Case Study 3: Optical Imaging as a Linear System

ESE 105, Fall 2023

DUE: Tuesday, November 21, 2023, 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 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. Compute ray propagation through free space and lenses using linear algebra and MATLAB.
2. Visualize how light propagates through an imaging system and discover what it means for an image to be "formed."
3. Explore computational imaging by manipulating rays emitted by a hologram.

1 Part 1: Ray tracing

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 θ_x and θ_y) with respect to the primary axis z of travel, called the optical axis.

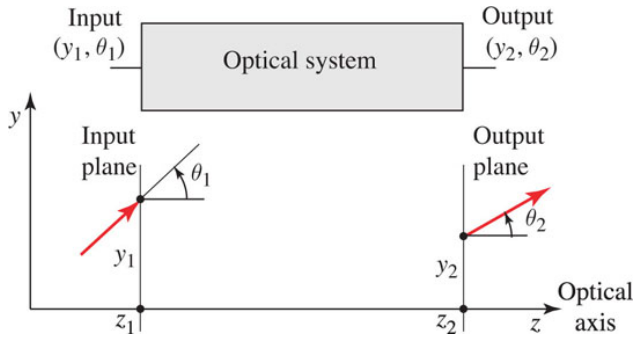

Figure 1: A ray enters an optical system at position y_1 and angle θ_1 and leaves the system at position y_2 and angle θ_2. Figure from Ref. [1].

To simplify the situation, let's first assume that a ray begins at position y_1 and travels within the yz plane at angle θ_1 with respect to the z axis (Figure 1). We represent an optical system using a ray-transfer matrix $\boldsymbol{M}$, which describes how the incoming ray (y_1, θ_1) is transformed by $\boldsymbol{M}$ into an outgoing ray (y_2, θ_2), such that

$$\begin{bmatrix} y_2 \\ \theta_2 \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} y_1 \\ \theta_1 \end{bmatrix}. \tag{1}$$

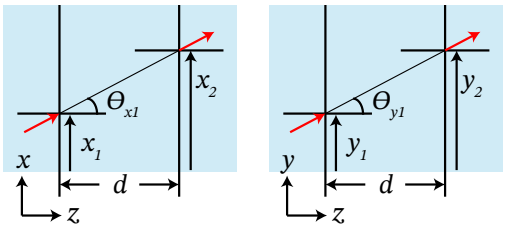

Figure 2: Free-space propagation in the (left) xz and (right) yz planes. Figure adapted from Ref. [1].

1.1 Ray propagation in free space

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

$$x_2 = x_1 + d\tan(\theta_{x1}) \approx x_1 + d\theta_{x1} \tag{2}$$
$$\theta_{x2} = \theta_{x1} \tag{3}$$
$$y_2 \approx y_1 + d\theta_{y1} \tag{4}$$
$$\theta_{y2} = \theta_{y1}. \tag{5}$$

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 θ_x represents ray's direction of travel in the xz plane, while θ_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 $\boldsymbol{M}_d$ that represents propagation in free space, given by

$$\boldsymbol{M}_d = \begin{bmatrix} 1 & d & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

where

$$\begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix} = \boldsymbol{M}_d \begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} \tag{7}$$

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

1.2 Lenses

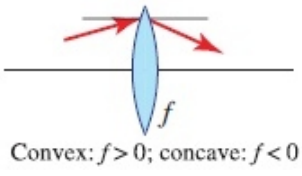

Figure 3: Propagation through a thin lens. Figure from Ref. [1].

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 3), 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 θ_2 is given by

$$\theta_2 = -\frac{y}{f}. \tag{8}$$

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

$$\boldsymbol{M}_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1/f & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/f & 1 \end{bmatrix}, \tag{9}$$

such that

$$\begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix} = \boldsymbol{M}_f \begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} \tag{10}$$

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

1.3 Modeling an entire imaging system

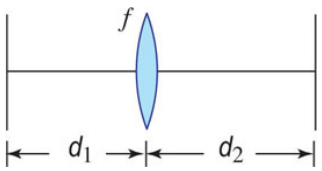

Figure 4: A thin lens imaging system. The object plane located at a distance d_1 in front of the lens corresponds to an image plane located at a distance d_2 behind the lens, subject to Equation (11). Figure from Ref. [1].

A thin lens $\boldsymbol{M}_f$ may be combined with two free space propagations $\boldsymbol{M}_{d1}$ and $\boldsymbol{M}_{d2}$ to create an *imaging system* (Figure 4). The distances d_1 and d_2 must be compatible with the focal length f such that

$$\frac{1}{d_1} + \frac{1}{d_2} = \frac{1}{f}. \tag{11}$$

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.

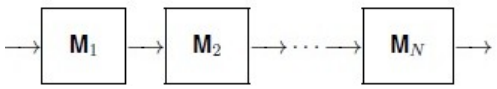

Figure 5: A series of optical components, where rays exiting system $\boldsymbol{M}_1$ enter system $\boldsymbol{M}_2$, which then enter system $\boldsymbol{M}_3$, and so on. Figure from Ref. [1].

A series of optical elements with ray transfer matrices $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_N$ (Figure 5) is equivalent to a single optical component with ray-transfer matrix $\boldsymbol{M}$ given by

$$\boldsymbol{M} = \boldsymbol{M}_N \cdots \boldsymbol{M}_2 \boldsymbol{M}_1. \tag{12}$$

Thus, the rays produced by the system in Figure 4 can be computed from the input rays using

$$\begin{bmatrix} x_2 \\ \theta_{x2} \\ y_2 \\ \theta_{y2} \end{bmatrix} = \boldsymbol{M}_{d2}\boldsymbol{M}_f\boldsymbol{M}_{d1}\begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix} = \boldsymbol{M}\begin{bmatrix} x_1 \\ \theta_{x1} \\ y_1 \\ \theta_{y1} \end{bmatrix}. \tag{13}$$

Note the order of operations: rays are first transformed by the first matrix $\boldsymbol{M}_{d1}$ that represents propagation by a distance d_1, then by $\boldsymbol{M}_f$ that represents the lens, and finally by $\boldsymbol{M}_{d2}$ that represents propagation by a distance d_2.

1.4 Tasks–lab exercise due Friday 11/17

Complete all items within this task by the end of lab on Friday 11/17. Turn in PDFs of your code and plots using `publish()`*, along with your .m files.*

1. *Ray tracing through free space.* 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.*, 8 rays with angles θ_x between $-\pi/20$ rad and $\pi/20$ rad. For simplicity, let $y = 0$ and $\theta_y = 0$ for *all rays in your simulation*.

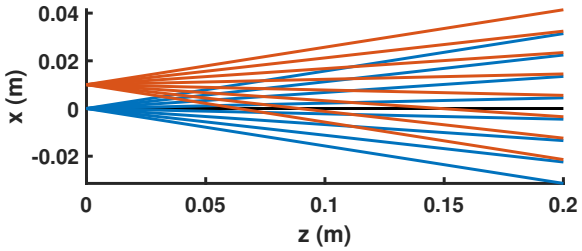

Figure 6: A fan of rays propagating through free space, originating from two points on an object located at $z = 0$. Ray colors correspond to the origin of each ray on the object: (blue) $x = 0$ and (red) $x = 10$ mm.

   Propagate the rays using Equation (7) and $\boldsymbol{M}_d$ for some $d > 0$. Use `plot()` in MATLAB to generate a 2D ray diagram similar to that in Figure 6. 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 6.

   Hint: To plot the ray diagram, consider using the *matrix form* of `plot(X,Y)`, which plots the columns of `X` vs. the columns of `Y`. Thus, you can use *one line of code* 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 $z = d$:

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

2. *Ray tracing through a finite-sized lens.* Modify your ray tracing simulation to include a thin lens $\boldsymbol{M}_f$ of *focal length* $f = 150$ mm and *radius* $r_{\text{lens}} = 20$ mm and a second propagation step $\boldsymbol{M}_{d2}$. 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 7.

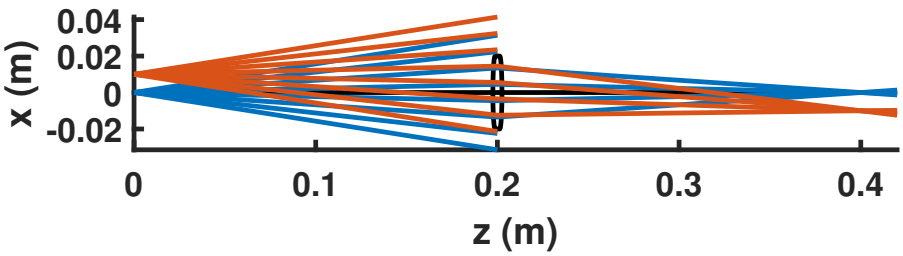

Figure 7: A fan of rays propagating through free space, bent by a thin lens of radius 20 mm at $z = 200$ mm, and then propagating through free space again. Rays that "miss" the lens are "lost" at $z = 200$ mm and are no longer plotted on the diagram. Ray colors correspond to the origin of each ray on an object at $z = 0$: (blue) $x = 0$ and (red) $x = 10$ mm. *Notice the location of where the red rays converge in the image plane $z = 0.4$ m, compared to their origin at $z = 0$.*

Model the travel of your rays using Equations (7) and (13), $\boldsymbol{M}_{d1}$, $\boldsymbol{M}_{d2}$, and $\boldsymbol{M}_f$. Note that any ray that "misses" the lens (because it lands outside of the lens radius at $z = 0.2$ m in Figure 7) is "lost" and not redirected toward the image plane.

Use `plot()` in MATLAB to generate a 2D ray diagram similar to that in Figure 7. Be sure the ray paths are legible in your plot.

Your diagram *will not* look exactly like the one in Figure 7 because *your lens focal length is different*.

2 Part 2: Simulating a hologram

The Nobel Prize in Physics 1971 was awarded to Dennis Gabor "for his invention and development of the holographic method." Today, holography has many applications in imaging, sensing, art, 3D displays, security, and data storage. We'll use our linear algebraic framework as a tool to understand how a hologram works and how to use an imaging system to capture information from it.

2.1 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 $N = 3 \times 10^6$ rays, stored in `lightField.mat`. 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:

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

2.2 Tasks

1. *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?

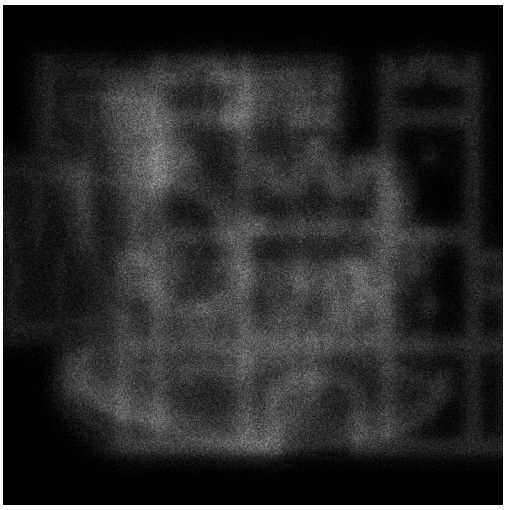

Figure 8: Image created from the light field dataset `lightField.mat` using the function `rays2img()`. A movie visualizing the rays in the `lightField.mat` dataset is located on Canvas in the Case Study 3 module.

   (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 $\boldsymbol{M}_d$ 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?

2. 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 $\boldsymbol{M}_d$? 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.

3. *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 to 1.3) to design a propagation matrix $\boldsymbol{M}_{d2}$ and a lens matrix $\boldsymbol{M}_f$ 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) Can you identify the object that emitted the light rays? Why or why not?

   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 camera, your eyes, etc. 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: Competition

You most likely had difficulty creating a sharp image in Section 2.2, which stems from how holograms interact with light. Find a credit card or other object with a holographic security sticker nearby. *Notice how the object looks different depending on how you tilt it.*

This effect is one of the fundamental characteristics of a hologram; it interacts with incoming light in a special way such that the *interference pattern changes as you observe the hologram from different directions*. This property enables holograms to look "3D," as if they were real objects, instead of looking like a "flat" 2D image like normal photographs or digital screens.

Your tasks in this competition are:

1. *Discover* the identities of the *3 objects* contained within the `lightField.mat` dataset. Utilize any physics or data-driven approaches *that are based on topics covered in ESE 105* to accomplish this task. *Show images of the objects* in your report.

   No credit will be given for computational techniques or software packages not taught in ESE 105.

2. *Describe* in your report *your process* for discovering the images in the `lightField.mat` dataset. Your explanation should be clear enough so that any other ESE 105 student can follow your description to reproduce your results.

The competition will be scored holistically in terms of 1) the scientific and technical reasoning used to discover the images, 2) the clarity of your narrative describing your process for discovering the images, and 3) identification of the objects.

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 to produce your PDFs.
3. A 3-5 page report, answering each question within the case study and documenting your design choices. *Use the provided Word or Overleaf/LaTeX 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–3 pts
  - 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–3 pts
  - 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.

- Design rationale–3 pts
  - Are the processes and approaches described in the report well-reasoned?
  - Does the report show that students learned from their computations and observations?
- Design execution–3 pts
  - Do the computations and data presented in the report harmonize with the rationale given in the report?
  - Do the computations and data presented in the report address the tasks and questions posed in the case study instructions?
- Reproducibility–2 pts
  - Is the report written clearly enough so that a fellow student can reproduce the results 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.