

University of Nevada, Reno

Looking Mass: Detecting and Analyzing Gravitational Lensing

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

by

Ethan M. Park

Aditya Sidher

Marcus Casey

Nikita Ignatyuk

Dr. Emily Hand, Thesis Advisor

May, 2020

**UNIVERSITY
OF NEVADA
RENO**

THE HONORS PROGRAM

We recommend that the thesis
prepared under our supervision by

ETHAN M. PARK

entitled

Looking Mass: Detecting and Analyzing Gravitational Lensing

be accepted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Emily Hand, Ph.D., Thesis Advisor

Matthew Means, P.S., Director, **Honors Program**

May, 2020

Abstract

Gravitational lensing occurs when light is distorted by gravity and its presence can cause distortions in images of space. Manual calculations to detect gravitational lensing can be slow and cumbersome, and the use of machine learning methods such as CNNs are impeded by the fact that there are not enough images containing gravitational lensing to create large datasets. Looking Mass is an open-source software application that modifies source images to produce new images that realistically simulate gravitational lensing. The team intends for datasets generated from Looking Mass to be used for gravitational lensing detection methods.

Acknowledgements

I would like to thank team members Aditya Sidher, Marcus Casey, and Nikita Ignatyuk for their contributions in the development and completion of Looking Mass. I would also like to thank instructors Dr. David Feil-Seifer, Dr. Sergiu Dascalu, and Ms. Devrin Lee as well as external advisors Dr. Emily Hand and Dr. Melodi Rodrigue for their guidance in the development of Looking Mass. Finally, I would like to personally thank Dr. Emily Hand for her advisement of this thesis. Without the collective efforts of these individuals, this project would not have been possible.

Table of Contents

| | |
|---|-----|
| Abstract | i |
| Acknowledgements | ii |
| Table of Contents | iii |
| List of Tables | v |
| List of Figures | vi |
| List of Illustrations | vii |
| 1 Introduction | 1 |
| 2.1 Main Goals and Objectives | 3 |
| 2.2 Main Functionality and Characteristics | 3 |
| 2.3 Intended Audience | 4 |
| 2.4 Technology Description | 4 |
| 3 Significance | 5 |
| 3.1 Why It's Worthwhile | 5 |
| 3.2 Impact on Professional Growth | 5 |
| 3.3 Similar Works | 6 |
| 3.4 Project Impact and Context Considerations | 7 |
| 4 Specification | 7 |
| 4.1 Elicitation | 7 |
| 4.2 Technical Requirements | 9 |
| 4.3 Use Cases | 11 |
| 5 Design | 16 |
| 5.1 System-Level Diagram | 16 |
| 5.1.1 GUI | 16 |
| 5.1.2 CNN | 16 |
| 5.1.3 File I/O | 17 |
| 5.1.4 Image Processor | 17 |
| 5.1.5 GravLens Generator | 17 |
| 5.2 Program Units | 18 |

| | |
|--|----|
| 5.3 Main Data Structures..... | 22 |
| 5.4 Detailed Design..... | 24 |
| 6 Implementation | 29 |
| 6.1 Project Summary..... | 29 |
| 6.1.1 Functionality Implemented..... | 29 |
| 6.1.2 Functionality Not Implemented..... | 31 |
| 6.1.3 Contributions of Team Members..... | 33 |
| 6.2 User Interface Snapshots..... | 34 |
| 7 Future Work..... | 43 |
| 8 Conclusion | 44 |
| Bibliography | 46 |
| Appendix A: Glossary of Terms..... | 47 |

List of Tables

| | |
|---|----|
| Table 1: Non-functional Requirements for Looking Mass. | 9 |
| Table 2: Functional Requirements for Looking Mass. | 10 |
| Table 3: Detailed Use Cases for Looking Mass..... | 12 |
| Table 4: Template for Use Case 4..... | 13 |
| Table 5: Template for Use Case 5..... | 13 |
| Table 6: Template for Use Case 8..... | 14 |
| Table 7: Description of the classes of Looking Mass. | 19 |
| Table 8: Data structures used for CNN_Controller. | 22 |
| Table 9: Data structures used for PhysicsMetadata. | 23 |
| Table 10: Use cases implemented..... | 29 |
| Table 11: Functional requirements implemented. | 30 |
| Table 12: Non-functional requirements implemented. | 30 |
| Table 13: Use cases not implemented..... | 31 |
| Table 14: Functional requirements not implemented. | 32 |
| Table 15: Non-functional requirements not implemented. | 32 |

List of Figures

| | |
|--|----|
| Figure 1: Image distortion caused by gravitational lensing | 2 |
| Figure 2: Use Case Diagram for Looking Mass. | 11 |
| Figure 3: Requirement Traceability Matrix for Looking Mass. | 15 |
| Figure 4: Context Model of Looking Mass..... | 16 |
| Figure 5: Class Diagram for Looking Mass..... | 18 |
| Figure 6: Activity diagram of analyze an image for gravitational lensing. | 25 |
| Figure 7: Activity diagram of generated images being fed to the CNN. | 26 |
| Figure 8: Flowchart representing the StartCNN() function. | 27 |
| Figure 9: Flowchart representing the ImageReader::read() function..... | 28 |
| Figure 10: Main Menu of Looking Mass..... | 34 |
| Figure 11: Input File Selection System of Looking Mass. | 35 |
| Figure 12: A source image displayed in Looking Mass. | 36 |
| Figure 13: Output File Selection System of Looking Mass..... | 37 |
| Figure 14: Metadata Entry System for Looking Mass..... | 38 |
| Figure 15: Coordinate Selection System of Looking Mass. | 39 |
| Figure 16: Pop up indicating image processing has started..... | 40 |
| Figure 17: Pop up indicating image processing has ended. | 41 |
| Figure 18: A destination image displayed in Looking Mass. | 42 |
| Figure 19: Image generated by Looking Mass in computer's file system. | 43 |

List of Illustrations

| | |
|--|---|
| Illustration 1: How gravity bends light, causing gravitational lensing | 2 |
|--|---|

1 Introduction

Gravitational lensing is an effect of general relativity and occurs when light is distorted by gravity as shown in Illustration 1. It can often be caused from different sources such as galaxies, dark matter, and black holes. Gravitational lensing can result in images of space being distorted as shown in Figure 1. This distortion can help researchers find these sources of gravitational lensing without having to physically see the source. The detection of gravitational lensing using manual calculations is often time consuming and requires a great deal of work.

A machine learning approach to the detection of gravitational lensing is hindered by the fact that the total number of images that contain gravitational lensing is relatively small. Therefore, it is difficult to apply machine learning methods, such as training a convolutional neural network (CNN), to detect gravitational lensing as datasets are too small compared to what is ideally needed.

The Looking Mass software development team seeks to address this issue by developing an application that can artificially generate realistic gravitational lensing in images. Looking Mass is an application that modifies source images to produce new images that realistically simulate gravitational lensing based on user specifications.

Ideally, Looking Mass will be used by researchers to produce datasets of images with realistic gravitational lensing. These datasets can then be used to train methods for gravitational lensing detection. Theoretically, such methods can significantly decrease the time it takes to determine whether an image contains gravitational lensing.

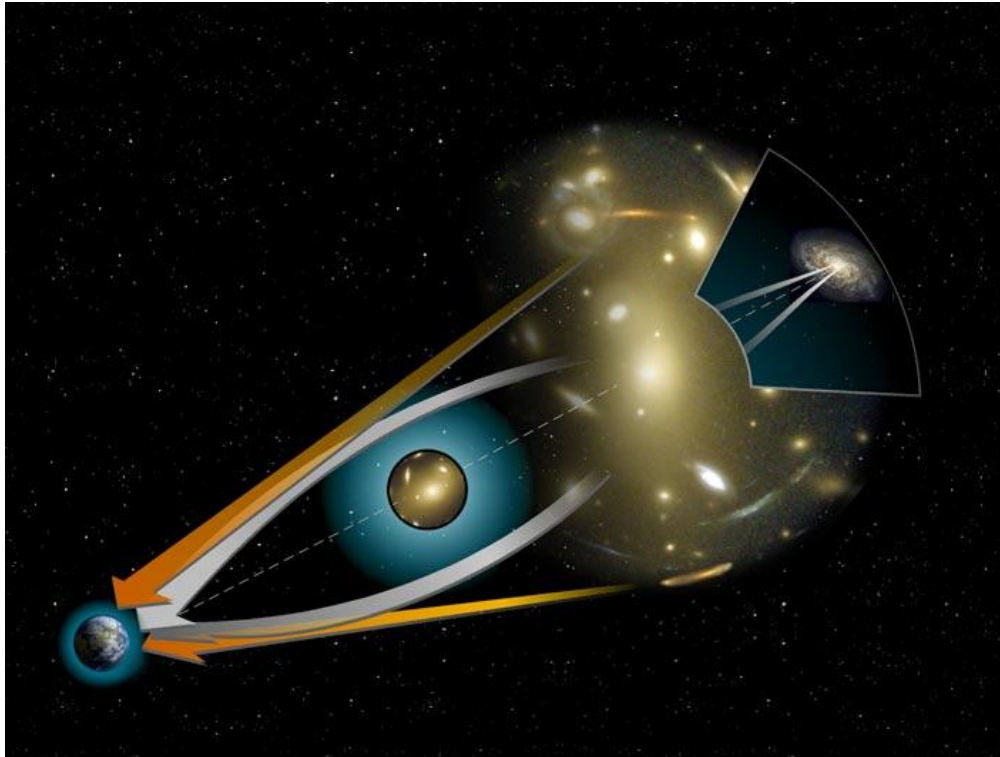


Illustration 1: How gravity bends light, causing gravitational lensing (NASA, 2005).

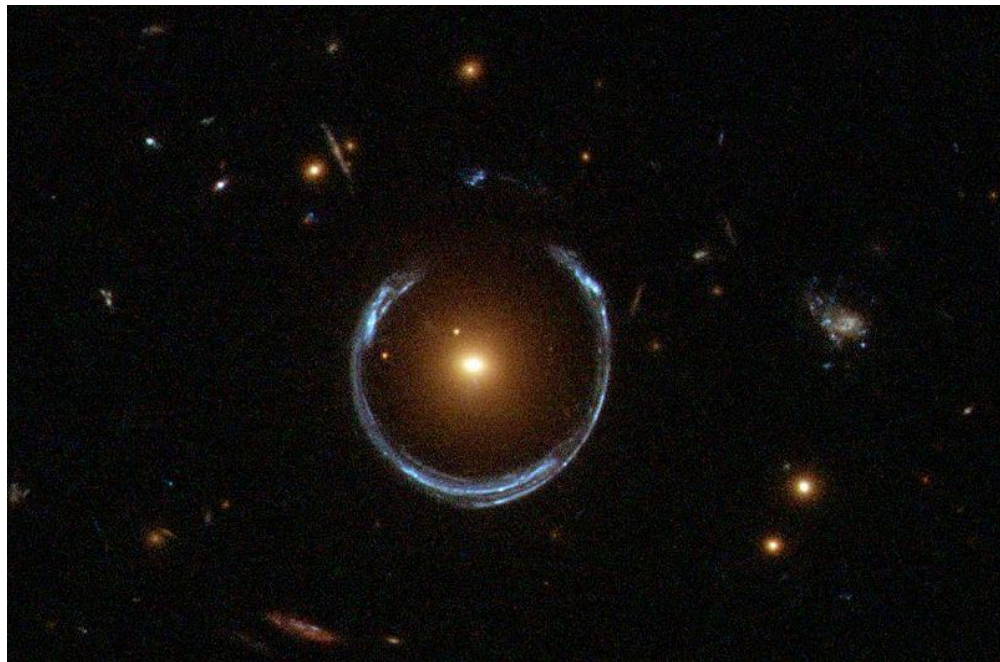


Figure 1: Image distortion caused by gravitational lensing (ESA/Hubble, 2011).

2 Project Description

2.1 Main Goals and Objectives

The main objective of Looking Mass is to provide open-source software that can artificially generate realistic gravitational lensing in images of space. Looking Mass aims to serve as a tool for researchers and other individuals interested in gravitational lensing as the application can assist in generating datasets of images with simulated gravitational lensing. Datasets created with Looking Mass can be used to develop methods of detecting gravitational lensing. For example, these datasets could be used to train a CNN (or other machine learning models) that can accurately detect whether an image contains gravitational lensing.

A secondary goal of the Looking Mass development team is to have the application be able to detect gravitational lensing in images using a CNN. This addition would be convenient to users of Looking Mass who are interested in detecting gravitational lensing as these users could use the detection component already built into Looking Mass. The team decided to prioritize generation over detection due to the lack of proper datasets to train a CNN, the team's lack of experience in artificial intelligence, and time constraints.

2.2 Main Functionality and Characteristics

The main functionality and characteristics of Looking Mass include a file explorer, a metadata entry system, an image processor, a CNN, and a graphical user interface (GUI). The file explorer allows the user to navigate through their computer's file system to choose read and write file paths. These paths determine where Looking Mass retrieves

the input image and where it saves the output image respectively. Metadata entry allows the user to enter metadata into Looking Mass that influences how the gravitational lensing is applied to the image. The image processor creates a copy of the image specified by the read path and applies gravitational lensing to the copied image based on user-entered metadata. The newly created image is then saved in the directory specified by the write path. The CNN accepts images and determines whether the images contain gravitational lensing. Finally, the GUI ties all the other components together into an interface that users can easily use to access the features of Looking Mass.

2.3 Intended Audience

The intended audience for Looking Mass is anyone that is interested in gravitational lensing. This categorization may include researchers that require datasets of gravitationally lensed images, instructors that need a tool to teach students about gravitational lensing, or astronomy enthusiasts who wish to learn more about how gravitational lensing works. The team aims for Looking Mass to be accessible to anyone interested in gravitational lensing with knowledge about the subject ranging from novice to expert. Additionally, because of the image manipulation and dataset generation features of Looking Mass, individuals interested in image processing and machine learning may also be interested in this project.

2.4 Technology Description

Looking Mass is written in Python, using Kivy for its GUI and OpenCV for its image processing. The team used GitHub for version control and chose Visual Studio Code as a development environment.

3 Significance

3.1 Why It's Worthwhile

Looking Mass is worthwhile as the number of images containing gravitational lensing is a relatively small dataset compared to what is ideally needed for machine learning applications. Looking Mass addresses this issue by allowing users to create realistic, artificial datasets of gravitationally lensed images, thus, realistic, artificial datasets large enough to satisfy the needs of these applications. The generation of such datasets means Looking Mass can be used as a research tool that can potentially assist with new scientific discoveries and breakthroughs. For example, manual detection of gravitational lensing can be time consuming, and datasets generated by Looking Mass can potentially help with methods that accelerate the detection process. Furthermore, Looking Mass can be used as an education tool to help instructors teach students about gravitational lensing.

3.2 Impact on Professional Growth

In terms of progressing professional growth, this project helped the team gain valuable experience in long-term software project development, managing a complicated repository of code, and learning new software in a relatively short amount of time. The project also helped the team develop important skills that are useful in industry such as image processing and user interface development. Finally, the project allowed the team to work with clients outside of computer science. More specifically, this project allowed the team to work on a product designed for individuals interested in gravitational lensing such as physicists and astronomy enthusiasts.

3.3 Similar Works

One similar product is GravLens3 which is a phone app that generates gravitationally lensed images (GravLens3, 2015). Though GravLens3 can generate images with simulated gravitational lensing, like Looking Mass, it is relatively simplistic and would not be suitable in generating large sets of data needed by researchers. Looking Mass addresses this issue by allowing users to customize the lensing generated in an accurate and realistic manner.

Another similar product is Hydralens: Gravitational Lens Model Generator which generates gravitational lens models for Lenstool, PixeLens, glafic and Lensmodel (Lefor, 2014). The application is relatively old, using Basic (an outdated programming language) with little documentation. Looking Mass addresses these issues by using a language that is more commonly used (Python) by current software developers with well documented code. Additionally, the open-source nature of Looking Mass allows users to modify the source code to suit individual needs.

Finally, one last similar product the team found is a research paper on the analysis of strong gravitational lenses with CNNs (Hezaveh, 2017). This product is like Looking Mass in that it uses a CNN to detect gravitational lensing in images. The team that worked on this paper had to access multiple databases to gather enough images for a dataset. An application like Looking Mass could address this issue by allowing researchers to generate their own datasets—hence avoiding the need to seek out images from multiple databases.

3.4 Project Impact and Context Considerations

The social impact of Looking Mass mostly centers around the scientific community—individuals with interests in gravitational lensing specifically. If the team’s project is successful enough to cause scientific discoveries or breakthroughs, its impact may be significant enough to benefit the society at large. In terms of environment impact, if the application becomes popular it may incentivize researchers to use more electronic rather than paper copies when analyzing gravitational lensing. Reduced paper use would reduce waste and would counteract deforestation.

4 Specification

4.1 Elicitation

To elicit the requirements for Looking Mass, the team conducted interviews of individuals with known or potential interest in the project. For this purpose, the team interviewed Dr. Melodi Rodrigue, Mr. Nikita Ignatyuk, and Mr. Abhijaat Sidher.

The team chose to interview Dr. Rodrigue because as a physics Ph.D and professor, she has knowledge in astronomy-based research. Additionally, Dr. Rodrigue is one of the external advisors of Looking Mass and can provide insight as to how the components of the project involving physics should be implemented. Dr. Rodrigue’s main concern was that some gravitational lenses can be potentially extremely dim and difficult to detect. To address this concern, the team made it a requirement that users can enter metadata into Looking Mass that influences the lensing generated. Dr. Rodrigue also stated that she

would prefer to use a tool like Looking Mass on Mac or Linux and that FITS is a preferred image format for astronomers.

Nikita Ignatyuk is a member of the development team and was interviewed because he is an undergraduate student who is majoring in physics in addition to computer science and engineering. This circumstance makes Mr. Ignatyuk a valuable interviewee as he can provide insights on both the physics and software aspects of Looking Mass. Mr. Ignatyuk stressed the importance of the application to be able to generate gravitational lensing realistically as datasets generated from Looking Mass must be as similar as possible to datasets containing images with real gravitational lensing. He also suggested that the software be open-source for it to be accessible to as many people as possible. When asked what operating system he would prefer to use the application on, Mr. Ignatyuk stated that he would prefer to use Windows. He also stated PNG was his file extension preference when viewing images.

Finally, Abhijaat Sidher is Aditya Sidher's brother (who is member of development team) and was interviewed because he is a former graduate student who has research experience. Mr. Sidher stated that the application should be able to accept most images regardless of size or format. He also recommended that the team implement a way for the user to use a file explorer for Looking Mass to input and output images. Like Mr. Ignatyuk, he has Windows as his preferred operating system and PNG as his preferred image file type. Mr. Sidher also suggested that compatibility with Unix-type operating systems may be better as they are free—making them more accessible.

4.2 Technical Requirements

The team developed non-functional requirements, shown in Table 1 and functional requirements, shown in Table 2, based on the information elicited from the interviews. Each requirement was given a priority (or tier) level that ranges from one to three. A priority level one requirement is one that was to be implemented by the end of the Spring 2020 semester. A priority level two requirement is one that was to be potentially implemented by the end of the Spring semester. A priority level three requirement is one that was not intended to be completed by the end of the Spring semester.

| Name | Tier | Description | |
|-------|------|--|--|
| NFR01 | [1] | Looking Mass shall have an intuitive user interface. | |
| NFR02 | [1] | Looking Mass shall accept the following file extensions: .jpg, .png, .pgm, and .ppm. | |
| NFR03 | [1] | Looking Mass shall be Linux compatible. | |
| NFR04 | [2] | Looking Mass shall be able to run on most home machines. | |
| NFR05 | [2] | Looking Mass shall be compatible with multiple operating systems. | |
| NFR06 | [3] | The CNN used for Looking Mass shall take 1 - 2 weeks of training time. | |

Table 1: Non-functional Requirements for Looking Mass.

| Name | Tier | Description |
|------|------|---|
| FR01 | [1] | Looking Mass shall have a user interface. |
| FR02 | [1] | Looking Mass shall have the option to generate a single image that simulates gravitational lensing. |
| FR03 | [1] | Looking Mass shall have output for when processing begins and ends. |
| FR04 | [1] | Looking mass shall allow the user to view the modified image. |
| FR05 | [1] | The user interface shall allow the user to exit the program, with a confirmation prompt to avoid lost work. |
| FR06 | [2] | Looking Mass shall have the option to process multiple images simultaneously. |
| FR07 | [2] | Looking Mass shall place processed images into a user specified folder. |
| FR08 | [2] | The user interface shall alert the user with a window upon completion, error, or exception of the image processor. |
| FR09 | [2] | Looking Mass shall have the option of producing a doctored image based on user specifications. |
| FR10 | [2] | If an image must be doctored, Looking Mass shall create and doctor a copy of the image, leaving the original image unaltered. |
| FR11 | [3] | Looking Mass shall detect if there is gravitational lensing in an image. |
| FR12 | [3] | The user interface shall allow the user to refer to an online image for processing. |
| FR13 | [3] | Looking Mass shall implement direct manipulation interface methods such as drag and drop. |
| FR14 | [3] | Looking Mass shall accept optional metadata prior to processing an image. |
| FR15 | [3] | The user interface shall allow user to stop processing for an image, in case of hang ups or long processing times. |
| FR16 | [3] | Looking Mass shall display the confidence score for each image it processes. |

Table 2: Functional Requirements for Looking Mass.

4.3 Use Cases

The team also developed use cases, shown in Figure 2, that model ways in which can users interact with Looking Mass based on the requirements elicited. A detailed description of each use case is given in Table 3 with templates for Use Case 4, Use Case 5, and Use Case 8 provided in Table 4, Table 5, and Table 6 respectively.

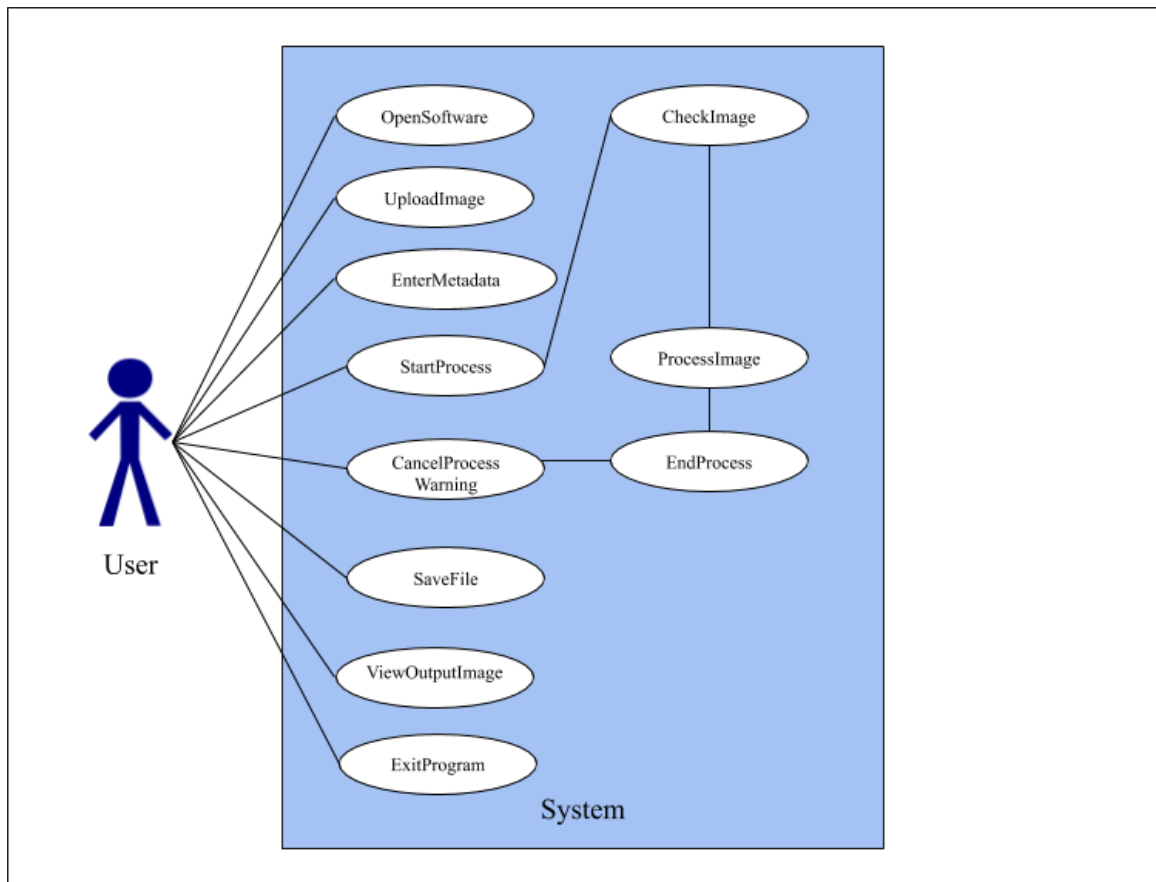


Figure 2: Use Case Diagram for Looking Mass.

| | | |
|------|----------------------|---|
| UC01 | OpenSoftware | The user can open Looking Mass by using the desktop icon, by accessing it through an applications menu, or by running the executable directly from its file location. |
| UC02 | UploadImage | The user can upload images to the program that they would like the program to process. |
| UC03 | EnterMetadata | The user can enter additional metadata for an image and specify how the program should process the image based on the entered metadata. |
| UC04 | StartProcess | The application will start processing the image after upload, when the process button is pressed, and will begin queuing logic for processing. |
| UC05 | CancelProcessWarning | The user can cancel processing at any time, in case of error or long processing times and it'll prompt a warning asking user to continue. |
| UC06 | EndProcess | The processing will end, allowing for the user to see the outputted results in the UI, as well as trace down any issues that may have occurred. |
| UC07 | ViewOutputImage | The user will be able to view the output image, with coordinates of any gravitational lensing detected. |
| UC08 | SaveFile | The image outputted will be able to be saved to the user's hard drive, so that they may access it again in the future or use it for reference. |
| UC09 | ExitProgram | The user can exit the program completely by using the close button on the top right of the application. |
| UC10 | ProcessImage | The user can click process image and the program will analyze and process the image. |
| UC11 | CheckImage | The system will be able to check whether an image uploaded by the user is valid and can be processed by Looking Mass. |

Table 3: Detailed Use Cases for Looking Mass.

| User Case: StartProcess | |
|-------------------------|--|
| Use Case ID | UC04 |
| Actor | User |
| Precondition(s) | <ol style="list-style-type: none"> 1. User has opened the software. 2. User has uploaded an Image. 3. User has entered metadata. |
| Flow of Events | <ol style="list-style-type: none"> 1. User grabs mouse. 2. User clicks process image. 3. User sees window that image processing has begun. 4. User can see that image processing can be cancelled. |
| Postcondition(s) | User has started the image processing. |

Table 4: Template for Use Case 4.

| User Case: CancelProcessWarning | |
|---------------------------------|---|
| Use Case ID | UC05 |
| Actor | User |
| Precondition(s) | <ol style="list-style-type: none"> 1. An image is being processed. |
| Flow of Events | <ol style="list-style-type: none"> 1. The user requests the software to cancel processing an image. <ol style="list-style-type: none"> 2a. Ask the user for confirmation. 2b. The user clicks "yes": continue. 2c. The user clicks "no": exit this use case without cancelling the image processing. 3. Tell the worker thread processing the image to release all associated memory. 4. Delete the process worked on by the worker thread. 5. Update the status of images processed. |
| Postcondition(s) | One less image is being processed. |

Table 5: Template for Use Case 5.

| User Case: SaveFile | |
|---------------------|---|
| Use Case ID | UC08 |
| Actor | User |
| Precondition(s) | 1. There is a processed image to be saved. |
| Flow of Events | <p>1. The user requests the processed image to be saved in a target directory with their chosen name.</p> <p>2a. Check the target directory.</p> <p>2b. If the directory is write-protected, display an error and exit this use case without saving.</p> <p>2c. If a file with the requested name already exists, ask for confirmation to overwrite.</p> <p>2d. If the file can be saved with the requested name in the target directory, proceed.</p> <p>3a. Ask the user for confirmation.</p> <p>3b. The user clicks "okay": save.</p> <p>3c. The user clicks "no": exit this use case without saving.</p> |
| Postcondition(s) | The target directory has a file with the specified name with the saved processed image/data. |

Table 6: Template for Use Case 8.

4.4 Requirement Traceability Matrix

The team also created a requirement traceability matrix, shown in Figure 3, that maps the functional requirements of Looking Mass to its use cases.

| | UC 01 | UC 02 | UC 03 | UC 04 | UC 05 | UC 06 | UC 07 | UC 08 | UC 09 | UC 10 | UC 11 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| FR 01 | X | | | | | | | | | X | |
| FR 02 | | | | X | | | | X | | X | X |
| FR 03 | | | | | | | | | | X | |
| FR 04 | | X | | | | | X | | | | |
| FR 05 | X | | | | X | X | | | | | |
| FR 06 | | | | X | | | X | | | X | |
| FR 07 | | | | | | | | X | | X | |
| FR 08 | | | | | | | | | | | |
| FR 09 | | | | | X | X | | | | X | X |
| FR 10 | | | | | | | X | X | | X | |
| FR 11 | | | | | | | | | | X | |
| FR 12 | | | | | | | | | | X | |
| FR 13 | | X | | | | | | | | | |
| FR 14 | | | | | | | | | | X | |
| FR 15 | | | X | | | | | X | | | X |
| FR 16 | | | | | X | X | | | X | | |

Figure 3: Requirement Traceability Matrix for Looking Mass.

5 Design

5.1 System-Level Diagram

The context model of Looking Mass, see Figure 4, consists of the main components Looking Mass system which are the GUI, the CNN, the File I/O components, the image processor, and the system that generates gravitational lensing in images (subsystem of the image processor).

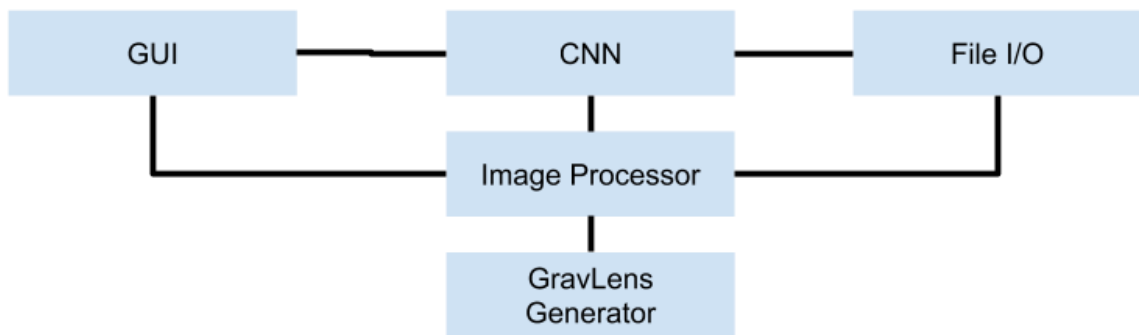


Figure 4: Context Model of Looking Mass.

5.1.1 GUI

This component has the classes and methods necessary to implement the GUI. The GUI is the part of the application visible to the user. It initiates most actions within the application as well as allows oversight and management of ongoing actions.

5.1.2 CNN

This component has the classes and methods necessary for implementing a CNN. The CNN can detect whether there is gravitational lensing in an image with a minimal rate of error and sends this information to the main Looking Mass system. Additionally, it should be able to accurately find additional information about the image if metadata is provided.

5.1.3 File I/O

File I/O is composed of classes that can read in the information carried by images of file types and convert them to the appropriate one needed for image processing or the CNN. The information read in is then stored in a database that is readily available for the CNN to use for determining whether an input file has gravitational lensing.

The composition of the output file database is a new image which has a separate name such as *imageName_output.filetype* giving the location of gravitational lensing through the image if it occurs or none if it doesn't.

5.1.4 Image Processor

The image processor takes in images either from the CNN or the File I/O and applies image processing. The data shared is directly used from the image input from the File I/O which can then be used by the CNN for further training, GravLens Generator for gravitational lensing application, or the GUI for displaying the image.

5.1.5 GravLens Generator

The GravLens Generator is composed of the classes and methods necessary to simulate the effects of gravitational lensing in an image. This component takes an image provided by the user (through the GUI) and generates a modified image that simulates gravitational lensing with user-specified or randomized parameters governing the position and strength of gravitational lensing in the image.

5.2 Program Units

Figure 5 shows the class diagram for the project. The class diagram consists of 12 classes and 39 methods. Table 7 gives a more detailed description of each class used in the project.

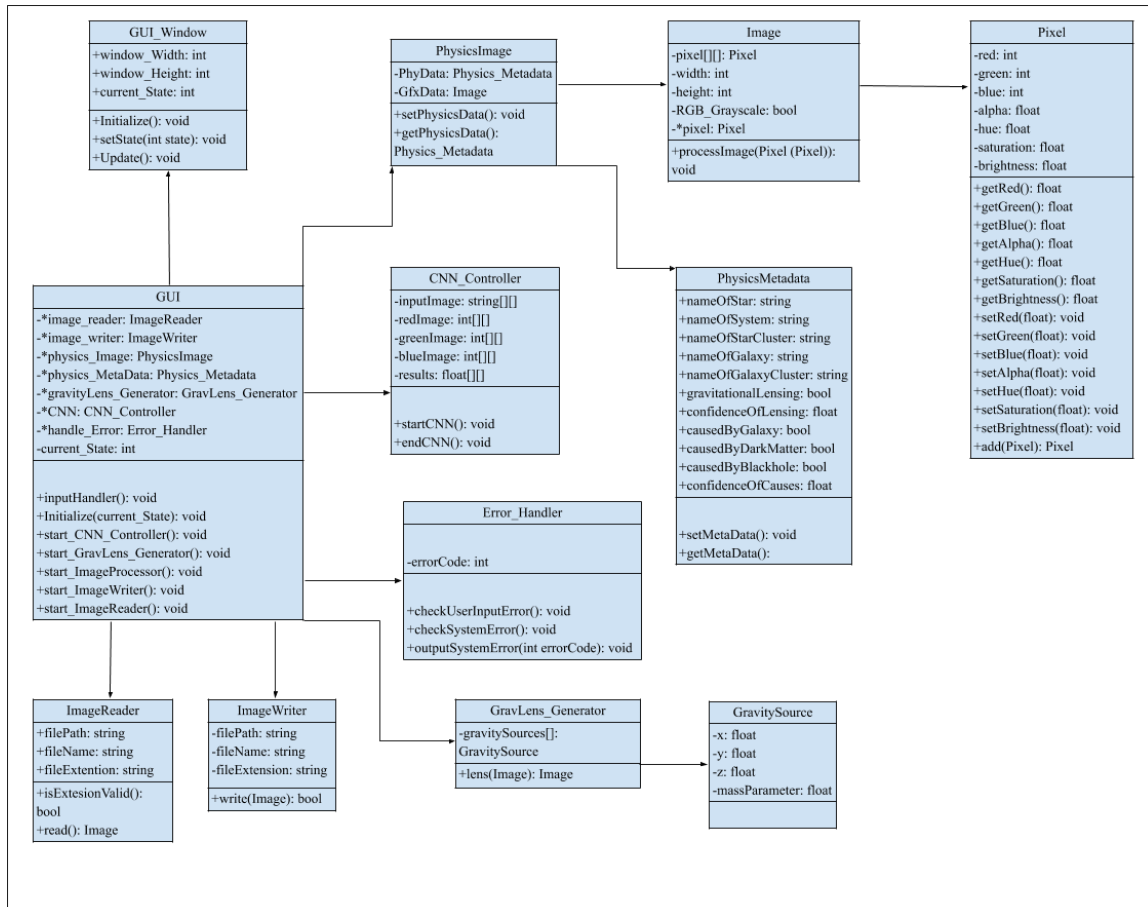


Figure 5: Class Diagram for Looking Mass.

| Class | Methods | Purpose |
|-----------------|---|---|
| ImageReader | isExtensionValid() checks to see if the file extension for the selected image is valid. read() reads in the selected file and store it into the image class. | ImageReader retrieves from the GUI the file path of the input image and stores the location in case other images are loaded in at a different time. |
| ImageWriter | write() writes the contents of the Image class into a user specified output directory. | ImageWriter writes to the specified directory given by the user and store this same location for later use. |
| PhysicsImage | getPhysicsData() retrieves data from the PhysicsMetadata and Image class. setPhysicsData() stores data into the PhysicsMetadata and Image class. | PhysicsImage is a wrapper for the image class that stores the metadata and graphics data for the input and output images. It works off the image output from the CNN and image processor. |
| PhysicsMetadata | setMetaData() retrieves data from the PhysicsMetadata class. getMetaData() stores data into the PhysicsMetadata class. | PhysicsMetadata gets and sets physics information from the user. It is also be helpful in determining and storing the information displayed in the image, e.g. name of star system, galaxy. |
| Image | processImage() uses the image processor to process the image. | Image applies processing on the image and hold the original image information as well as output the new image once processed. |

Table 7: Description of the classes of Looking Mass.

| Class | Methods | Purpose |
|--------------------|--|--|
| Pixel | <p>getRed () gets the red color value of the pixel.</p> <p>getGreen () gets the green color value of the pixel.</p> <p>getBlue () gets the blue color value of the pixel.</p> <p>getAlpha () gets the alpha value of the pixel.</p> <p>getHue () gets the hue of the pixel.</p> <p>getSaturation () gets the saturation of the pixel.</p> <p>getBrightness () gets the brightness of the pixel.</p> <p>setRed () sets the red color value of the pixel.</p> <p>setGreen () sets the green color value of the pixel.</p> <p>setBlue () sets the blue color value of the pixel.</p> <p>setAlpha () sets the alpha value of the pixel.</p> <p>setHue () sets the hue of the pixel.</p> <p>setSaturation () sets the saturation of the pixel.</p> <p>setBrightness () sets the brightness of an image.</p> <p>add () adds pixels.</p> | Pixel stores RGBA values in float format, between 0.0 and 1.0, and allows access in alternate formats. |
| GravLens_Generator | <p>lens () uses the image processor to simulate gravitational lensing on an image</p> | GravLens_Generator stores gravitational sources and allows application onto given images. |

Table 7 (continued).

| Class | Methods | Purpose |
|----------------|--|---|
| GravitySource | No Methods. | GravitySource stores the position of gravitational sources in x, y, z coordinates ranging from 0 to 1. |
| GUI | <p>inputHandler() detects and handles user input.</p> <p>Initialize() initializes the GUI.</p> <p>start_CNN_Controller() starts the CNN Controller.</p> <p>start_GravLens_Generator() starts the Gravitational Lensing Generator.</p> <p>start_ImageProcessor() starts the Image Processor.</p> <p>start_ImageWriter() starts the Image Writer.</p> <p>start_ImageReader() starts the Image Reader.</p> | GUI oversees all other processes. It determines how the user can work through each function in a simple and easy to use manner. |
| GUI_Window | <p>Initialize() initializes the GUI window.</p> <p>setState() sets the stat of the GUI window.</p> <p>Update() updates the GUI window.</p> | GUI_Window displays the actual window the user will see when using the Looking Mass GUI. |
| CNN_Controller | <p>startCNN() starts the CNN.</p> <p>endCNN() stops the CNN.</p> | CNN_Controller oversees the main process for how the CNN starts, pauses, and stops. It can be controlled through the GUI. |

Table 7 (continued).

| Class | Methods | Purpose |
|---------------|---|---|
| Error_Handler | checkUserInputError () checks for errors caused by user input. checkSystemError () checks for errors made by the system. outputSystemError () outputs the error that has occurred. | Error_Handler gives the user information on what the problem may be if they run into any. |

Table 7 (continued).

5.3 Main Data Structures

Although the CNN portion of the project was not completed due to most detection requirements being of third priority level, Table 8 shows what the application would need to store the CNN parameters that allow it to detect gravitational lensing. Once the CNN is trained, it saves its state to an external file. From this point, the CNN loads its state from that file.

| CNN_Controller | |
|---------------------------|--|
| Keys | Values |
| inputImage: string[][] | The input image will use the array of strings to find the file path for training and applying its filters for finding gravitational lensing. |
| redImage: int[][] | The redImage pertains to the red pixel values for the image for when the CNN stacks them for training. |
| greenImage: int[][] | The greenImage pertains to the green pixel values for the image for when the CNN stacks them for training. |
| blueImage: int[][] | The blueImage pertains to the blue pixel values for the image for when the CNN stacks them for training. |
| results: float[][] | The results will be returned from the CNN and holds the information needed for displaying to the GPU and reading out to the Image Processor. |

Table 8: Data structures used for CNN_Controller.

Table 9 shows how the physics metadata is stored in a data structure with the individual variables all being a part of the class method structure. The data structure is mainly to help keep track of the information stored within each image to tell them apart from one another.

The Physics Metadata also helps store any additional information from the image.

| Physics Metadata | |
|-----------------------------|--|
| Keys | Values |
| nameOfStar: string | Holds the name and information for the observed image star in a class struct. |
| nameOfSystem: string | Holds the name and information for the observed image system in a class struct. |
| nameOfStarCluster: string | Holds the name and information for the observed star cluster in a class struct. |
| nameOfGalaxy: string | Holds the name and information for the observed image in Galaxy in a class struct. |
| nameOfGalaxyCluster: string | Holds the name and information for the observed image in a Galaxy Cluster. |
| gravitationalLensing: bool | Holds the value if gravitational lensing is detected in the image. (true/false) |
| confidenceOfLensing: float | Holds the value of the confidence of gravitational lensing detected in the image. |
| causedByGalaxy: bool | Holds the value of if gravitational lensing was flagged due to a galaxy. |
| causedByDarkMatter: bool | Holds the value of if gravitational lensing was flagged due to dark matter. |
| causedByBlackhole: bool | Holds the value of if gravitational lensing was flagged due to a blackhole. |
| confidenceOfCauses: float | Holds the value of how confident the CNN is in its analysis. |

Table 9: Data structures used for PhysicsMetadata.

5.4 Detailed Design

Figure 6 shows how the user would use Looking Mass to analyze an image and see whether the image has gravitational lensing. During this process the user has the option of entering additional physics metadata which can help the CNN analyze the image and may lead to the system generating additional output such as a doctored version of the image that contains useful information.

Figure 7 shows how a user can generate images by clicking on the Process Image menu and where they are asked for an input image. The Generate New Image functionality is used for creating new distortions in the image for training and letting the user see how the CNN was trained.

Figure 8 shows that when a CNN is loaded, the `StartCNN()` function is called. The `StartCNN()` function begins by checking to determine if the input image is a valid image, based on file types and size. This image is then loaded into an array based on its RGB values. The `StartCNN()` function is then called again, and checked to see if all files have been loaded. If not, it loops through the function until `isLoading` returns that all images have been properly handled. Then the `endCNN()` function is called to end the loop.

Figure 9 shows that when the user requests an image to be processed, it must first be loaded into an internal representation, which is when `ImageReader::read()` is called, where the instance whose method is called already has the path, filename and extension assigned. The function loads the requested file, extracts the appropriate dimensions of the image, then reads in and converts the pixels of the image into the internal representation, and returns

the Image object it has created. It throws an error if an unsupported type of file is loaded, or any other unexpected action occurs.

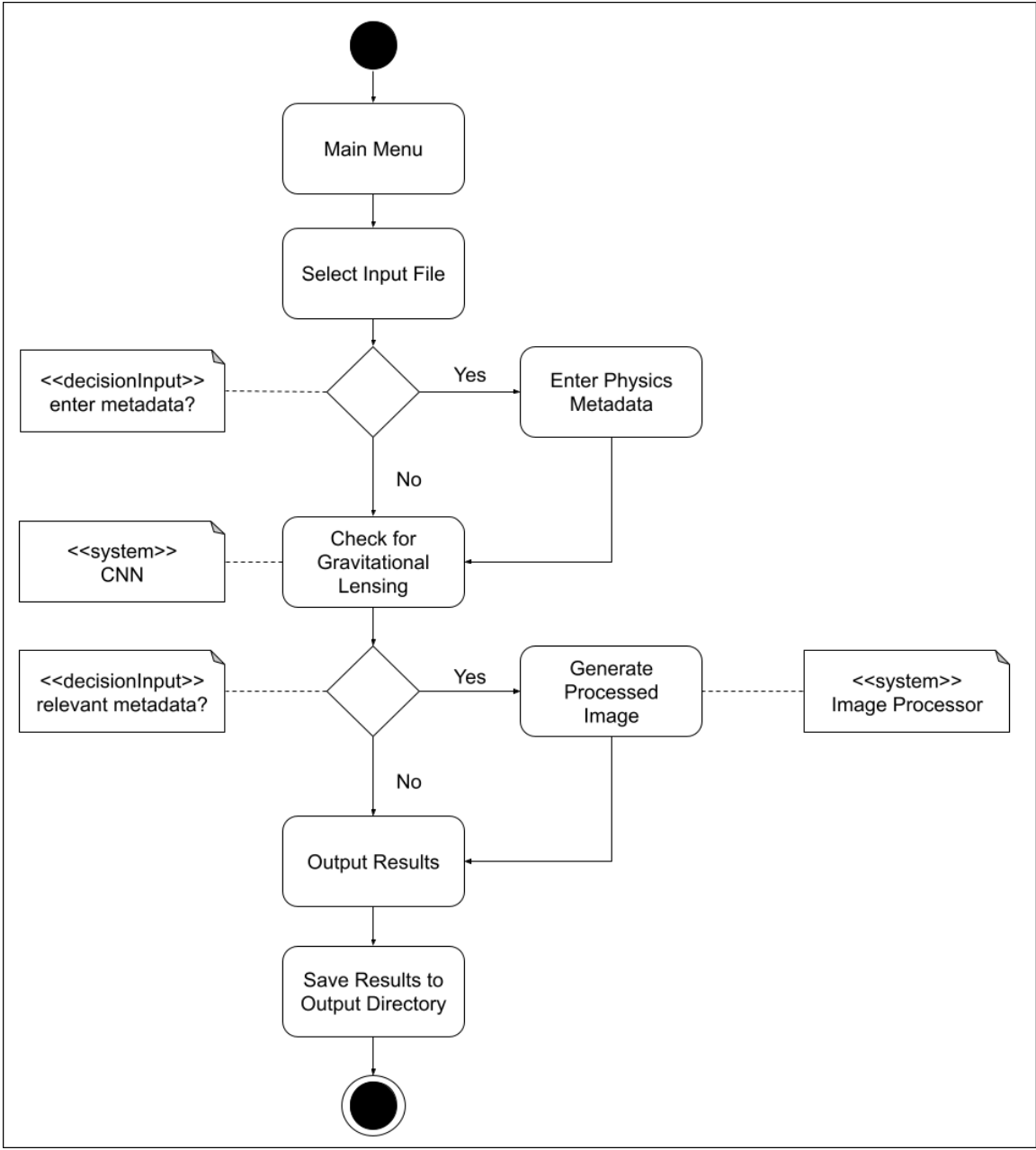


Figure 6: Activity diagram of analyze an image for gravitational lensing.

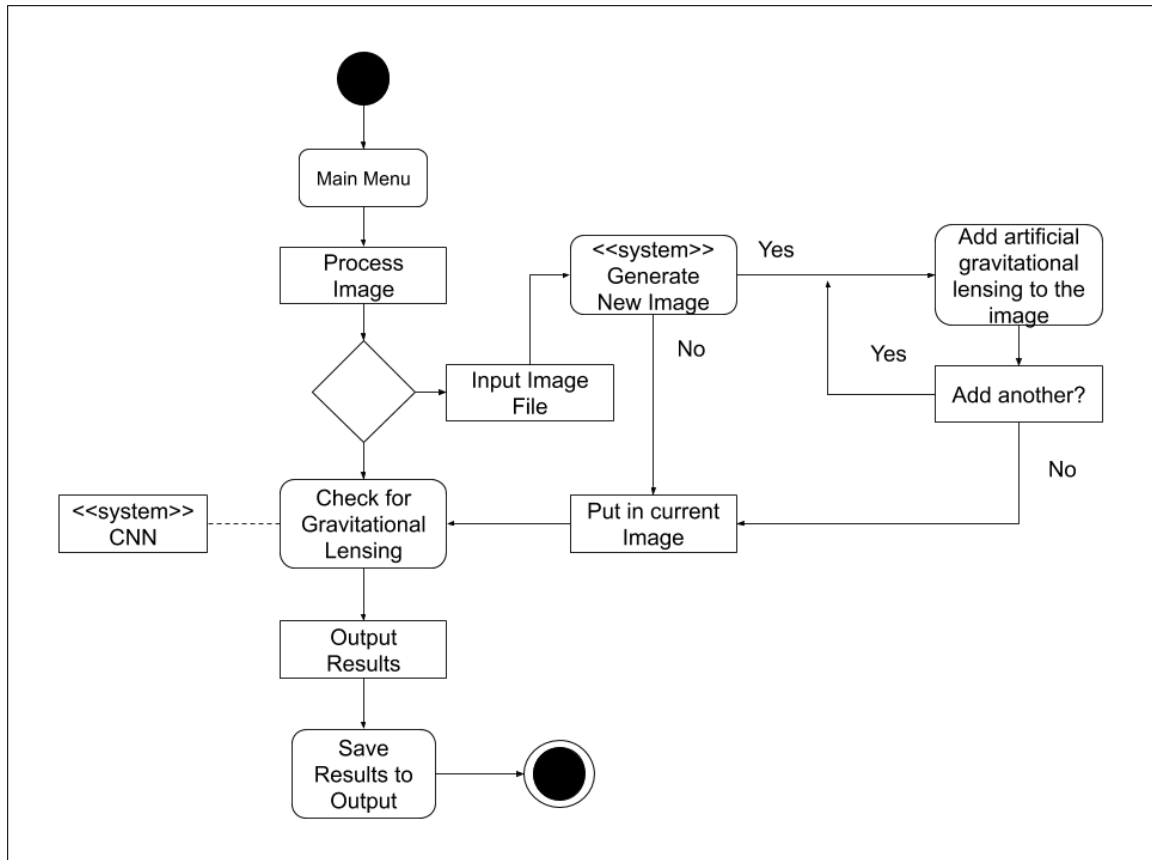


Figure 7: Activity diagram of generated images being fed to the CNN.

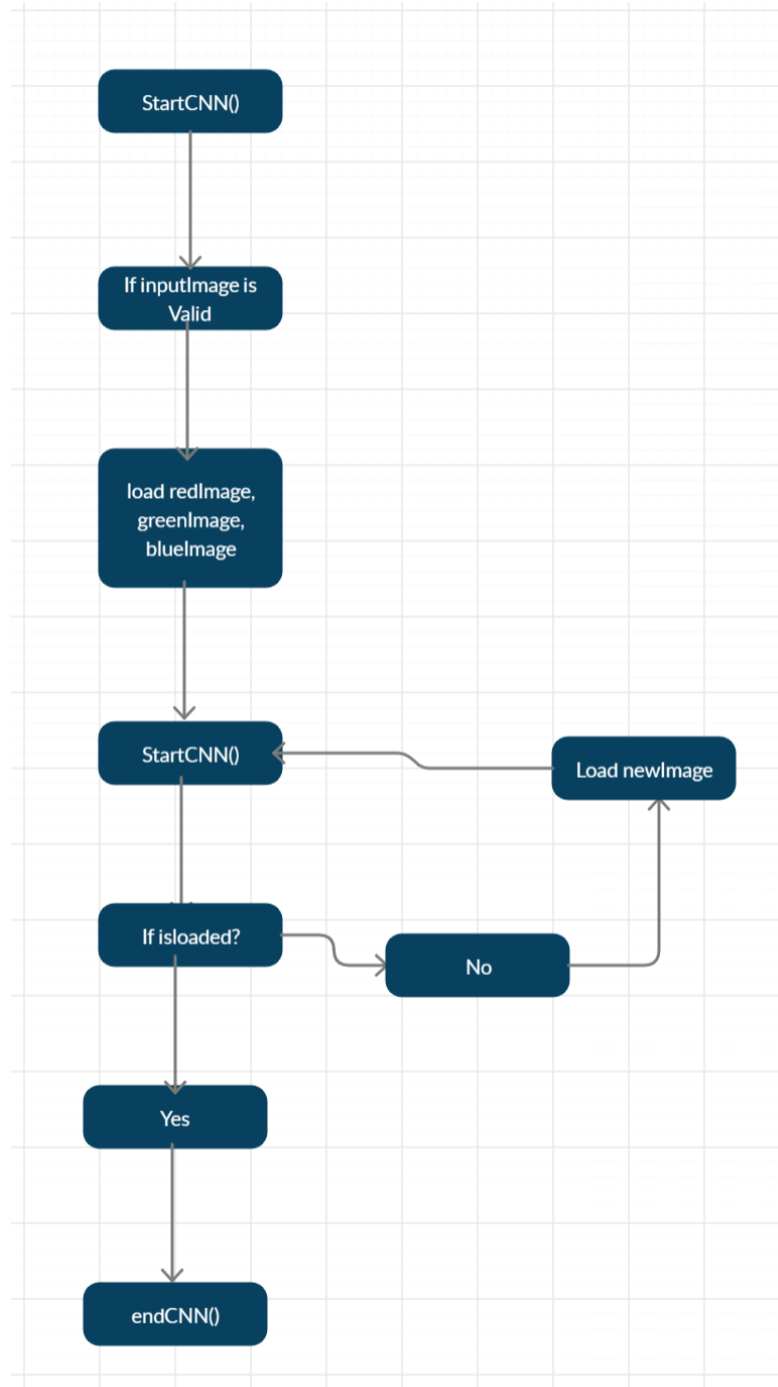


Figure 8: Flowchart representing the StartCNN() function.

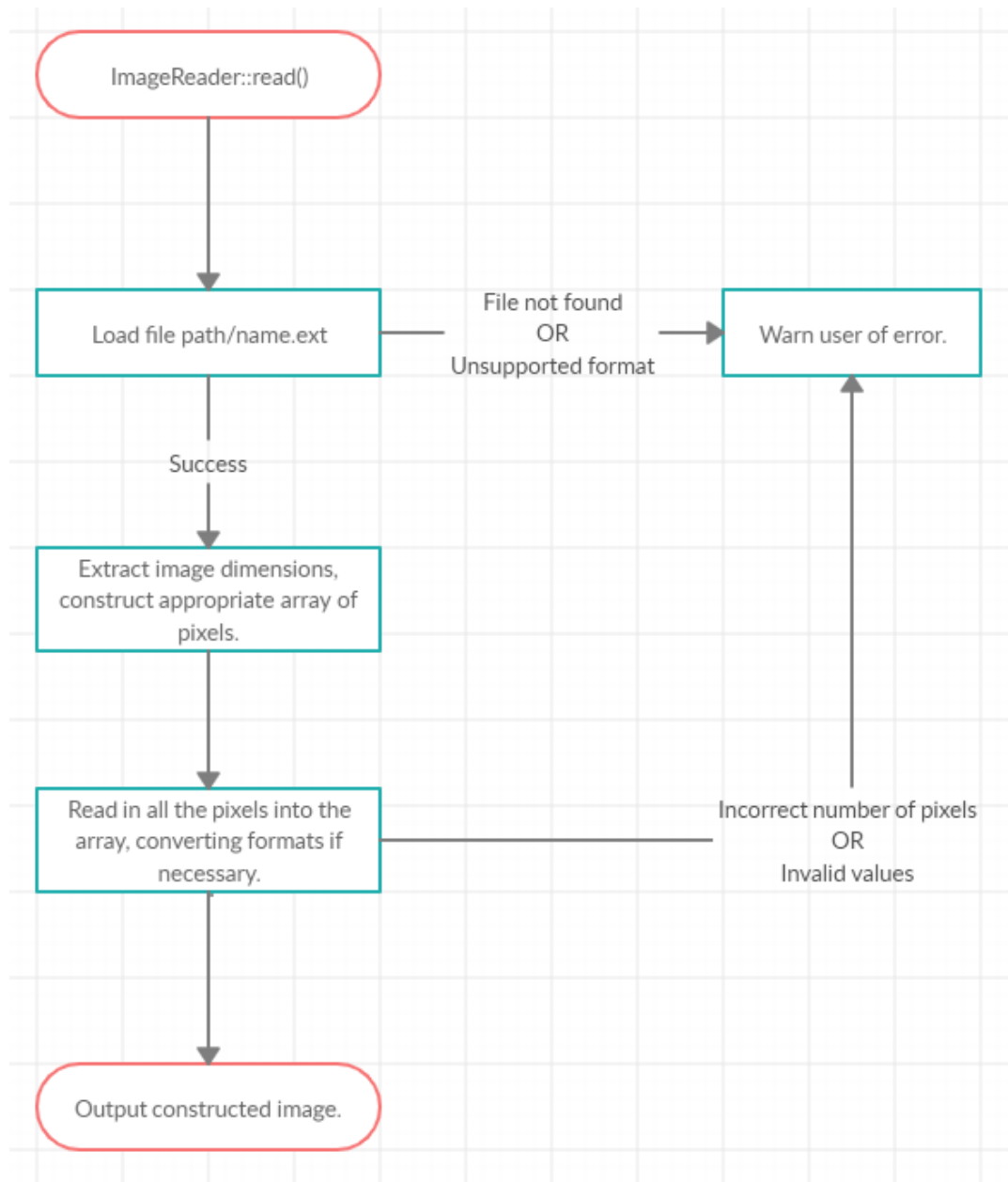


Figure 9: Flowchart representing the `ImageReader::read()` function.

6 Implementation

6.1 Project Summary

6.1.1 Functionality Implemented

Lists of use cases, functional requirements, and non-functional requirements implemented are shown in Table 10, Table 11, and Table 12 respectively. As shown by the tables, most requirements and use cases were implemented.

| | | |
|------|---------------|---|
| UC01 | OpenSoftware | The user can open Looking Mass by using the desktop icon, by accessing it through an applications menu, or by running the executable directly from its file location. |
| UC02 | UploadImage | The user can upload images to the program that they would like the program to process. |
| UC03 | EnterMetadata | The user can enter additional metadata for an image and specify how the program should process the image based on the entered metadata. |
| UC04 | StartProcess | The application will start processing the image after upload, when the process button is pressed, and will begin queuing logic for processing. |
| UC06 | EndProcess | The processing will end, allowing for the user to see the outputted results in the UI, as well as trace down any issues that may have occurred. |
| UC08 | SaveFile | The image outputted will be able to be saved to the user's hard drive, so that they may access it again in the future or use it for reference. |
| UC09 | ExitProgram | The user can exit the program completely by using the close button on the top right of the application. |
| UC10 | ProcessImage | The user can click process image and the program will analyze and process the image. |
| UC11 | CheckImage | The system will be able to check whether an image uploaded by the user is valid and can be processed by Looking Mass. |

Table 10: Use cases implemented.

| Name | Tier | Description |
|------|------|---|
| FR01 | [1] | Looking Mass shall have a user interface. |
| FR02 | [1] | Looking Mass shall have the option to generate a single image that simulates gravitational lensing. |
| FR03 | [1] | Looking Mass shall have output for when processing begins and ends. |
| FR04 | [1] | Looking mass shall allow the user to view the modified image. |
| FR05 | [1] | The user interface shall allow the user to exit the program, with a confirmation prompt to avoid lost work. |
| FR06 | [2] | Looking Mass shall have the option to process multiple images simultaneously. |
| FR07 | [2] | Looking Mass shall place processed images into a user specified folder. |
| FR08 | [2] | The user interface shall alert the user with a window upon completion, error, or exception of the image processor. |
| FR09 | [2] | Looking Mass shall have the option of producing a doctored image based on user specifications. |
| FR10 | [2] | If an image must be doctored, Looking Mass shall create and doctor a copy of the image, leaving the original image unaltered. |
| FR14 | [3] | Looking Mass shall accept optional metadata prior to processing an image. |

Table 11: Functional requirements implemented.

| Name | Tier | Description |
|-------|------|---|
| NFR01 | [1] | Looking Mass shall have an intuitive user interface. |
| NFR03 | [1] | Looking Mass shall be Linux compatible. |
| NFR04 | [2] | Looking Mass shall be able to run on most home machines. |
| NFR05 | [2] | Looking Mass shall be compatible with multiple operating systems. |

Table 12: Non-functional requirements implemented.

6.1.2 Functionality Not Implemented

Lists of use cases, functional requirements, and non-functional requirements not implemented are shown in Table 13, Table 14, and Table 15 respectively. Although a priority level one non-functional requirement, the team was not able to complete Non-functional Requirement 02 as Looking Mass cannot accept images with .ppm and .pgm file extensions. This failure was caused because images with these file extensions would require the development of separate image processing algorithms and lensing calculations that the team could not implement within the allotted time. In other words, the team prioritized developing software that would work for the other file extensions. All other requirements that were not implemented are of the third priority level i.e. requirements that the team did not expect to have completed by the end of the Spring 2020 semester. Use Case 05 was not implemented as it was not a critical use case required for completion of Looking Mass. The time that would have been spent implementing this use case was used to ensure more important use cases were implemented. Use Case 07 was only implemented partially as the team did not focus on implementing gravitational lensing detection as stated in previous sections.

| | | |
|------|----------------------|---|
| UC05 | CancelProcessWarning | The user can cancel processing at any time, in case of error or long processing times and it will prompt a warning asking user to continue. |
| UC07 | ViewOutputImage | The user will be able to view the output image, with coordinates of any gravitational lensing detected. |

Table 13: Use cases not implemented.

| Name | Tier | Description |
|------|------|--|
| FR11 | [3] | Looking Mass shall detect if there is gravitational lensing in an image. |
| FR12 | [3] | The user interface shall allow the user to refer to an online image for processing. |
| FR13 | [3] | Looking Mass shall implement direct manipulation interface methods such as drag and drop. |
| FR15 | [3] | The user interface shall allow user to stop processing for an image, in case of hang ups or long processing times. |
| FR16 | [3] | Looking Mass shall display the confidence score for each image it processes. |

Table 14: Functional requirements not implemented.

| Name | Tier | Description |
|-------|------|--|
| NFR02 | [1] | Looking Mass shall accept the following file extensions: .jpg, .png, .pgm, and .ppm. |
| NFR06 | [3] | The CNN used for Looking Mass shall take 1 - 2 weeks of training time. |

Table 15: Non-functional requirements not implemented.

6.1.3 Contributions of Team Members

The contributions of each team member are as follows:

Marcus Casey

- File I/O
- Research / Discovery of main platforms used
- Pair programming UI elements and app integration
- Code documentation
- Bug fixes
- Approximately 800 lines contributed after refactoring
- Approximately 3 hours every week since start of project

Aditya Sidher

- Metadata entry and use
- GUI, applying gravitational lensing using coordinates picked by user
- Code documentation
- GUI exit error
- Code refactoring
- Bug fixes
- Approximately 4 hours every week since start of project
- Approximately 1000 lines contributed after refactoring

Nikita Ignatyuk

- Image processing algorithm and gravitational lensing code
- Scientific verification of numbers and units
- Error handling
- Code refactoring
- Code documentation
- Approximately 800 lines contributed after refactoring
- Approximately 3.5 hours every week since start of project

Ethan Park

- GUI, developed basis for pop up system and widget templates
- Code documentation
- Batch processing for images
- Code refactoring
- Bug fixes
- Approximately 1000 lines contributed after refactoring
- Approximately 4 hours every week since start of project

6.2 User Interface Snapshots

Figure 10 shows the main menu of Looking Mass. In the main menu the user can set the read and write paths for images, enter in metadata to influence how lensing is applied, choose where on the image lensing is applied, and process the image.

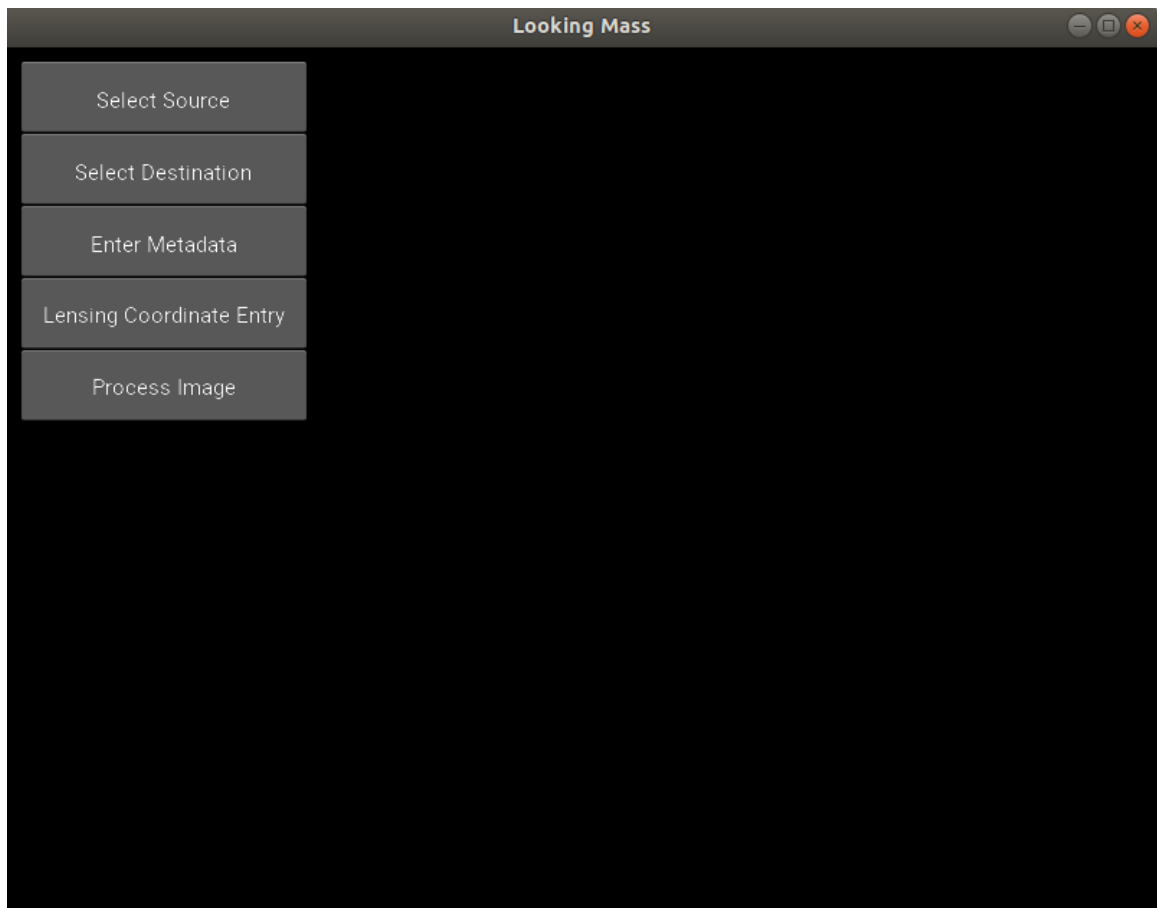


Figure 10: Main Menu of Looking Mass.

Figure 11 shows the pop up that appears when the “Select Source” button is clicked. The pop up displays a file explorer that allows the user to choose an input image from the computer’s file system.

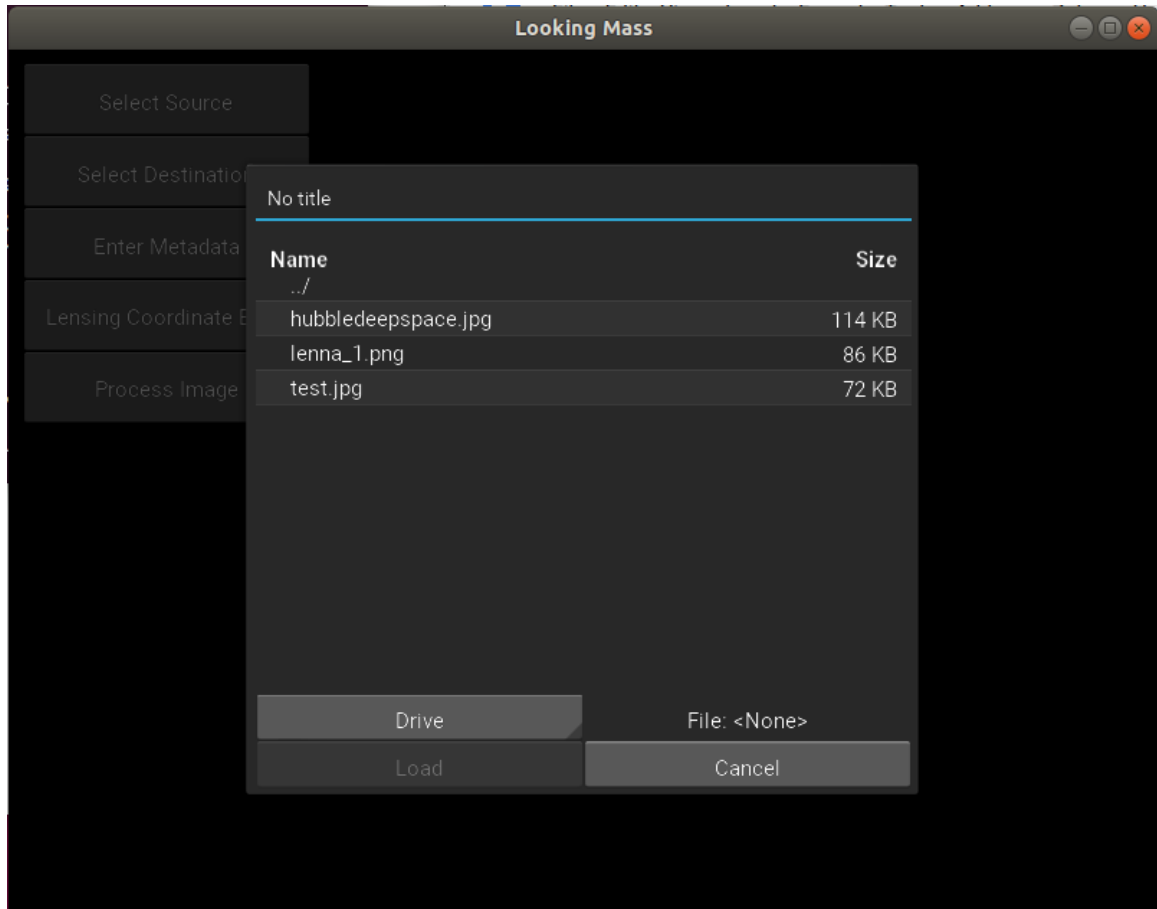


Figure 11: Input File Selection System of Looking Mass.

Figure 12 shows that after the input image is selected, it will be displayed at the top right of the application.

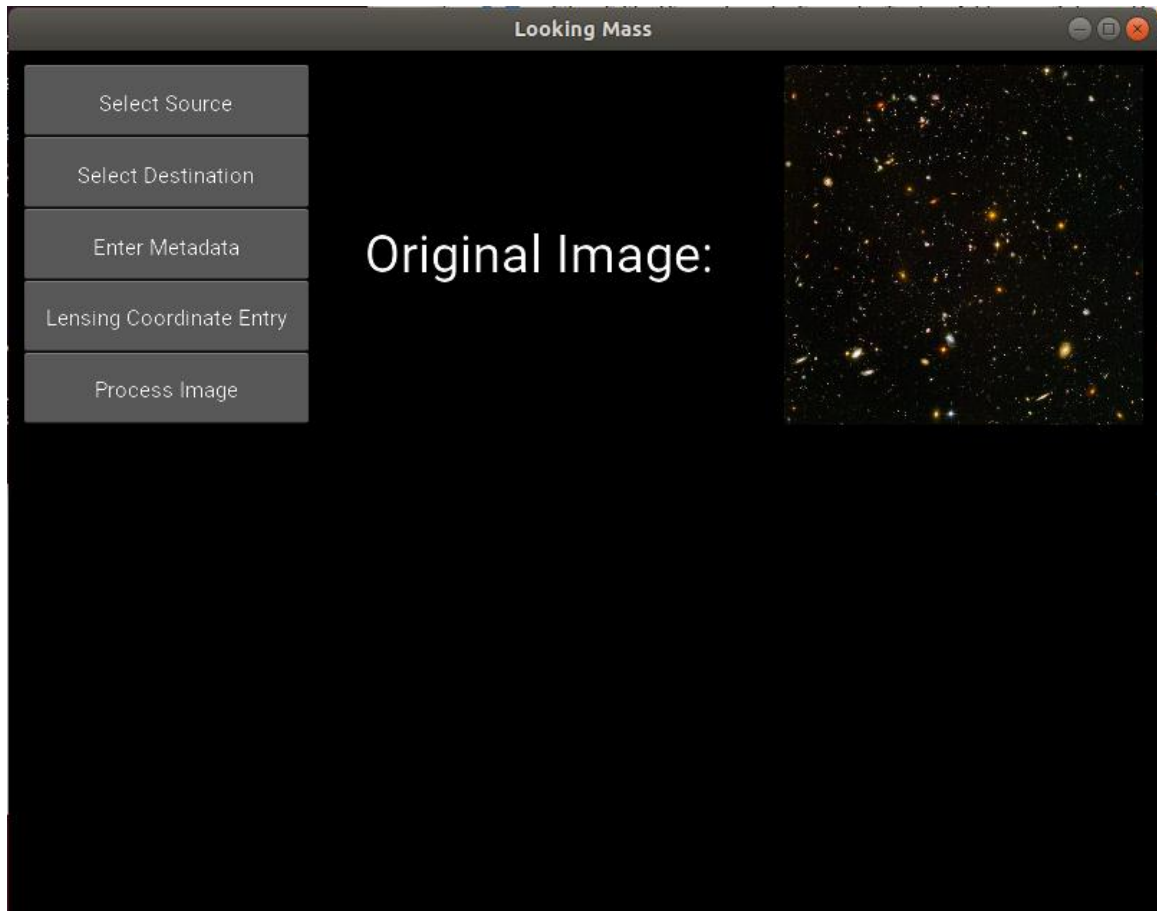


Figure 12: A source image displayed in Looking Mass.

Figure 13 shows the pop up that appears when the “Select Destination” button is clicked. The pop up displays a file explorer that allows the user to choose a name and location for the output image.

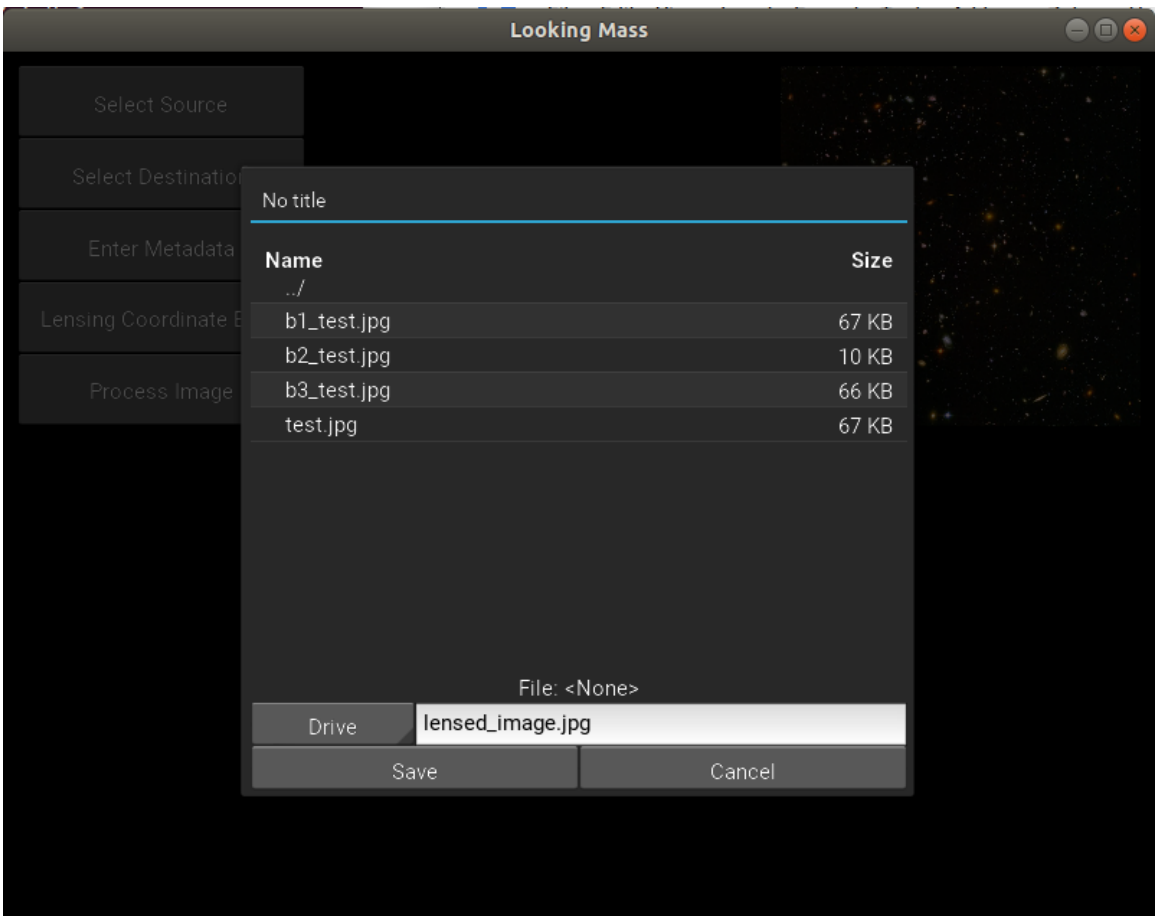


Figure 13: Output File Selection System of Looking Mass.

Figure 14 shows the pop up that appears when the “Enter Metadata” button is clicked. In this pop up the user can change the first four parameters to influence how gravitational lensing will be applied onto input images. If switch at the bottom of the pop up is on, Looking Mass will also process all other images in the input image’s directory.

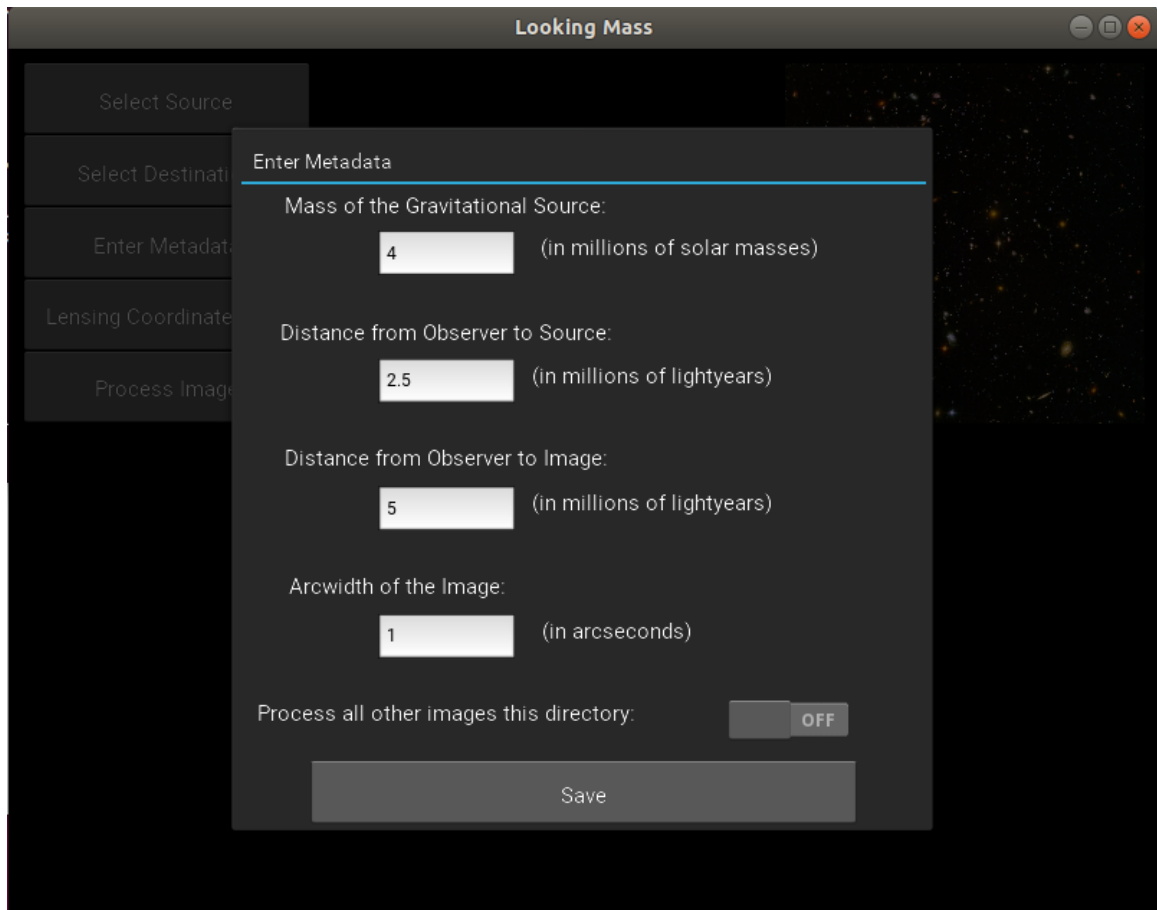


Figure 14: Metadata Entry System for Looking Mass.

Figure 15 shows the pop up that appears when the “Lensing Coordinate Entry” button is clicked. In this pop up the user can select the area where lensing should be applied using the mouse cursor. The coordinates on the bottom right of the pop up will update based on where the user clicks.

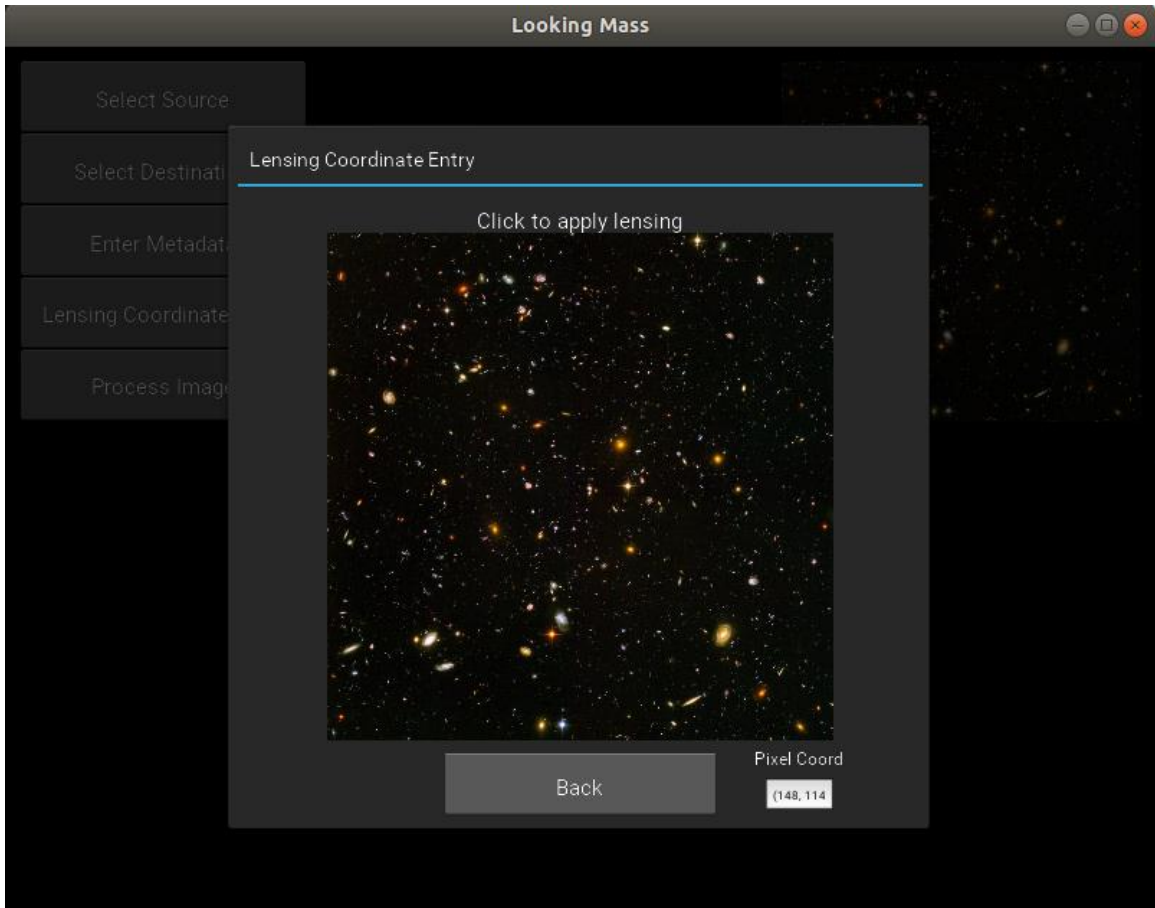


Figure 15: Coordinate Selection System of Looking Mass.

Figure 16 shows the pop up that appears when the “Process Image” button is clicked. This pop up notifies the user that Looking Mass has started lensing generation process.

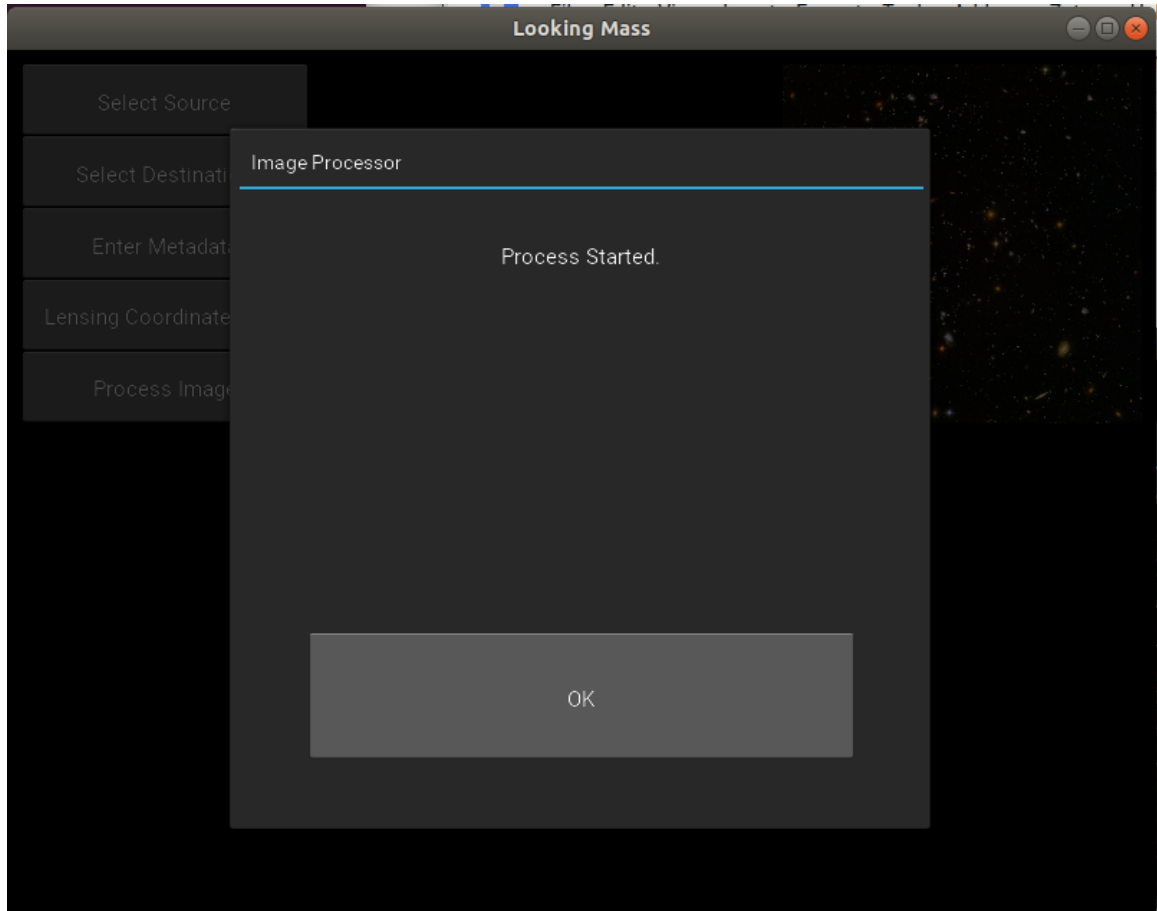


Figure 16: Pop up indicating image processing has started.

Figure 17 shows that once the new image with applied lensing has been generated, Looking Mass will notify the user that the application has finished processing the image. If images are being processed in a batch, the pop up will show after all images have been processed.

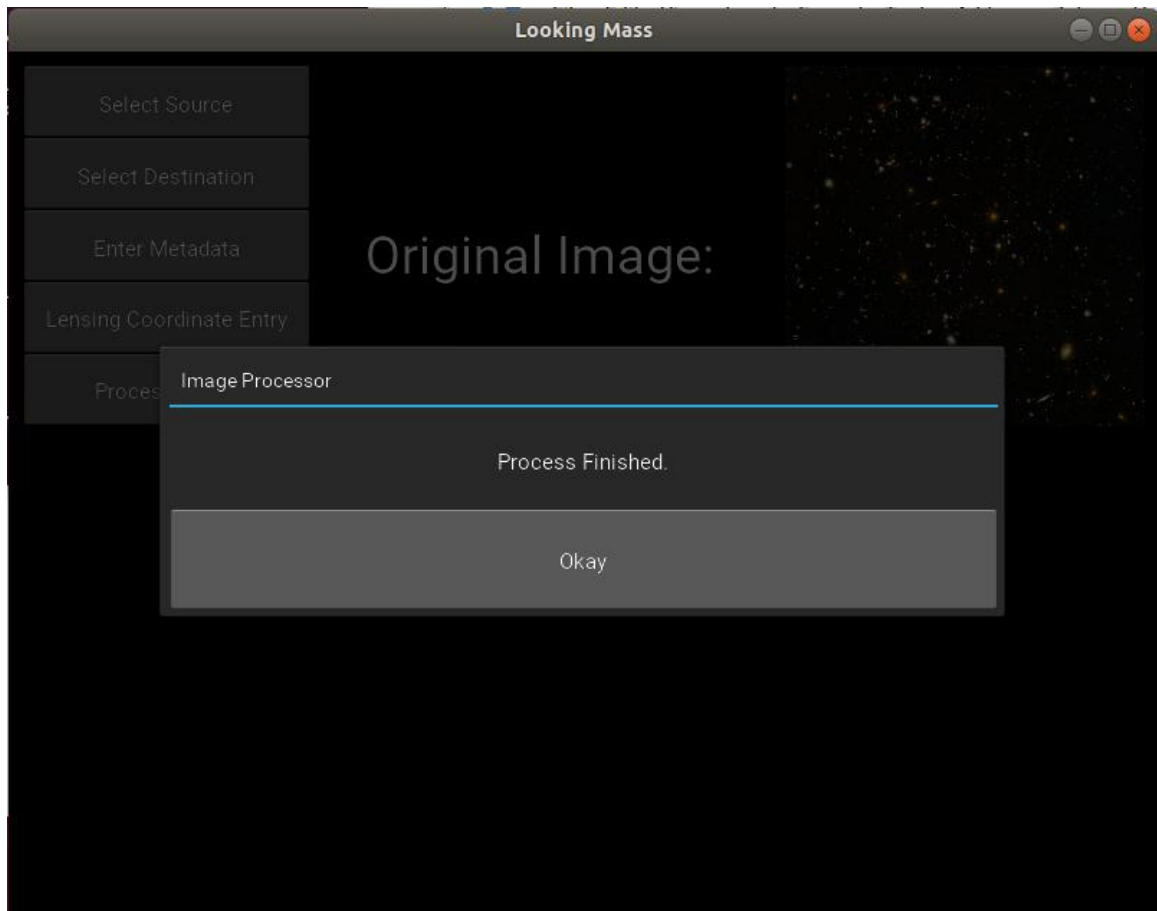


Figure 17: Pop up indicating image processing has ended.

Figure 18 shows that once the new image has been generated, it will be displayed at the bottom right of the application.

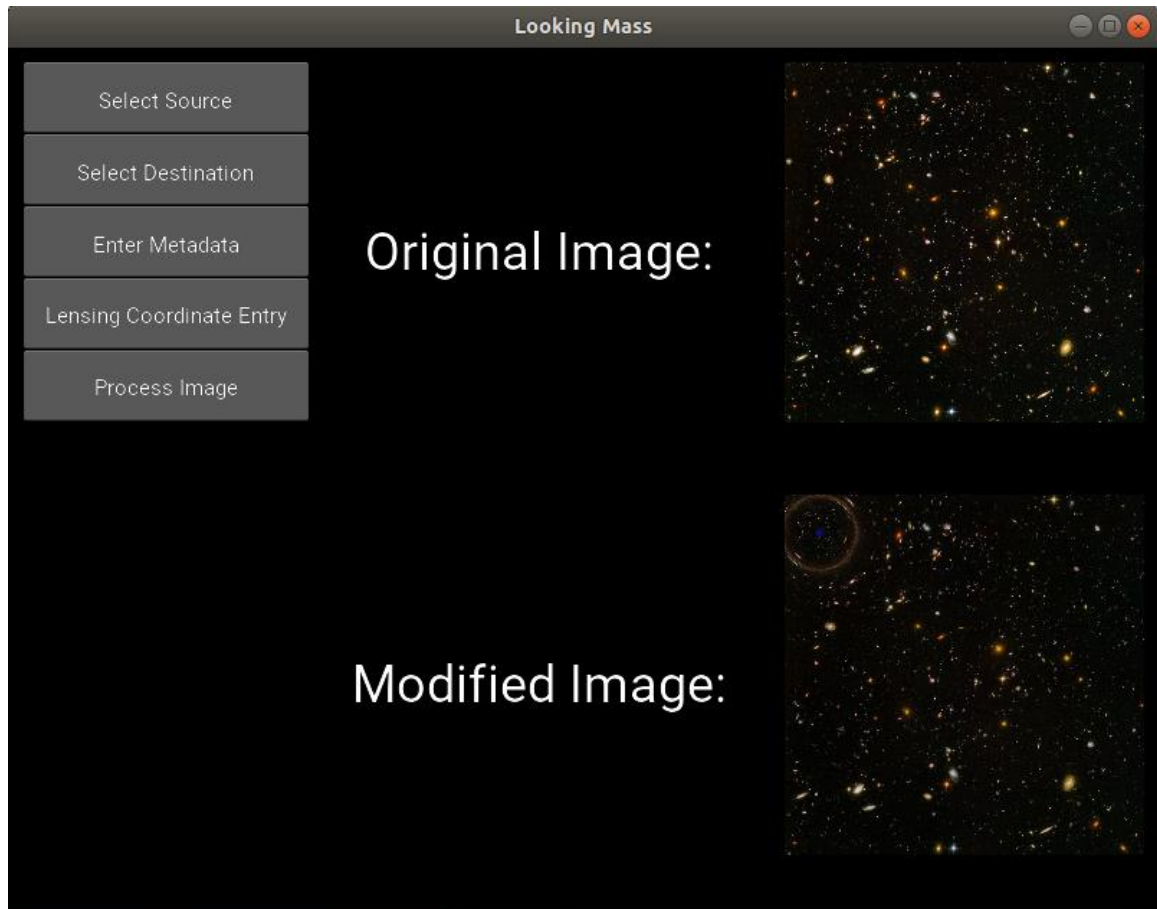


Figure 18: A destination image displayed in Looking Mass.

Figure 19 shows that Looking Mass saved the output image to output directory specified by the image and that the image can be opened.

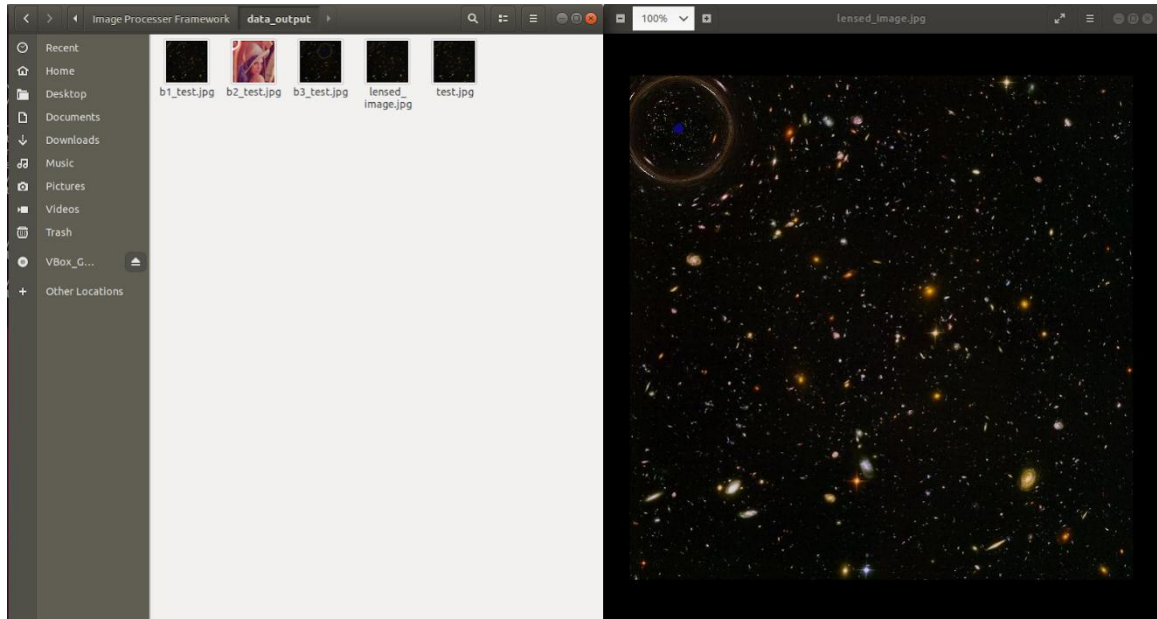


Figure 19: Image generated by Looking Mass in computer's file system.

7 Future Work

One avenue of potential future work could be completing the rest of the requirements and use cases. Use cases and requirements that the team expected to complete by the end of the Spring 2020 semester should take precedence over other tasks. These tasks would involve further developing Looking Mas such that it can accept more types of file extensions for images and can cancel image processing. Upon completion of these tasks, the team could work on the lensing detection component of the application. Such a component would be useful as users would not have to seek out or implement methods for detection and could instead directly feed processed images into the CNN of Looking Mass.

Although not explicitly outlined during the specification phase of the project, the team agreed that there were many features of Looking Mass that could be improved. One area that the team felt could be improved was how metadata was entered. The team originally wanted the application to be as accessible as possible. The current metadata entry system contradicts this goal as it requires a fair amount of knowledge about gravitational lensing to achieve a desired lensing effect. To remedy this issue, the team could expand upon the system, implementing basic and advanced options. On one hand, basic options would allow more novice users to apply lensing presets in an easy to understand manner. On the other hand, advanced options would allow more experienced users to fine tune the lensing to precise specifications. Additionally, the team would like to spend more time developing a more robust error handling system.

A more ambitious project the team thought of pursuing was the creation a virtual reality (VR) counterpart to Looking Mass as several members of the development team have experience in computer graphics and digital interactive game design. The application would stitch images of space together to create a virtual environment that resembles a star map projection room. Users would be able to apply gravitational lensing to the star map in real time using hands-on VR controls.

8 Conclusion

By creating an application that generates images with realistic gravitational lensing, the Looking Mass development team has created a tool that individuals can use to create large datasets of gravitationally lensed images. The datasets generated from Looking Mass have the potential to create new opportunities for researchers that require large datasets for

experiments and studies. Ideally, the datasets generated from Looking Mass can be used to develop new, faster methods of detecting gravitational lensing. The development of such methods has the potential to facilitate research on the sources of gravitational lensing such as black holes and dark matter.

Bibliography

ESA/Hubble, & NASA. (2011). *A Horseshoe Einstein Ring from Hubble* [Photograph].

Retrieved from

https://commons.wikimedia.org/wiki/File:A_Horseshoe_Einstein_Ring_from_Hubble.JPG

Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing*. New York: Pearson.

GravLens3. (2015.). Retrieved October 10, 2019, from App Store website:

<https://apps.apple.com/ca/app/gravlens3/id318275930>

Hezaveh, Y. D., Levasseur, L. P., & Marshall, P. J. (2017). Fast automated analysis of strong gravitational lenses with convolutional neural networks. *Nature*.

<https://doi.org/10.1038/nature23463>

Lefor, A. (2014). *HydraLens: Gravitational lens model generator*. Astrophysics Source Code Library. <https://doi.org/10.20356/C4301H>

Maoz, D. (2016). *Astrophysics in a nutshell*. Princeton: Princeton University Press.

Moore, T. A. (2013). *A general relativity workbook*. Mill Valley (Calif.): University Science Books.

NASA. (2005). *Gravitational lens-full* [Photograph]. Retrieved from

https://en.wikipedia.org/wiki/File:Gravitational_lens-full.jpg

Appendix A: Glossary of Terms

A

Application (App): Software designed to help a user perform a group of coordinated functions, tasks, or activities.

Artificial Intelligence: The study of “intelligent agents” or in other words systems that perceive their environments and take actions that maximize the chances of achieving their goals.

B

Black hole: A massive object whose gravity caused it to collapse into an incredibly small space. They are dense enough to visibly alter the path of light, and anything -- including light -- passing near it gets sucked in by the black hole’s gravity, thus it appears black.

C

Code: The set of instructions that form a computer program that can be executed by a computer.

Computer Graphics: The study of digitally synthesizing and manipulating visual content.

Convolutional Neural Network (CNN): A class of deep neural network commonly used to analyze visual imagery. Deep neural networks are artificial neural networks with multiple layers between input and output layers. Artificial neural networks are computing systems that are inspired by biological neural networks and learn to perform tasks by considering examples.

D

Dark Matter: A form of matter that is hypothesized to exist in space. It could take many forms including weakly interacting particles (or cold dark matter) and high-energy randomly moving particles created after the Big Bang.

Direct Manipulation: A human-computer interaction style in which performing actions correspond to the manipulation of physical objects. An example of direct manipulation would be using the mouse to drag and drop a file into a folder.

F

File Extension: The group of letters appearing after the dot on computer files. File extensions indicate the format of the file.

G

Gravitational lensing: The effect of gravity bending the path of light, as described by General Relativity. This allows one to see an object geometrically hidden behind another object.

General Relativity: A prominent theory in physics, first proposed by Einstein, that describes gravity as the curvature of spacetime instead of as a force.

Graphical User Interface (GUI): A visual way of interacting with a computer using items such as windows, icons, and menus, used by most modern operating systems.

I

Image Processing: The process of modifying and extracting data from images.

I/O: Input output, for every input in a computer, there is an output. Usually represented in typing on a keyboard and text appearing on a screen.

M

Machine Learning: The study of algorithms and statistical models that computer systems use to perform specific tasks without explicitly being programmed to perform these tasks.

Metadata: A set of data that provides information about other data.

O

Open-source Software: A type of computer software in which the source code is released under a license. Depending on the license the copyright holder may grant users the right to study, change, or distribute the software.

Operating System: Software involved with the basic functions of a computer such as task scheduling, executing applications, and controlling peripherals. Common operating systems are Windows, macOS, and Linux.

P

Program: A collection of instructions that performs a specific task when executed by a computer.

R

RGB/RGBA: Stands for Red Green Blue (Alpha). Red, Green, and Blue refers to a system for representing colors used on a computer display, while Alpha represents transparency.

S

Software: A collection of data or computer instructions that a computer uses to function.

Spacetime: Space (3D) and time (1D) combined into a single coordinate system (4D). This has particular uses in General Relativity, where all four coordinates may interact with each other on equal footing.

String: Strings are objects that represent sequences of characters in programming languages.

U

User Interface (UI): In terms of human-computer interaction, user interfaces are how users and machines interact.

W

Wrapper: Referring to when a class ‘wraps around’ a resource, to provide a different interface than the object it is wrapping.