  $(x, y, z) = (0, 0, 0)$  and the other at  $(x, y, z) = (10, 0, 0)$  mm. For each point, simulate rays at several small angles, e.g., 5 rays with angles  $\theta_x$  between  $-\pi/20$  rad and  $\pi/20$  rad. For simplicity, let  $y = 0$  and  $\theta_y = 0$  for all rays in your simulation.

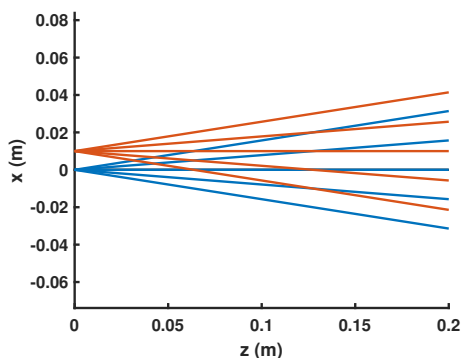


Figure 3: A fan of rays propagating through free space, originating from two points on an object located at  $z = 0$ . Ray colors correspond to the origin of each ray on the object: (blue)  $x = 0$  and (red)  $x = 10$  mm.

Propagate the rays using Equation (7) and  $\mathbf{M}_d$  for some  $d > 0$ . Use `plot()` in MATLAB to generate a 2D ray diagram similar to that in Figure 3. Show where your rays originate from, and show

where they travel after some distance  $d > 0$ . You only need to show the  $xz$  plane (ignore the  $y$  direction since  $y = 0$  for all rays in your simulation). Your diagram *does not* need to look exactly like the one in Figure 3.

*Hint:* To plot the ray diagram, Consider the MATLAB syntax `plot(X,Y)`, which plots the columns of  $X$  vs. the columns of  $Y$ . Thus, you can use MATLAB to draw any number of rays (line segments) between any arbitrary number of locations. The following example plots the ray trajectories between two  $z$  planes:  $z = 0$  and  $z = d$ :

---

```

1 % rays_in is a 4 x N matrix representing the rays emitted from an object
2 % rays_out is a 4 x N matrix representing the rays after propagating distance d
3 ray_z = [zeros(1,size(rays_in,2)); d*ones(1,size(rays_in,2))];
4 plot(ray_z, [rays_in(1,:); rays_out(1,:)]);

```

---

## 2. Lightfield dataset.

- (a) Load `lightField.mat`. Use the function `rays2img()` to render an image of the rays. Use reasonable values for the sensor width (e.g., 5 mm) and the number of pixels (e.g., 200). What do you see? Can you discern the object that generated the rays?

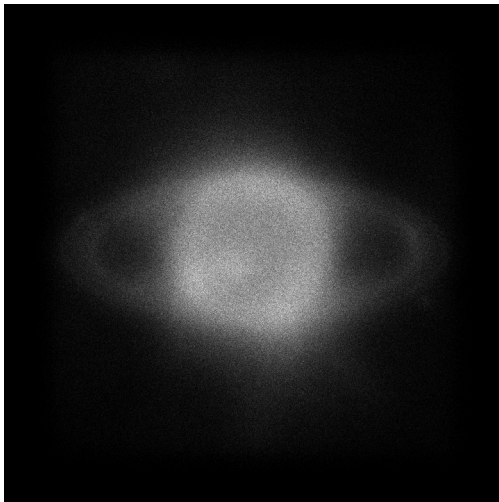


Figure 4: Image created from the light field dataset `lightField.mat` using the function `rays2img()`. [A movie visualizing the rays in the lightField.mat dataset](#) is located on Canvas in the Case Study 3 module.

- (b) Can you recover a sharp image by increasing/decreasing the sensor width? Why or why not?
- (c) Can you recover a sharp image by increasing/decreasing the number of sensor pixels? Why or why not?
- (d) Use Equation (7) and  $\mathbf{M}_d$  to propagate the rays by some distance  $d > 0$ , and then visualize the rays using `rays2img()`. What happens to the rays after propagation? Is there a value of  $d > 0$  that will create a sharp image?
3. Based on your analysis above and what you know about optics and rays (e.g., how your cell phone camera works and our class discussions), why are you not able to create a sharp, focused image from the rays in the dataset using the propagation matrix  $\mathbf{M}_d$ ? Appeal to mathematical concepts as appropriate.

*Hint:* There are no sources of background light in the dataset, so you don't have to worry about a bright light corrupting the image. Each of the rays `lightField.mat` originates from a specific point on the object, entering the optical system at a random angle with respect to the optical axis.

## 2 Part 2: Lenses to create images

### 2.1 Background physics

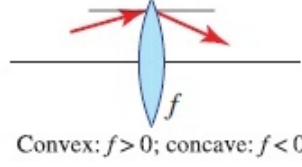


Figure 5: Propagation through a thin lens. Figure from Ref. [1].

The purpose of a lens is to change the direction of each ray that passes through it, *i.e.*, to “bend” the rays. In a thin lens (Figure 5), we assume that a ray exits at the same position as it enters. A lens’s focal length  $f$  is defined such that if a ray is travelling parallel to the optical axis ( $\theta_1 = 0$ ), then its exit angle  $\theta_2$  is given by

$$\theta_2 = -\frac{y}{f}. \quad (8)$$

That is, the ray intersects the optical axis at a distance  $f$  behind the lens, called its focal point. Again, we assume that the lens bends each ray *linearly* with respect to both height and angle, *i.e.*,  $\theta_2 = -y/f + \theta_1$ .

We therefore have

$$\mathbf{M}_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1/f & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/f & 1 \end{bmatrix}, \quad (9)$$

such that

$$\begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix} = \mathbf{M}_f \begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} \quad (10)$$

can be used to compute the new position and angle of any ray after passing through a lens of focal length  $f$ .

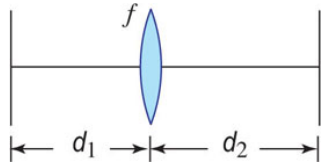


Figure 6: A thin lens imaging system. The object plane located at a distance  $d_1$  in front of the lens corresponds to an image plane located at a distance  $d_2$  behind the lens, subject to Equation (11). Figure from Ref. [1].

A thin lens  $\mathbf{M}_f$  may be combined with two free space propagations  $\mathbf{M}_{d1}$  and  $\mathbf{M}_{d2}$  to create an *imaging system* (Figure 6). The distances  $d_1$  and  $d_2$  must be compatible with the focal length  $f$  such that

$$\frac{1}{d_1} + \frac{1}{d_2} = \frac{1}{f}. \quad (11)$$

Under this *imaging condition*, all rays originating from a location  $(x, y)$  in the  $z$ -plane located a distance  $d_1$  in front of the lens, called the *object plane*, will be bent by the lens such that they converge *at a single point*  $(x', y')$  in the  $z$ -plane located at a distance  $d_2$  behind the lens, called the *image plane*. Thus, the imaging system forms a *one-to-one mapping* between locations in the object plane and locations in the image plane.

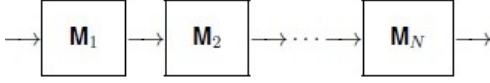


Figure 7: A series of optical components, where rays exiting system  $\mathbf{M}_1$  enter system  $\mathbf{M}_2$ , which then enter system  $\mathbf{M}_3$ , and so on. Figure from Ref. [1].

A series of optical elements with ray transfer matrices  $\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_N$  (Figure 7) is equivalent to a single optical component with ray-transfer matrix  $\mathbf{M}$  given by

$$\mathbf{M} = \mathbf{M}_N \cdots \mathbf{M}_2 \mathbf{M}_1. \quad (12)$$

Thus, the output rays of the system in Figure 7 can be computed from the input rays using

$$\begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix} = \mathbf{M}_N \cdots \mathbf{M}_2 \mathbf{M}_1 \begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} = \mathbf{M} \begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix}. \quad (13)$$

Note the order of operations: rays are first transformed by the first optical component  $\mathbf{M}_1$ , then by  $\mathbf{M}_2$ , and so on.

## 2.2 Tasks

*Milestone 2: You should complete all items within this task by the end of Friday 11/18.*

1. *Visualizing imaging using ray tracing.* Modify your ray tracing simulation in Section 1.3 to include a thin lens  $\mathbf{M}_f$  of focal length  $f = 150$  mm and a second propagation step  $\mathbf{M}_{d2}$ . Choose a reasonable value of  $d_2$  such that you are able to visualize the convergence of your rays to two unique points, similar to Figure 8.

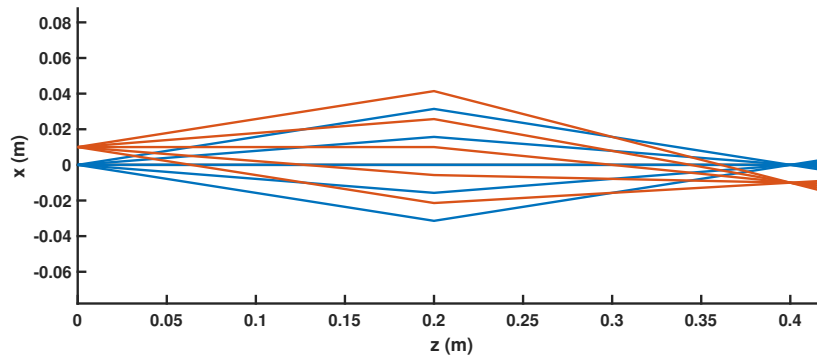


Figure 8: A fan of rays propagating through free space, bent by a thin lens at  $z = 200$  mm, and then propagating through free space again. Ray colors correspond to the origin of each ray on an object at  $z = 0$ : (blue)  $x = 0$  and (red)  $x = 10$  mm. Note the convergence of the rays in an image plane located at  $z = 400$  mm. *Your ray diagram will not look exactly like this one because your lens focal length is different.*

Model the travel of your rays using Equations (7) and (13),  $\mathbf{M}_{d1}$ ,  $\mathbf{M}_{d2}$ , and  $\mathbf{M}_f$ . Use `plot()` in MATLAB to generate a 2D ray diagram similar to that in Figure 8. Be sure the ray paths are legible in your plot. *Your diagram will not look exactly like the one in Figure 8 because your lens focal length is different.*

## 2. Create an image.

- (a) Design and implement an optical system that creates a sharp image from the `lightField.mat` dataset. In particular, you will need to use the imaging concepts above (Sections 1.1 and 2.1) to design a propagation matrix  $\mathbf{M}_{d_2}$  and a lens matrix  $\mathbf{M}_f$  such that these matrices will create an imaging system (Equation (11)). Report the values of  $d_2$  and  $f$  in your design.
- (b) Transform the rays from the `lightField.mat` dataset appropriately (Equation (13)) using your lens and propagation matrices. Use `rays2img()` to create an image from your rays after you transform them.
- (c) Identify the object that emitted the light rays.

Note that you *do not know* the value of  $d_1$  for the `lightField.mat` dataset. This situation exactly mirrors the real-world imaging scenarios encountered by your cell phone, digital camera, and eyes. All of these imaging systems must adjust themselves dynamically (*i.e.*, “hunt” for focus by adjusting  $f$ ,  $d_2$ , or both) so that they can create sharp images. In general, none of these systems “know” the exact value of  $d_1$  (how far the objects are away from them), and they must be able to focus on objects across a large range of distances (*e.g.*, 20 mm to infinity).

## 3 Part 3: Toward computational imaging

### 3.1 Background

Engineers and physicists have been designing optical imaging systems using the core concepts in Sections 1.1 and 2.1 for centuries. However, with recent incredible developments in computing power, *computational imaging* has *become important* in *today’s imaging innovations*. We’ll briefly explore a few of those concepts here.

Examine Equation (7). If we possess information on a handful of rays originating from an object (like we do in the `lightField.mat` dataset), then it seems natural for us to leverage our linear algebra knowledge to *compute* an image of the object *rather than using a physical imaging system* to create the image on a camera.

More explicitly, we know

$$\begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} = \mathbf{M}_d^{-1} \begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix}, \quad (14)$$

as long as inverse  $\mathbf{M}_d^{-1}$  exists. Thus, it is trivial for us to *compute* the original locations where rays were reflected by an object of interest, as long as we know *both the ray positions* ( $x_2, y_2$ ) *and ray travel directions* ( $\theta_{x2}, \theta_{y2}$ ) after the rays travel some distance away from the object. This measurement is actually non-trivial, and *several* companies have *attempted* to *market* these technologies in the past.

### 3.2 Tasks

*Milestone 3: You should complete all items within this task by the end of class on Monday 11/21.*

1. Does the matrix inverse  $\mathbf{M}_d^{-1}$  exist? Justify mathematically.

If the inverse exists, then write an expression for  $\mathbf{M}_d^{-1}$ . You should be able to calculate the inverse by hand, *without using a software package*.

2. Compare your expression of  $\mathbf{M}_d^{-1}$  to the propagation matrix  $\mathbf{M}_{-d}$ , that is, the matrix in Equation (6) calculated for a *negative distance*  $-d$ . Are they equivalent? Does this equivalence make physical or practical sense? Use diagrams and/or conceptual arguments in your explanation.
3. *Revisit the lightfield dataset.* Propagate the rays in `lightField.mat` using your expression of the matrix inverse  $\mathbf{M}_d^{-1}$ . Find a suitable distance  $d$  such that you are able to see a sharp image of the object using the function `rays2img()`.

Are you able to recover a sharp image of the object? How does the image compare to the image you generated in Part 2, item 2?

## 4 Tips

- Use matrix and vector operations within your code where possible.
- You have complete freedom to implement code for all parts of the case study as you wish.
- Be efficient with your coding style! In particular, the matrix multiplications in Equations (7) and (13) can be computed using a *single line of code*, even for any number of rays! Please avoid implementations that manually repeat a large number of scalar operations.

## 5 What to turn in

1. Any MATLAB `.m` files you write or modify
2. PDFs or Word DOCs for each of the MATLAB files above. Use the `publish()` command similar to what you've produced for previous MATLAB homeworks.
3. A 3-5 page report, answering each question within the case study and documenting your design choices. *Use the provided Word or Overleaf/L<sup>A</sup>T<sub>E</sub>X template.* Make sure that your report clearly states/presents:
  - (a) Answers to each question/prompt within the “tasks” sections of the case study
  - (b) Justification to your answers using calculations or plots from your MATLAB code when possible
  - (c) Design choices that you've made
4. A signed version of the provided honor code

## 6 Rubric

- Correctness of MATLAB code - 40%
  - Implementation of ray-transfer matrices and ray calculations using efficient and proper matrix syntax in MATLAB.
  - Plotting of 2D ray diagrams and images in the light field dataset
- Presentation - 20%
  - Plots are easy to read and interpret, with appropriate font sizes, line widths, axis labels, etc.



- Report should be well-organized, concise, and clearly written.
- Programming style - 20%
- Study design - 20%
  - Strategy for exploring various imaging system parameters.
  - Rigor of responses to the various prompts/questions posed in the “tasks” sections of the case study.

## References

- [1] B. E. Saleh and M. C. Teich, *Fundamentals of Photonics*, 3rd ed. Hoboken, NJ: John Wiley & Sons, Inc., 2019.