Final Report

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

June 1, 2017

Senior Capstone, Oregon State University

Abstract

This document contains the final report summarizing the *North American Eclipse 2017* senior capstone project at Oregon State University. This project is part of the larger Eclipse Megamovie Project. The Eclipse Megamovie project is a citizen science project centered around the eclipse that will pass over the United States on August 21st, 2017. It is a collaboration between Google and scientists from UC Berkeley and several other institutions with the aim of compiling a large dataset of eclipse observations. Acquiring coronal data is of particular interest as the corona is not normally visible from Earth. Specifically, the project will crowdsource photos of the eclipse from photographers at various locations along the path of totality. These images will be aligned spatially and temporally and stitched into a unique movie that shows the eclipse over a period of 1.5 hours as it passes across the United States. Additionally, the complete photo dataset will be open sourced so that independent researchers may do their own analysis.

David Konerding, Project Sponsor	Date
Bret Lorimore	Date
George Harder	Date
Jacob Fenger	Date

TABLE OF CONTENTS

1 Introduction

Bret interned at Google during the summer of 2016 where he worked on parts of the Eclipse Megamovie Project. He asked his manager at the time, David Konerding, to sponsor a capstone team to assist with the project. Bret requested that the team consist of himself, George Harder, and Jacob Fenger. Since Bret was familiar with the project from his internship, he quickly molded into the technical expert of the group. All of us worked on technical portions of the project as well as ensuring the writing portions of the project were as good as possible. David Konerding advised us on the development of the simulator, processor, and developer pipeline while he was working on other aspects of the Eclipse Megamovie Project.

2 Requirements

2.1 Original Requirements Document

Requirements

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

November 4th, 2016

CS 461 - Fall 2016

Abstract

On August 21, 2017 a total solar eclipse will pass over the United States. The path of totality will stretch from Oregon to South Carolina. There has not been a total solar eclipse like this, crossing the country from coast to coast, since the eclipse of 1918. The Eclipse Megamovie Project is a collaboration between Google and scientists from UC Berkeley and several other institutions with the aim of compiling a large dataset of eclipse observations. Acquiring coronal data is of particular interest as the corona is not normally visible from Earth. Specifically, the project will crowdsource photos of the eclipse from photographers at various locations along the path of totality. These images will be aligned spatially and temporally and stitched into a unique movie that shows the eclipse over a period of 1.5 hours as it passes across the United States. Additionally, the complete photo dataset will be open sourced so that independent researchers may do their own analysis.

Google will contribute applications providing, among other things, backend image processing, photo upload capabilities, and static informational content. This senior capstone project will consist of two distinct sub-projects, specifically, improving/implementing an image processing algorithm facilitating the classification and alignment of solar eclipse images before they are stitched into a movie, and a location-based eclipse simulator.

David Konerding	11/4/2016
David Konerding, Project Sponsor	Date
Bret Lorimore	Date
George Harder	Date
Jacob Fenger	Date

TABLE OF CONTENTS

1 Introduction

1.1 Purpose

The purpose of this software requirements specification (SRS) is to describe in detail the Eclipse Image Processor and Eclipse Simulator that our group will produce. By writing these requirements down and agreeing to them with our sponsor both parties will have a clear understanding of what the finished product will be and what it will be able to do. The intended audience for this SRS is our sponsor, the Senior Capstone Instruction Team, and ourselves.

1.2 Scope

We are producing two products: an Eclipse Image Processor and an Eclipse Simulator. The Eclipse Image Processor will take images, find the Sun and the Moon in the images, crop the image around these bodies, and extract information about the relative position of the Sun and the Moon in these images. The Eclipse Simulator will provide users with a 2D visual representation of the eclipse from a specified location from 12 hours before it occurs to 12 hours after.

1.3 Definitions, acronyms, and abbreviations

Eclipse Megamovie Project: The Eclipse Megamovie Project is a collaboration between Google and scientists from Berkeley and several other institutions with the aim of collecting large quantities of observations of the solar eclipse that will pass over the United States on August 21, 2017. The project will crowdsource photos of the eclipse from photographers at various points along the path of totality.

EXIF: EXIF refers to the Exchangeable Image File Format, a standard media file format. Ancillary data tags associated with other media files are frequently referred to as ”EXIF fields”, ”EXIF data”, etc. This is how ”EXIF” will be used within the scope of this document.

GPS: GPS stands for global positioning system. We will refer to the latitude and longitude information obtained from a global positioning system as ”GPS coordinates”, ”GPS data”, etc. throughout this document.

JPEG/JPG: JPEG is a lossy compression technique for images. When we refer to JPEG/JPG files in this document we are referring to image files compressed in this method with the .jpeg or .jpg file extension.

PNG: PNG refers to the Portable Network Graphics image file format. Images in the PNG format are frequently referred to as ”PNGs” and are saved with the .png file extension.

1.4 References

This SRS makes reference to a report by Larisza D. Krista and Scott W. McIntosh titled ”The Standardisation and Sequencing of Solar Eclipse Images for the Eclipse Megamovie Project.” This technical report was produced as a

collaboration between scientists at the University of Colorado at Boulder, the National Center for Atmospheric Research and the National Oceanic and Atmospheric Administration. This paper can be found on arxiv.org.

1.5 Overview

The remainder of this SRS contains an overall description of the Eclipse Image Processor and Eclipse Simulator systems in section 2. Following these descriptions are specific requirements for the systems in section 3.

2 Overall Description

2.1 Product perspective

1) These products, the Eclipse Image Processor and the Eclipse Simulator, are both components of the larger Eclipse Megamovie Project. They are designed to operate as wholly independent modules that can be ”plugged into” the existing Eclipse Megamovie codebase. The Eclipse Image Processor will be a binary that receives images from another application, processes them, and exports them. The Eclipse Simulator will be a standalone JavaScript module that can be added to an existing webpage.
2) The Eclipse Image Processor does not directly interface with the user. It resides behind the front-end of the Eclipse Megamovie website. The Eclipse Simulator does interface directly with the user. It should function on most modern internet browsers (Chrome, Firefox, Safari). The simulator will appear to the user as a 2D animated depiction of the Sun and the Moon as they appear at the specified time and location. The simulator will also have background imagery in addition to the Sun and the Moon. Besides the images, the simulator will have a time slider, a location input, and a time display.
3) This system does not interface with hardware.
4) The Eclipse Image Processor will be designed to work on Ubuntu 16.04. It is necessary for the image processor to work on this operating system because the machines that will be running the processor use Ubuntu 16.04. The Eclipse Simulator is a JavaScript module that will work on modern browsers like Chrome, Firefox and Safari. We expect our users will use these popular browsers so it is necessary for our product to interface with them.

2.2 Product functions

1) The Eclipse Image Processor application will ingest photos of the eclipse, align them spatially, and categorize them to help recover temporal ordering. These aligned and categorized images will be cropped so that the Sun occupies the same amount of space in each image and will be exported. All data that is required to take the raw input images and produce the exported images will be saved to a data file. Additionally, select EXIF information will be extracted from the image files and included in this output data file.
2) The Eclipse Simulator will be a standalone JavaScript module enabling users to ”preview” the eclipse. It will be designed in a stylized, 2D manner. The simulator will incorporate a time slider that allows users to simulate the eclipse in a time window spanning from 12 hours before the eclipse to 12 hours after it. As users drag the time slider, the eclipse will animate in the simulator window. The view of the eclipse which users are presented will be specific to the selected location.

2.3 User characteristics

1) The Eclipse Image Processor application will be used by Google Engineers.
2) The Eclipse Simulator application will be used by the general public. No unusual technical/scientific knowledge is expected of these users. It is assumed however, that these users are familiar with the internet and web browsers.

2.4 Constraints

None.

2.5 Assumptions and dependencies

1) This SRS assumes the availability of Ubuntu 16.04.
2) This SRS assumes the availability of Google Cloud Platform n1- standard-4 virtual machines.
3) This SRS assumes the availability of the OpenCV computer vision library.

2.6 Apportioning of requirements

See Gantt Chart in Appendix.

3 SPECIFIC REQUIREMENTS

3.1 External Interfaces

3.1.1 Eclipse Simulator

1) The simulator is a standalone JavaScript module that can be included on an existing webpage.
2) Users can select the location from which to simulate the eclipse. This can be entered at any point while using the simulator.
   a) Location can be entered as: latitude/longitude, address, zip code, city name, state name.
   b) Initial simulator location can be programmatically set as initialization parameter.
3) Users will be able to adjust the simulator time from 12 before the eclipse to 12 hours after it.
   a) Time can be advanced via a draggable slider or clickable buttons.

3.1.2 Eclipse Image Processor

1) The image pre-processor will be compatible with Ubuntu 16.04 and will include a script that to install all dependencies/build the binary.
2) The application will accept the following input as command line arguments:
   a) Required: image_list_file
      i) Absolute or relative (to the directory the binary was invoked from) path to file containing a list of image filenames with no directory prefix.
   b) Required: output_dir
      i) Directory to write output files to.
   c) Optional: image_path_prefix
      i) Absolute or relative (to the directory the binary was invoked from) path to prepend to each image filename in image_file_list. Defaults to "./".
3) The application will accept JPEG (.jpeg/.jpg) and PNG (.png) image files.
   a) Images of an invalid format will be disregarded and an error message will be written to stderr.
4) The application will write the following output to the output_dir directory:
   a) image_transformations.txt
      i) File containing one line per image processed with the following values (comma separated):
         A) processed_image: processed image filename
         B) image_type: image type (FULL_DISK/TOTALITY/etc.), see requirement #2 of 3.2.2
         C) rot_angle: angle original image was rotated (degrees)
         D) crop_topl_x: x coordinate of top left corner of cropped image (refers to rotated image)
         E) crop_topl_y: y coordinate of top left corner of cropped image (refers to rotated image)
         F) crop_botr_x: x coordinate of bottom right corner of cropped image (refers to rotated image)
         G) crop_botr_y: y coordinate of bottom right corner of cropped image (refers to rotated image)

H) rel_center_offset: relative offset of solar/lunar disk centers, see requirement #3 of 3.2.2 (optional, only included for CRESCENT type images)

I) diamond_rel_size: size of diamond relative to the size of the solar disk, see requirement #4 of 3.2.2 (optional, only for DIAMOND_RING type images)

J) timestamp: timestamp at which image was taken (optional, only included when images have an EXIF timestamp field)

K) eclipse_path_percent: percentage through eclipse totality path at which image was taken, see requirement #8 of 3.2.2 (optional, only included for images with GPS coordinate EXIF field)

b) Processed image files

i) Processed image files will be saved into a sub-directory of output_dir called "images" and will be named as follows: *_pp.[png|jpeg|jpg].

5) Images that are rejected will not be added to the image_transformations.txt file and will not be cropped/exported to the images directory. When an image is rejected a message with this information will be printed to stdout.

6) The application will format all log messages as follows:

a) img_preproc:level:timestamp:message

i) Values of level: ERROR/WARNING/INFO/DEBUG

ii) Value of timestamp: current timestamp

iii) Value of message: specific logging message

3.2 Functional Requirements

3.2.1 Eclipse Simulator

1) Displayed solar/lunar placement will be based on location and time and will account for edge cases like when the location is not in the path of totality. For example, if the location is on the opposite side of the world as the eclipse, the simulator should shift to a night time display.

2) Simulator will display the local time that the simulator is set to, e.g. there is a well defined time associated with the user selecting Corvallis, Oregon as their location and a simulator time of -3:13 (3 hours 13 minutes) before the eclipse. This time should be displayed on the simulator.

3.2.2 Eclipse Image Processor

1) Invalid JPEG and PNG files (e.g. cannot be opened by OpenCV, width/height equal to 0px, etc.) will be ignored and an error message will be written to stderr.

2) The application will classify the input images as being one of the following types:

a) FULL_DISK

i) Image of an unobscured solar disk.

b) TOTALITY

i) Image of a total solar eclipse.

c) CRESCENT

i) Image of a partially eclipsed Sun, creating a ”crescent” shape.

d) DIAMOND_RING

i) Image of a nearly fully eclipsed Sun where there is one ”hot spot” on the Sun’s perimeter. This hot spot along with the Sun’s perimeter have the shape of a diamond ring.

3) For images of type CRESCENT, the application will compute/export a delta of the position of the center of the Sun and the Moon relative to the size of the solar disk. This delta will be a signed value based on the Sun’s position, i.e. if the Moon is to the left of the Sun (in the cropped/rotated image) the delta will be negative and conversely, if the Moon is to the right of the Sun the delta value will be positive.
4) For images of type DIAMOND_RING, the application will compute/export the size of the ”diamond” relative to the size of the solar disk.
5) Images where the solar disk has a radius of less than 50px will be rejected (see requirement #5 in 3.1.2).
6) The application will crop the images to be square with the Sun centered. The images will be cropped so that there is a 100px pad between the solar perimeter and the edge of the image on all sides.

a) Images that do not have enough room around the Sun to allow for the pad described above will be rejected, see requirement #5 in 3.1.2.

7) The application will rotate DIAMOND_RING and CRESCENT type images so that they are aligned horizontally, as described by Krista et al.
8) For images with GPS EXIF information, the application will compute/export the percentage through the eclipse’s path of totality at which the image was taken. 0% will be defined as the westmost point on that path of totality that is over land, this point is on the west coast of Oregon. 100% will be defined as the eastmost point on the path of totality that is over land, this point is on the east coast of South Carolina. The application will compute the point on the path of totality nearest the point where the image was taken. This point will be used to compute the percentage through the path of totality at which the image was taken.
9) Images with GPS EXIF information that are not on the path of totality will be rejected, see requirement #5 in 3.1.2.

3.3 Performance Requirements

3.3.1 Eclipse Simulator

1) All simulator resources will load in less than 500ms given a 1-10 Mbps internet connection.

3.3.2 Eclipse Image Processor

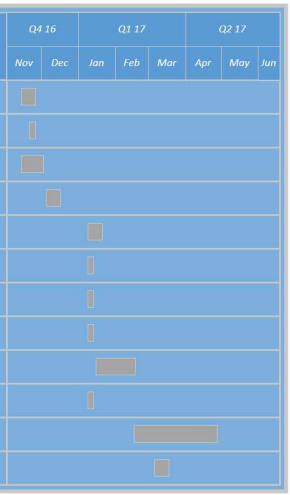
1) The application should take less than 5 seconds to process an image when running on a Google Cloud Platform n1-standard-4 virtual machine.

4 Supporting Information

4.1 Appendix

1) Figure 1. Project Gantt Chart

ID	Task Name	Start	Finish	Duration
1	Eclipse Simulator: Basic UI	11/14/2016	11/25/2016	2w
2	Eclipse Simulator: Google Maps API Integration	11/21/2016	11/25/2016	1w
3	Eclipse Simulator: Sun and Moon Position Calculations	11/14/2016	12/2/2016	3w
4	Eclipse Simulator: Connect User Interface with Model	12/5/2016	12/16/2016	2w
5	Eclipse Simulator: UI Fit and Finish	1/9/2017	1/20/2017	2w
6	Eclipse Image Processor: Write Scripts for Application Build and Installation	1/9/2017	1/13/2017	1w
7	Eclipse Image Processor: Command Line Input and File Parsing	1/9/2017	1/13/2017	1w
8	Eclipse Image Processor: Output Writing and Formatting	1/9/2017	1/13/2017	1w
9	Eclipse Image Processor: Rough End to End Pipeline Implementation	1/16/2017	2/17/2017	5w
10	Eclipse Image Processor: Standard Logging Functionality	1/9/2017	1/13/2017	1w
11	Eclipse Image Processor: Accuracy and Speed Optimization	2/17/2017	4/27/2017	10w
12	Eclipse Image Processor: EXIF and GPS Timestamp Processing	3/6/2017	3/17/2017	2w



2.2 Requirements Revisions

2.2.1 Requirements Changes

# in Original Document	**Requirement**	**Changes**	**Comments**
3.1.1.3	Users will be able to adjust the simulator time from 12 hours before the eclipse to 12 hours after it. 1) Time can be advanced via a draggable slider or clickable buttons.	Changed to 1.5 hours before / after the eclipse.	This change came from our UX designer and product manager, who felt that 12 hours created too much padding before / after the eclipse.
3.1.2.1	The image pre-processor will be compatible with Ubuntu 16.04 and will include a script that to install all dependencies/build the binary.	Changed so that we will include *instructions* to install dependencies and a textitmakefile to build the binary.	This allows for separation of dependency installation and building of the actual program.

3.1.2.2	The application will accept the following input as command line arguments: 1) Required: image_list_file a) Absolute or relative (to the directory the binary was invoked from) path to file containing a list of image filenames with no directory prefix. 2) Required: output_dir a) Directory to write output files to. 3) Optional: image_path_prefix a) Absolute or relative (to the directory the binary was invoked from) path to prepend to each image filename in image_file_list. Defaults to "./".	Removed image_path_prefix parameter and made it so that image_list_file must contain full paths. Added mode parameter. Added parameters for `cv::HoughCircles`.	image_path_prefix / image_list_file changes made at client's request, for convienience. Added parameters were added to improve image processor development workflow.
3.1.2.4	[Requirement too long, see original requirements document]	Renamed output file. Removed image crop / rotation / classification / EXIF based values. Added variable number of found circles, execution times, and general observations.	These changes were made as the overall goal of the image processor was simplified to simply recognize total solar eclipse position. Additionally, the format was made more flexible to aid in development of modified image processor pipelines.

3.1.2.5	Images that are rejected will not be added to the image transformations.txt file and will not be cropped/exported to the images directory. When an image is rejected a message with this information will be printed to stdout.	Removed	This requirement was removed because we never reached this stage in development of the image processor.
3.1.2.6	The application will format all log messages as follows: a) img preproc:level:timestamp:message i) Values of level: ERROR / WARNING / INFO / DEBUG ii) Value of timestamp: current timestamp iii) Value of message: specific logging message	Removed	This requirement was removed because our sponsor did not need logging built in to the image processor at this time.
3.2.1.1	Displayed solar/lunar placement will be based on location and time and will account for edge cases like when the location is not in the path of totality. For example, if the location is on the opposite side of the world as the eclipse, the simulator should shift to a night time display	Changed, the simulator is restricted to the United States so it does not need to account for cases around the world.	After discussions with our sponsor we agreed to restrict the simulator the US because that is the only place the eclipse is visible.

3.2.1.2	Simulator will display the local time that the simulator is set to, e.g. there is a well defined time associated with the user selecting Corvallis, Oregon as their location and a simulator time of -3:13 (3 hours 13 minutes) before the eclipse. This time should be displayed on the simulator.	Removed	This requirement was removed because our simulator ended up including a time slider that displayed local times relative to the eclipse so a standalone time display was unecessary.
3.2.2.1	Invalid JPEG and PNG files (e.g. cannot be opened by OpenCV, width/height equal to 0px, etc.) will be ignored and an error message will be written to stderr.	Removed	In discussions with our sponsor we decided that the image processor would only be handling valid images and as such would not need to identify invalid images and write and error message.
3.2.2.2	The application will classify the input images as being one of the following types: a) FULL DISK i) Image of an unobscured solar disk. b) TOTALITY i) Image of a total solar eclipse. c) CRESCENT 9 i) Image of a partially eclipsed Sun, creating a crescent shape. d) DIAMOND RING i) Image of a nearly fully eclipsed Sun where there is one hot spot on the Suns perimeter. This hot spot along with the Suns perimeter have the shape of a diamond ring.	Removed	Because of development constraints our sponsor decided that the image processor would only processes images of the eclipse at totality and ass such would not handle classification.

3.2.2.3	For images of type CRESCENT, the application will compute / export a delta of the position of the center of the Sun and the Moon relative to the size of the solar disk. This delta will be a signed value based on the Suns position, i.e. if the Moon is to the left of the Sun (in the cropped/rotated image) the delta will be negative and conversely, if the Moon is to the right of the Sun the delta value will be positive.	Removed	See explanation for requirement 3.2.2 2.
3.2.2.4	For images of type DIAMOND RING, the application will compute / export the size of the diamond relative to the size of the solar disk.	Removed	See explanation for requirement 3.2.2 2.
3.2.2.5	Images where the solar disk has a radius of less than 50px will be rejected.	Removed	Ideal solar disk radius is subject to change and depends on the current image processor implementation.
3.2.2.6	The application will crop the images to be square with the Sun centered. The images will be cropped so that there is a 100px pad between the solar perimeter and the edge of the images on all sides.	Removed	Google will be handling the cropping of images and their respective ideal dimensions.

3.2.2.7	The application will rotate DIAMOND RING and CRESCENT type images so that they are aligned horizontally, as described by Krista et al.	Removed	Google will be handling the rotation of images.
3.2.2.8	For images with GPS EXIF information, the application will compute/export the percentage through the eclipses path of totality at which the image was taken. 0% will be defined as the westmost point on that path of totality that is over land, this point is on the west coast of Oregon. 100% will be defined as the eastmost point on the path of totality that is over land, this point is on the east coast of South Carolina. The application will compute the point on the path of totality nearest the point where the image was taken. This point will be used to compute the percentage through the path of totality at which the image was taken.	Removed	We do not handle any GPS EXIF informaion with the image processor. This will be done by Google.
3.2.2.9	Images with GPS EXIF information that are not on the path of totality will be rejected.	Removed	See above requirements comment.

3.3.1.1	All simulator resources will load in less than 500ms given a 1-10 Mbps internet connection.	Changed the loading time to 700ms.	This was changed due to client preferences.

2.2.2 Requirements Additions

# in Latest Document	**Requirement**	**Comments**
3.1.1.4	Users will be able to ”play” the eclipse and advance through the available time domain automatically by pressing a play button.	Added to enhance user experience when visualizing the solar eclipse.
3.1.2.2	[Requirement too long, see original requirements document]	Additionally parameters were added when running the image processor to allow testing of different processing methods.
3.1.2.4	The application will write the following output to the output_dir directory (When run in batch mode): a) metadata.txt i) File containing one line per image processed with the following values: A) processed_image: processed image filepath (absolute) B) found_circle(s): circles found by the image processor, format: c(center_x, center_y, radius) C) execution_time(s): time(s) taken for varius parts of the image processor to execute, format t(”name”, num_secs) D) observation(s): observations about the image, format, ”observation text” b) Processed image files i) Processed image files will be saved into output_dir	The purpose of the image processor changed as development began. These requirements changes reflect that change.

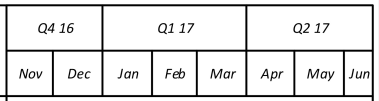
3.1.3.1	The developer pipeline will accept the following command line arguments: a) Required: $DIR, the directory to use for the image/data storage. The following will be saved into DIR: i) If download flag set: A clone of the $GCS_BUCKET ii) A directory called output that will contain: A) All the processed images $GCS_BUCKET B) A metadata file that will contain the processed image names along with output information from the image processor C) An HTML file that includes summaries of the image processor output info.	The developer pipeline replaced the image processor manager. This was due to requirements changed by our sponsor.
3.1.3.2	Required: $GCS_BUCKET, the Google Cloud Storage Bucket that contains the image to process.	We utilize Google Cloud Storage Buckets for easy storage and access to eclipse images.
3.1.3.3	Optional: download, if set, the developer pipeline will download the images from Google Cloud Storage. Otherwise, it will assume these images are included in $DIR/$GCS_BUCKET	See above requirement explanation.
3.1.3.4	Optional: $PIPELINE_FLAGS, arguments to pass to the image processor when it is involved.	See requirement 3.1.3.1 requirement explanation.
3.2.1.2	The simulator location will be restricted to the United States.	This requirement was added because the eclipse only occurs in the US so having the simulator function outside the US is not important.
3.2.1.3	The simulator environment will darken as the eclipse progresses.	This requirement was added because our sponsor requested it.
3.2.1.4	The simulator will feature a zoom mode, where the sun appears larger in the sky. In zoom mode, the sun will remain in the center of the screen. The simulator will effectively track along with the suns movement.	This requirement was added because our sponsor wanted to give photographers a close up view of the eclipse in the simulator.

3.2.1.5	The simulator will be mobile friendly.	This requirement was added by our sponsor since our simulator was going to be live on the web.
3.2.2.1	The image processor will be implemented as a class that will be easily inheritable / modifiable by developers.	This requirement was added by our sponsor to facilitate future image processor development.
3.2.2.2	The image processor will feature two modes, ”window” and ”batch”. In window mode, after an image is processed, windows will open showing the original image, processed image, and intermediate images. In batch mode, all images will be processed sequentially without opening any windows. Processed images and metadata will be exported as described in External Interfaces: Eclipse Image Processor.	This requirement was added to facilitate development of the image processor.
3.2.2.3	The image processor will identify the circles of the sun/moon in the images it processes.	This requirement was added to clarify the functionality of the image processor.
3.2.2.4	The image processor will record the number of seconds (wall clock time) needed to complete various portions of each images processing.	This requirement was added so that we could monitor and improve the performance of the image processor.
3.2.3.1	The developer pipeline will be able to download images from Google Cloud Storage, if requested.	This requirement was added so that the pipeline can use many different datasets.
3.2.3.2	The developer pipeline will build and invoke the image processor on the requested images using batch mode.	This requirement was added so that the pipeline runs over entire image sets.
3.2.3.3	The developer pipeline will assemble the results of running the image processor into and HTML file. This HTML file, along with the processed images it references, will be uploaded to Google Cloud Storage to a public URL.	This requirement was added so that the image processor’s results are easily viewable by a team of developers.

3.2.3.4	The HTML file created by the developer pipeline will summarize the data included in the metadata.txt file exported by the image processor.	This requirement was added so that the image processor's results are easily viewable by a team of developers.
3.3.3.1	The image processor developer pipeline will download/upload images from/to Google Cloud Storage in parallel.	This requirement was added so that the developer pipeline efficiently downloads photos.

2.2.3 Final Gantt Chart

ID	*Task Name*	*Start*	*Finish*	*Duration*
1	Eclipse Simulator	11/21/2016	3/13/2017	16.2w
2	Eclipse Image Processor	3/20/2017	5/19/2017	9w
3	Eclipse Image Processor Developer Pipeline	3/20/2017	5/19/2017	9w
4	Eclipse Image Classification	5/5/2017	6/12/2017	5.4w



3 DESIGN DOCUMENT

3.1 Original Design Document

Design Document

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

December 1, 2016

CS 461 - Fall 2016

Abstract

This document describes in detail the design for the various components of the North American Solar Eclipse 2017 senior capstone project. These components are described according to the IEEE 1016-2009 standard. There are three high level components detailed in this document, the Eclipse Image Processor, Eclipse Image Processor Manager, and Eclipse Simulator. The sections of this document are broken into subsections corresponding to these components as applicable.

David Konerding — 12/1/2017

David Konerding, Project Sponsor — Date

Bret Lorimore — Date

George Harder — Date

Jacob Fenger — Date

TABLE OF CONTENTS

1 Design Stakeholders and Their Concerns

The primary stakeholder in this project is David Konerding of Google. He is one of the managers of the Eclipse Megamovie project that is sponsoring this Senior Capstone project. David Konerding's concerns are listed below.

1.1 Image Processor

1.1.1

The image process needs to take in an image, identify if the image has a total solar eclipse, and if it does further process it so that it can be stitched into a timelapse movie by Eclipse Megamovie engineers.

1.1.2

The mean processing time for an image must be one second.

1.1.3

Images must be processed in no longer than five seconds.

1.1.4

Images must be filtered so that the processed images are only images of the eclipse at totality.

1.1.5

Only high quality images, defined as having a 50 pixel solar disk size and padding around the disk of 100 pixels, should be accepted by the image processor.

1.1.6

Once images have been filtered, they need to have metadata attached in a way that allows easy stitching of eclipse images.

1.1.7

The image processor needs to be able to be called by the image processor manager with appropriate input data.

1.1.8

Image processor needs to be able to use GPS EXIF information associated with images.

1.2 Image Processor Manager

1.2.1

Image processor manager should download images needing processing from Google Cloud Storage.

1.2.2

Image processor manager should invoke image processor with downloaded images.

1.2.3

Image processor manager should upload processed images and corresponding metadata - the output from the image processor - to Google Cloud Storage and Datastore, respectively.

1.2.4

Image processor should download/upload images at the same time the image processor is processing other images.

1.2.5

Image processor manager should invoke multiple image processor instances concurrently with different input images. The number of image processor instances launched should be determined by the number of cores on the host VM.

1.2.6

Image processor manager instances should be able to run alongside other image processor manager instances running on (potentially) different machines. These discrete instances should not attempt to process the same images.

1.3 Eclipse Simulator

1.3.1

The solar eclipse must be accurately simulated based on user entered location information.

1.3.2

Users will be able to adjust simulator time via a draggable slider or clickable buttons.

1.3.3

Simulator will only support locations within continental United States.

1.3.4

The simulator must load in less than 500ms given a 1-10 Mbps internet connection.

2 DESIGN VIEWPOINTS

2.1 Image Processor

2.1.1 Speed and Performance

Concerns: 1.1.2, 1.1.3

Elements: 4.1.1, 4.1.2, 4.1.3, 4.1.4

Analytical Methods: The primary criteria and methods in constructing this view is whether or not we are achieving the desired average speeds on our golden data set.

Viewpoint Source: George Harder

2.1.2 Accuracy

Concerns: 1.1.1, 1.1.4, 1.1.5, 1.1.8

Elements: 4.1.1, 4.1.2, 4.1.5, 4.1.6 4.1.9, 4.1.10

Analytical Methods: In constructing the corresponding view, we will be evaluating it based on whether or not the image processor is correctly identifying eclipses with at least 90% accuracy on our golden data set.

Viewpoint Source: George Harder

2.1.3 Input and Output

Concerns: 1.1.6, 1.1.7

Elements: 4.1.2, 4.1.4, 4.1.7, 4.1.8, 4.1.9

Analytical Methods: This view will be evaluating whether or not the input and output of the image process meet the specifications defined in the design document.

Viewpoint Source: George Harder

2.2 Image Processor Manager

2.2.1 Intra-instance Concurrency

Concerns: 1.2.4, 1.2.5

Elements: 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5, 4.2.6, 4.2.7

Analytical Methods: Overall VM CPU utilization, overall VM network interface utilization. Both these values should be maximized as much as possible.
Viewpoint Source: Bret Lorimore

2.2.2 Inter-instance Concurrency and Synchronization

Concerns: 1.2.1, 1.2.3, 1.2.6
Elements: 4.2.1
Analytical Methods: No image should be successfully processed by multiple image processor instances, whether or not these run on the same VM or are managed by the same image processor manager.
Viewpoint Source: Bret Lorimore

2.2.3 Invocation

Concerns: 1.2.2
Elements: 4.2.6
Analytical Methods: Multiple image processor processes should be able to be easily launched concurrently with different input data.
Viewpoint Source: Bret Lorimore

2.3 Eclipse Simulator

2.3.1 Interface

Concerns: 1.3.1, 1.3.2
Elements: 4.3.1, 4.3.2, 4.3.4, 4.3.7
Analytical Methods: Interface should be appealing to the user as well as being responsive and fast.
Viewpoint source: Jacob Fenger

2.3.2 Loading Performance

Concerns: 1.3.4
Elements: 4.3.3, 4.3.4, 4.3.5, 4.3.6, 4.3.7
Analytical Methods: The initial loading time of the simulator should be fast. Additionally, the interactions that the user has with the simulator should be responsive and should not show any significant slow downs.
Viewpoint source: Jacob Fenger

2.3.3 Simulation Accuracy

Concerns: 1.3.1, 1.3.3

Elements: 4.3.3, 4.3.5

Analytical Methods: In the simulator, the Sun and Moon display should reflect scientific accuracy when it comes to relative position and sizes. Additionally, the view of the Sun and Moon above the horizon shall be accurate.

Viewpoint source: Jacob Fenger

3 DESIGN VIEWS

3.1 Image Processor

3.1.1 Speed and Performance [Governed by Viewpoint 2.1.1]

The Eclipse Megamovie project expects to receive photographs on the order of hundreds of thousands from the numerous citizen photographers registered with the project. This being the case, it is of utmost importance that the image processor we are building for the project can process images in a timely manner. This is not only important to being able to build a movie from the images soon after the eclipse, but also because it prevents storage spaces on either end of the image processing pipeline from becoming clogged.

In order to achieve these design goals, we are designing a system that uses the OpenCV C++ library. This API provides us with high performing library methods, like Hough transforms, image crops, and image rotations that are necessary to the image processor's core functionality. In addition to the use of this language and library, we are also designing this system to be single threaded but to also interface with a manager that runs multiple instances of the image processor concurrently. This fact allows us to design the image process to process a single image quickly and accurately and leaves management of the application to a different art of the system.

3.1.2 Accuracy [Governed by Viewpoint 2.1.2]

From a high level, the most fundamental concern in the design of the image processor is that it can accurately identify an eclipse image. The other concerns associated with the design of the image processor are near meaningless if the application is not consistently identifying images that contain total eclipse.

To address this concern, and the related concerns around accepting only high quality images and determining the relative temporal and spatial positioning of images, we are designing this application with accuracy as a primary goal. The meet this goal, the system will build upon an existing eclipse identification algorithm. This algorithm, with improvements we add ourselves, will be the basis for the parts of the system that we design to meet the accuracy requirements of the image processor.

3.1.3 Input and Output [Governed by Viewpoint 2.1.3]

The image processor is one component in a much larger system. For the system to function the image processor needs to interface with the other components in a well defined and seamless manner. In addition to speed and accuracy, we need to design the system with a view toward optimizing the way the image processor interacts with other components.

The ensure a smooth interface with the image processor manager we are designing the image processor to accept a well defined set of command line arguments including the path to the list of images to be processed, the path to an output directory, and the path to prefix image file names with that points to their location. The design of the image processor also takes into account the needs of the engineers handling the processed images. The image processor will write processed images and their metadata in a specific format to an output file.

3.2 Image Processor Manager

3.2.1 Maximal Utilization [Governed by viewpoint 2.2.1]

It is the goal of the image processor manager to maximize hardware utilization on the VMs on which it is running. This means that ideally, the VMs where the image processor manager and therefore the image processor are running will have an average of 100% CPU utilization on all cores at all times.The image processor application will be single threaded and thus by invoking multiple instances of this application, the image processor manager can increase utilization on multiple CPU cores.

The downloading of images to process and uploading of processed images are both very high latency operations. Therefore, instead of waiting for these downloads/uploads to complete with virtually zero CPU utilization in the meantime, the image processor manager can achieve greater average CPU utilization by completing these high latency tasks concurrently to processing other images.

3.2.2 Image Processing [Governed by viewpoint 2.2.2]

The ultimate goal of the image processor manager is to facilitate the processing of eclipse images by the image processor component of this project. This can be achieved by downloading images to process from the cloud, assembling them into a form that can be consumed by the image processor and then invoking that application with the downloaded images as input data.

3.2.3 Synchronization [Governed by viewpoint 2.2.3]

The image processor must process the images that are uploaded by users. These images are uploaded by the eclipsemega.movie website to Google Cloud Storage and metadata entries are also created for them in Google Cloud Datastore. In order for the image processor manager to invoke the image processor with these images, it

must download them from Google Cloud.

The application to stitch processed images into movies, being developed by Google, expects to find processed eclipse images in Google Cloud Storage with corresponding metadata in Google Cloud Datastore. In order to meet this expectation, the image processor manager must upload the results of running the image processor to Google Cloud Storage and Google Cloud Datastore when ready.

To enable scalable image processing performance, it is desirable to enable many VMs to run image processor applications at once. In order to do this efficiently without redundancy, it is necessary to ensure that multiple VMs do not try to process the same images. For that reason, image processor manager instances must mark image files as pending processing before another instances of the image processor manager can queue them for processing. Without implementing this functionality, little to no performance improvements can be expected by deploying multiple image processor manager nodes.

3.3 Eclipse Simulator

3.3.1 User Interface View [Governed by Viewpoint 2.3.1]

The user interface shall utilize 2D animated depictions of the Sun and the Moon as they appear at a user specified time and location. In addition, the user interface will contain background imagery such as a city or hillside landscape. There will also be a time slider, a location input, and a time display for users to interact with or view.

3.3.2 Operating Performance View [Governed by Viewpoint 2.3.2]

The simulator will have low loading times to ensure fast performance for most users. Additionally, the simulator will need to respond in a timely matter when users are interacting with the module.

3.3.3 Eclipse Accuracy View [Governed by Viewpoint 2.3.3]

The simulator shall be accurate enough for any location in the continental United States. This accuracy includes accurate relative Moon and Sun sizes, positions in the rendered scene, and positions relative to one another.

4 Design Elements

4.1 Image Processor

4.1.1 OpenCV

Type: Library

Purpose: OpenCV is the computer vision library we will use to process images. This open source library can

quickly crop and otherwise manipulate images. It best suits our needs for a computer vision utility.

4.1.2 C++

Type: Class

Purpose: This application will be written in C++ in order to give us better control over the speed at which the application runs and the necessary functionality for reading and writing files.

4.1.3 Image Processing Time

Type: Constraint

Purpose: This element exists because our requirements specify that the average time for an image to be processed must be less than one second.

4.1.4 Serial Image Processing

Type: Framework

Purpose: The image processor will process images one at a time because the Image Processor Manager handles all parallelization.

4.1.5 Hough Transform

Type: Procedure

Purpose: The Hough transform is an algorithm for identifying circles and lines in images. It will be used by the image processor to identify total eclipse images. OpenCV has the circular Hough transform built in.

4.1.6 Image Quality Error Checking

Type: Constraint

Purpose: The requirements of the image processor specify that only images with a solar disk of at least 50 pixels and 100 pixels of padding around the sun will be accepted.

4.1.7 Command Line Arguments

Type: Constraint

Purpose: The image processor needs to have a well defined set of command line arguments so that it can interface with the image processor manager without difficulty.

4.1.8 Data writer

Type: System

Purpose: This component is meant to encapsulate the functionality necessary to write the processed images and their associated metadata to an output file.

4.1.9 Image Data Structure

Type: Class

Purpose: This class will encapsulate the information and methods needed to manage the images we will be processing. It will also work closely with the Data Writer to write images and their metadata to files.

4.1.10 Solar Eclipse Image Standardisation and Sequencing (SEISS)

Type: Procedure

Purpose: The SEISS algorithm is an image processing algorithm that can identify images of eclipses, including eclipses at totality [**?**]. We will be basing part of the image processor off of this algorithm.

4.2 Image Processor Manager

4.2.1 Image List Downloader

Type: Subsystem

Purpose: The purpose of this subsystem is to download and return a list of images from Datastore to be processed by the image processor application. It will use Datastore transactions to ensure that each image in this list is marked as pending processing before it can be retrieved by another image processor manager instance. The max number of image files retrieved will be determined by the number of image processor processes that are going to be launched. This value will be a parameter of the image list downloader subsystem.

4.2.2 Image Downloader

Type: Subsystem

Purpose: The purpose of this subsystem is to download and save individual image files. It will accept the name of an image file to retrieve from Cloud Storage and a place to store this file, and will then download the file and save it to the desired location.

4.2.3 Image Download Manager

Type: System

Purpose: The purpose of this system is to coordinate design elements 4.2.1 and 4.2.2, the *Image List Downloader* and *Image Downloader*, respectively. It will use the *Image List Downloader* to retrieve a list of images to download

and then will use the Python multiprocessing module to launch multiple instances of the *Image Downloader* subsystem concurrently to download the images in the list.

4.2.4 Result Uploader

Type: Subsystem

Purpose: The purpose of this subsystem is to upload an individual processed image to Google Cloud Storage and upload its metadata to Google Cloud Datastore.

4.2.5 Result Uploader Manager

Type: System

Purpose: The purpose of this system is to coordinate element 4.2.4, the *Results Uploader*. It will use the Python multiprocessing module to launch multiple instances of the *Results Uploader* concurrently to upload the output of the image processor.

4.2.6 Image Processor Invoker

Type: System

Purpose: The purpose of this system is to invoke multiple instances of the image processor application concurrently using the Python subprocess module. It will distribute images to process over multiple image processor processes to increase throughput. The number of image processor processes will be determined by the number of cores on the host VM.

4.2.7 Controller

Type: System

Purpose: The purpose of this system is to coordinate the three other systems that are part of the image processor manager, design elements 4.2.3, 4.2.5, and 4.2.6, the *Image Download Manager*, the *Result Uploader Manager*, and the *Image Processor Invoker*. It will run these systems in parallel so that images are downloaded/uploaded at the same time that other images are being processed.

4.3 Eclipse Simulator

4.3.1 Scalar Vector Graphics (SVG)

Type: System

Purpose: This element shall be used for the front-end display of the eclipse simulator. Two-dimensional images of the Sun and Moon will be altered based on how the user interacts with the module.

4.3.2 Cascading Style Sheets (CSS)

Type: System

Purpose: CSS helps with the front-end display of the simulator by helping produce better looking output.

4.3.3 Ephemeris JavaScript Library

Type: Library

Purpose: This library is used to compute eclipse information to be used for displaying the Sun and Moon.

4.3.4 View

Type: Component

Purpose: Combined the HTML, SVG, and CSS elements for simulator display and interaction for the user.

4.3.5 Model

Type: Component

Purpose: Backend library in JavaScript used for computing eclipse information which will be passed to the controller. This entity utilized the Ephemeris JavaScript library for support.

4.3.6 Controller

Type: Component

Purpose: Controls the interaction between the model and view. Information will be passed between these entities.

4.3.7 Model-View-Controller Architecture

Type: Relationship

Purpose: This architecture is defined by the interactions of the model, view, and controller entities.

5 Design Overlays

5.1 Image Processor

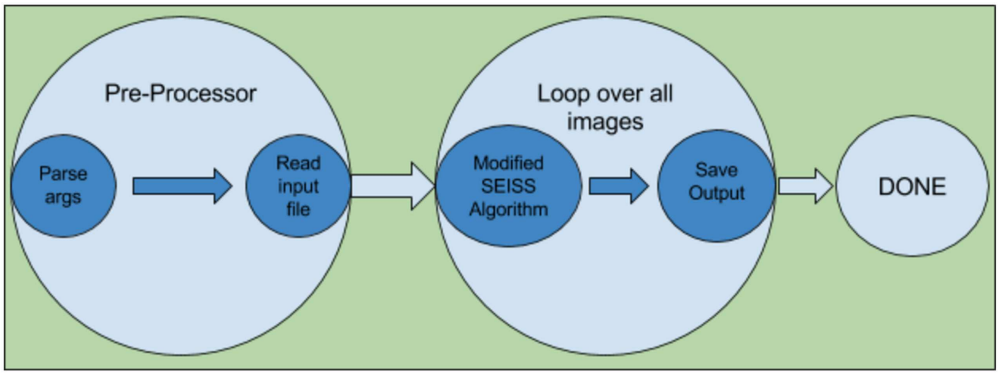


Fig. 1. Image processor basic dataflow

5.2 Image Processor Manager

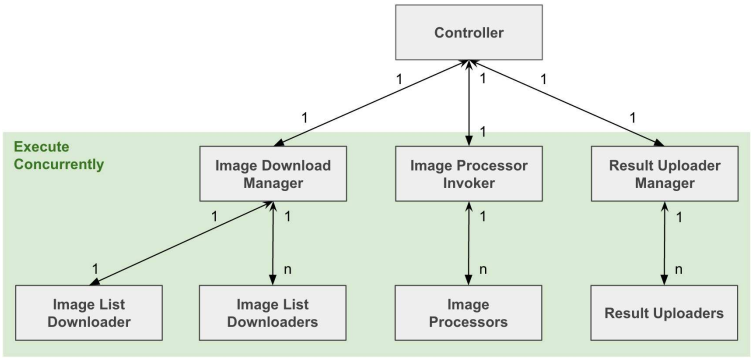


Fig. 2. Image processor manager system diagram [Associated with Viewpoint 2.2.1]

5.3 Eclipse Simulator

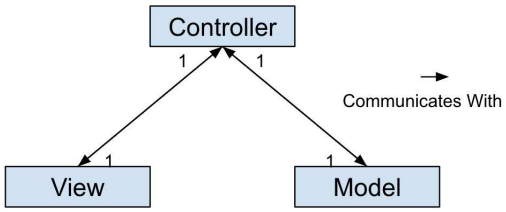


Fig. 3. Eclipse simulator MVC architecture

6 Design Rationale

6.1 Image Processor

The design of this system is based on needs surrounding accuracy, speed, and ability to interface with other components in the larger system. As was detailed in the technology review, we made certain decisions about what tools and algorithms to use based on the specific requirements of our system. The use of these tools has necessarily shaped the design of this application.

Additional information regarding design rationale can be found in the 'Purpose' section of the design elements.

6.2 Image Processor Manager

6.2.1 High Level System Design

From a high level the system is designed to be highly parallel with the aim of achieving as close to 100% CPU utilization on all cores as possible. In order to achieve this it is necessary to ensure that images to process are downloaded ahead of the time they're needed, this way the latency of their download time is hidden by processing other images. It is also necessary to ensure that processed images do not back up. The design elements outlined above fit together to form a system that naturally supports a very high level of concurrency. Subsystems to handle individual tasks are built and then multiple instances of these subsystems can be launched concurrently with different parameters. These subsystems are called by their parent systems.

6.2.2 Process Based Concurrency

As outlined in detail in our technology review, to achieve true multi-core concurrency in Python, use of process based concurrency such as that used in the built-in multiprocessing module is necessary. This is because multi-thread execution in Python is severely throttled by the Global Interpreter Lock (GIL) which essentially forces serialization of the execution of concurrent threads.

6.2.3 Motivation for separation of Image List Downloader and Image Downloader

Design element 4.2.1, the *Image List Downloader* is separated from design element 4.2.2, the *Image Downloader* as lists of all the images that need to be downloaded at a given time form relatively small collections of data. Therefore, it is more efficient to make a single Datastore query to retrieve all the images that need to be downloaded, rather than retrieving these one at a time using separate queries. As the image files themselves are quite large, these still need to be retrieved individually. In fact, the Google Cloud Client Library for Python does not support the download of multiple files from Cloud Storage using a single API call. On top of this constraint, downloading the actual image files individually ensures that we are able to establish a large number of concurrent connections to Cloud Storage to saturate the VM network interface as much as possible and achieve very high overall download speeds.

6.3 Eclipse Simulator

The goals for the simulator were to provide a fast and responsive experience to the user while providing scientifically accurate results. One main concern with this is that the time to compute information regarding the Sun and Moon must be quick enough to not pose any significant delay for the rest of the simulator.

We decided to utilize the MVC architecture since model and view components can be exchanged without compromising the whole system. System designer only need to account how the components interact with each other to update the system. In the technology review, we compared two libraries: Suncalc and Ephemeris. Initially, we chose SunCalc as the better library due to better documentation and ease of use. After further testing, the results that Ephemeris was providing were much better which spurred the change to utilizing Ephemeris as our support in ephemeris computations.

Additionally, we chose to utilize a scalar vector graphics format due to sub-element event processing and being easily to move the Sun and Moon around as animations.

7 Design Languages

1) UML Version 2.5

8 APPENDICES

8.1 Appendix I- Change History

1) November 30th, 2016- Document Created

8.2 Appendix II- Glossary of Terms

Eclipse Megamovie Project: The Eclipse Megamovie Project is a collaboration between Google and scientists from Berkeley and several other institutions with the aim of collecting large quantities of observations of the solar eclipse that will pass over the United States on August 21, 2017. The project will crowdsource photos of the eclipse from photographers at various points along the path of totality.

Metadata: In this document when we refer to metadata we are referring to any data associated with a file that is not that file itself - e.g. for a photo, the GPS coordinates at which that photo was taken are considered metadata.

Image processor instance/process: a single image processor process.

VM: A virtual machine - a hosted virtual server. In this document, when we refer to a VM we are specifically referring to a VM hosted in Google Cloud - this could be either a bare VM or a single node in a VM cluster.

Google Cloud Datastore: A fully managed, scheme-less NoSQL database solution offered from Google.

Google Cloud Storage: A fully managed file storage solution offered from Google, optimized for storing/accessing large files.

Mbps: Megabits per second. This is referring to download and upload speeds.

Ephemeris: An ephemeris is used to give positional information about astronomical objects

Machine: The term machine is used throughout this document in reference to a particular VM instance.

JPEG/JPG: JPEG is a lossy compression technique for images. When we refer to JPEG/JPG files in this document we are referring to image files compressed in this method with the .jpeg or .jpg file extension.

PNG: PNG refers to the Portable Network Graphics image file format. Images in the PNG format are frequently referred to as ”PNGs” and are saved with the .png file extension.

EXIF: EXIF refers to the Exchangeable Image File Format, a standard media file format. Within the scope of this document EXIF generally refers to metadata associated with images processed by the image processor.

SEISS: Solar Eclipse Image Standardisation and Sequencing. This refers to an algorithm developed by Krista et al. [**?**] that seeks to identify eclipse images and classify them by the phase that the eclipse is in. We will be using and modifying this algorithm to suit the needs of this project.

REFERENCES

[1] K. Larisza and S. McIntosh, "The standardisation and sequencing of solar eclipse images for the eclipse megamovie project."

3.2 Design Document Changes

Over the course of the year our design document changed significantly for several reasons. First, our priorities shifted from image processing to building a fully featured and robust simulator. Second, we had to scale back our initial goals for the image processor and in response to delays in Google allowing code to be open sourced. Third, we completely eliminated the image processor manager because of the combination of the two previously mentioned factors. Lastly, we added a developer tool to the image processor to facilitate further development after we left the project. This came about because as our priorities shifted away from image processing and development time waned, we wanted to make it easy for our sponsor to take over the development once we finished. The reasons for all of these changes can be summed up as real world challenges that are to be expected over the life of a project.

4 Tech Review

4.1 Original Tech Review

Tech Review

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

November 14th, 2016

CS 461 - Fall 2016

Abstract

This document compares various technologies for use in the North American Solar Eclipse 2017 senior capstone project. Specifically, we look at three specific parts of the project, the Eclipse Image Processor, the Image Processor Manager, and the Eclipse Simulator. For each of those high level components, we review applicable technologies for three of their sub-components. For each of the nine total sub-components, we make a conclusion about which of the considered technologies is best suited to our needs, and will thus be used in our implementation.

1 Introduction

This technology review proceeds in three similarly structured sections. First, George Harder discusses which technologies will be the most effective for the implementation of the Eclipse Image Processor. This section is broken down into three subsections that correspond to the pieces he is responsible for in this project. The pieces are Image Classification and Manipulation, Runtime Environment, and Circle Detection. In the next section Bret Lorimore discusses his area of responsibility, the Image Processor Manager. The three pieces that Bret examines are Downloading/Uploading from Google Cloud Storage, Storing Photo Metadata, and Application Parallelization. Lastly, Jacob Fenger details the pieces and technologies that will comprise the Eclipse Simulator. The three pieces he will be responsible for are User Interface, Gathering User Latitude/Longitude Information, and Displaying Sun/Moon Based On Latitude, Longitude, and Time Information.

An important note with regard to some of the technologies being discussed is that Google is sponsoring this project and as such we have been asked to use Google products when possible. This is especially relevant to the sections on runtime environments and storing the photo metadata.

2 Eclipse Image Processor

The eclipse image processor is the piece of our project that handles the spatial and temporal alignment of the eclipse images that are uploaded to the Eclipse Megamovie website. We have identified three pieces into which the image processor can be broken down. These are image classification and manipulation, the runtime environment, and circle detection. In order for this element of our project to operate effectively it is critical that these three pieces utilize robust and functional technologies. This section of the technology review details what options are available for implementing each of these three pieces, analyzes these options, and arrives at a determination as to which option is the best in the context of this project.

2.1 Image Classification and Manipulation

The image classification and manipulation portion of the eclipse image processor encapsulates a few key requirements for this project. For our project to be functionally complete it must determine temporal ordering of the images, align them spatially, and crop them such that the sun is the same size in all of the images. This process requires the use of a computer vision application programming interface (API). Three possible options of APIs are: OpenCV, SimpleCV, and VXL.

The relative strengths and weaknesses of these three options can be evaluated using several key criteria that emerge when considering the requirements for this element of the project. These criteria are: speed, support/documentation, availability, ease of installation, and ease of use. Fortunately, all three of these tools are available online and provide documentation on installation so these categories do not require much discussion

[1]–[3]. With these criteria met, and considering the performance requirements of our project, the next most important criteria to consider is speed.

OpenCV and VXL both rank highly in the speed category because they are written natively in C++ [1], [2]. SimpleCV on the other hand is written in Python and thus uses an interpreter [3], which causes speed penalties. SimpleCV's use of Python is a significant drawback and it cannot be ignored. Because of the large volume of images this tool is expected to process, we are targeting a mean processing time of 1 second per image. This is not necessarily impossible with an interpreted language, but it introduces major difficulties in trying to meet this requirement. OpenCV and VXL are the clear winners when considering speed.

While not as critical to meeting specific requirements as speed, it is important to consider the costs associated with learning a new API. While all three APIs provide online documentation, these docs are by no means equal. OpenCV and Simple CV both provide easy to understand and what appear to be comprehensive tutorials [1], [3]. While OpenCV may be more complicated than the aptly named SimpleCV it has a massive number of contributors on question sites like Stack Overflow [4]. Having an online community as a resource is an invaluable asset when learning an API and this is one of the great strengths of OpenCV. VXL, unlike the other two possibilities provides relatively poor and hard to navigate documentation. The time required to learn how to use an API can significantly slow down the development process of an application like our image processor. Having high quality documentation and tutorials at our disposal, as is the case with OpenCV and SimpleCV, is critical to surmounting the learning curve and its importance should not be overlooked.

TABLE 1
Comparison of Possible Technologies for Image Classification and Manipulation

	Available	Installation	Speed	Support and Documentation	Ease of Use
OpenCV	Yes	Manageable	Fast (native C++)	Excellent. Tutorial, docs, huge online community	Medium. May take some learning, offset by support/ documentation
SimpleCV	Yes	Manageable	Slow (native Python)	Good. Tutorial, docs, book available for purchase	Easy. Meant to be simple by design.
VXL	Yes	Manageable	Fast (native C++)	Poor. Hard to read docs, not much support online.	Difficult. Collection of libraries, poor docs, likely tough to learn.

Conclusion [See summary of previous discussion in Table 1]: OpenCV and SimpleCV are highly comparable

in terms of ease of use and support/documentation. SimpleCV may be easier at first, but any advantage it has is offset by the robust online community willing to support each other and answer questions that may arise when using OpenCV. VXL falls flat when compared to OpenCV and SimpleCV in those two categories. With this in mind, and when considering that speed is of critical importance, OpenCV is the clear choice of computer vision API for this project.

2.2 Runtime Environment

The high volume of images the Eclipse Megamovie project expects to collect necessitates some form of scalability for our image processor. Without a runtime environment that we can effectively scale to process hundreds of thousands or perhaps millions of images in a timeframe that meets the requirements of the larger system, the image processor's utility is near zero. We have identified three potential options to meet the needs for scaling this image processor: Docker containers, raw cloud based virtual machines (VMs), and Google Cloud Functions (GCF).

In order to determine which runtime environment will best allow us achieve the scalability necessary for the eclipse image processor several criteria will be considered. In this case they are: availability, cost, integration with our application, security, and overhead. Our group's partnership with Google for this project produces a natural inclination toward using the Google Cloud Platform for whichever of these options we end up using. Thus, all of these products have equal availability and our sponsor has made it clear that any monetary costs associated with using Google products are negligible. Because all of the options rank equally in the availability and cost categories only the other criteria will be discussed below.

As the Google Cloud Platform supports all three possible technologies we are considering, integration with our eclipse image processor would appear to be essentially equal across all three options. However, this is not the case. Using cloud based VMs or Docker Containers allows us to encapsulate our application in a manner that allows for rapid scaling in addition to easy retrieval and transmission of images and data [5], [6]. GCF on the other hand uses an event based microservice approach to app deployment [6]. Instead of containing our application, GCF is a Javascript function that would call our eclipse image processor whenever an event is triggered [6]. In this case the event would be a photo upload to the Eclipse Megamovie website. Rather than encapsulating our application GCF would spawn a child process, the eclipse image processor's executable, each time it was called. This results in a more complex deployment strategy because we would need to determine how to retrieve, store, and transmit data without access to a file system.

Security is of paramount importance when deploying a web based application that will be handling a high volume of uploads and downloads. User information being compromised as the result of a vulnerability in our application would inevitably harm participation in any future projects similar to this one. Fortunately, the Google Cloud Platform has a strong commitment to security [6]. Google Compute Engine VMs encrypt all

of their data as it is stored and transmitted [6]. This is an enormous advantage in terms of security because it allows us to focus on developing our application's functionality rather than worrying about securing it. Docker containers are deployed on top of VMs, so they have all of the same security advantages as a Compute Engine VM. In addition to this, Docker containers are completely isolated from the rest of the VM they are running on [5], so if one were compromised the issue could be handled in a manner that does not impact any other containers. GCF has its own advantages in terms of security. Each function runs in its own execution environment [5]. Like the Docker containers, this means that all of the instances of GCF are separate from one another. GCF is also running in what Google calls a "serverless" environment [5], this means that GCF is not storing data or using hardware that could be compromised. All of the options receive high marks in the security category.

In the context of evaluating these options, overhead refers to the amount of infrastructure that would need to be maintained in order to support each option as it scales to meet the demands of a high volume of uploads. For Google Compute Engine VMs, the overhead is fairly high. Each VM requires its own guest operating system, virtual memory, and virtual hardware to run an instance of our application [5], [6]. This is in contrast to Docker containers, which can run multiple instances of an application on top of a single host [5]. Each instance of the application only needs its dependencies to run. This model requires less overhead in order to scale our application, which in this case results in fewer places where things can go wrong. By far the most lightweight approach, GCF operates using a serverless model [6]. This provides advantages to us as developers because it greatly simplifies deployment. However, it presents new challenges in terms of how we retrieve and transmit data. In addition to the challenges of data management, another disadvantage of the serverless model is that it takes away our control over how much compute power we can provide to the eclipse image processor.

Conclusion [See summary of previous discussion in Table 2]: Taking all criteria into consideration, the best choice of runtime environment for our application will be Docker containers. These retain the best elements of the other options while shedding their defects.

2.3 Circle Detection

Detecting circles is an area of ongoing research in computer science and is of particular interest to this project because it is necessary for the temporal ordering and spatial alignment of the eclipse images. In order to determine the temporal ordering of the images we plan to use the relative difference in position of the centers of the sun and the moon. To spatially align the images we will be rotating them so that the visible portion of the sun is at a fixed angle relative to the center of the image. For either of these processes to be successful we need a technique capable of locating circles and their centers. We have identified three possible algorithms, Hough transforms, Blob detection, and the Ant System Algorithm.

TABLE 2
Comparison of Possible Technologies for Runtime Environments

	Available	Cost	Integration	Security	Overhead
Virtual Machines	Yes	Neglible	Good, can encapsulate app and dependencies easily	Excellent	High, VM per instance has high cost
Docker Containers	Yes	Neglible	Good, provides encapsulation	Excellent	Medium, clusters deployed on top of VMs
Google Cloud Functions	Yes	Neglible	Okay, JS can call our executable but introduces unnecessary complexity	Excellent	Low, lightweight serverless Javascript function calls

In order to select an algorithm for circle detection, three primary criteria must be considered. These are computational complexity, accuracy, and implementation difficulty. Computational complexity is of high importance because a slow algorithm or one that uses a large amount of memory jeopardizes our chances of meeting performance requirements and places strain on our runtime environment. We need this algorithm to achieve high accuracy to ensure the seamless integration of the eclipse image processor with the Eclipse Megamovie project. If our algorithm cannot correctly identify circles, our processor could not be considered functional. Lastly, an important consideration is the level of difficulty required in integrating our algorithm of choice into the eclipse image processor's codebase.

The Hough transform is a common algorithm for circle and line detection that is known to have high computational complexity and memory usage [7], [8]. This is generally disadvantageous to the performance of our eclipse image processor. Blob detection has a lower computational complexity and memory usage than the Hough transform because it does not need to store an array of all of the possible circle centers like the Hough transform does [7], [9], [10]. The least computationally complex algorithm is the Ant System Algorithm described by Chattopadhyay et al. [8]. This algorithm's complexity is determined solely by the number of pixels on the edge of shapes in the image. This is significantly lower than the complexity of other two algorithms which rely on the number of pixels in the whole image.

Accuracy is of critical importance to the success of the eclipse image processor. As such, it is necessary that the circle detection algorithm consistently detects the sun and moon in the eclipse images. Hough transforms have been shown to be very successful in this regard [11]. To this group's knowledge, blob detection has not

been shown to be effective at finding the sun and moon in eclipse images. OpenCV does include a blob detector that can filter blobs by their relative circularity and return their centers [9], [10], which meets the needs of this processor. However, the accuracy of blob detection is likely to suffer in this application because the pixels corresponding to the moon are likely to blend with the background of the image since they are both black. This results in there being no distinct blob to be detected in the crescent images. The Ant System Algorithm, while shown to be accurate [8], could fall victim to this same problem. The images that the Ant System Algorithm were tested on had white outlined shapes over a black background [8], this is not how the real world eclipse images will appear.

The final consideration in selecting a circle detection algorithm is the effort it will take to use it in the eclipse image processor. Both the Hough transform and blob detection score very highly in this category. OpenCV has implementations of these methods, documentation on how to use them, and even has code examples of their use [9], [12]. This is extremely useful and makes the implementation of the circle detection portion of the image processor much simpler. The Ant System algorithm on the other hand would need to be implemented by our team and would be based on pseudocode in the Chattopadhyay et al. paper. This presents a major disadvantage because it jeopardizes both the accuracy and performance of the algorithm. There is no guarantee that our implementation of the algorithm will work as well as the one the author's of the paper used. This also requires potentially several extra weeks of development and testing. Having to implement our own version of the Ant System Algorithm essentially eliminates it from contention as a possible circle detection algorithm.

TABLE 3
Comparison of Possible Technologies for Circle Detection

	Available	Cost	Integration
Hough Transform	High, uses a lot of memory and iterates over each pixel more than once and performs a computation each step	Excellent, known to have good accuracy and has been shown to work on eclipse images	Low, supported in OpenCV
Blob Detection	Medium, iterates over each pixel once performing computations at each step	Good, known to be accurate at finding circles and centers but hasn't been known to work with eclipses	Low, supported in OpenCV
Ant System Algorithm	Low, has complexity O(edge pixels), could be compromised by our implementation	Okay, not shown to work on real world images, could be worsened by our implementation	Extreme, we would have to write our own implementation based on pseudocode

Conclusion [See summary of previous discussion in Table 3]: While the Hough transform may struggle in terms of computational complexity, it emerges as a clear favorite when accuracy and ease of implementation

are taken into account. It is the only algorithm known to successfully detect the sun and moon in eclipse images and is supported in OpenCV.

3 Image Processor Manager

The image processor manager will be a Python application responsible for collecting/downloading user uploaded eclipse images to be processed by the image processor application, invoking the image processor with these images as input, and collecting the output of the image processor application and uploading it to Google Cloud. In this section of the technology review we consider various technologies for downloading/uploading photos from Google Cloud Storage where they will be stored, several technologies to manage our photo metadata, and different methods for incorporating parallelization into this application. We will be evaluating these various technologies on the following criteria as applicable: functionality, security, speed, ease of development, and ease of integration with the rest of the project.

3.1 Downloading/Uploading from Google Cloud Storage

There are several methods we could use to download/upload images from Google Cloud Storage. The first of these is the gsutil application. gsutil is part of the Google Cloud SDK and makes it easy to list the contents of a storage bucket, download files from a bucket, upload files to a bucket, and more from the command line [13], [14]. For our purposes, authentication using service accounts would be most suitable. Service accounts can be thought of as user accounts for applications where the applications "log in" using special authentication credentials instead of usernames/passwords. Configuring service account authentication for gsutil requires enabling a particular service account with the entire Google Cloud SDK using the gcloud application and a service account credentials file. As gsutil is a command line utility, it would need to be integrated with our app as a subprocess, using the Python `subprocess` module.

Google Cloud Storage is also accessible via a JSON Rest API. Like gsutil, this API allows users to easily list the contents of a storage bucket, download files from a bucket, upload files to a bucket, and more via HTTP requests [15]. Since our application has its own Cloud Storage buckets, authentication for the JSON API would be handled using service accounts [15]. Integrating the JSON API with our application would require use of a Python HTTP library and explicit creating of all the required API request URLs [15].

The final method we consider for interacting with Google Cloud Storage is the Google Cloud Client Library for Python. This client library is easily installable using pip (the standard Python package manager) and once installed, it allows programmatic access to Google Cloud Storage via a simple Python API [16]. Authentication for this client library is, like with the JSON API and gsutil, handled using service accounts [16]. Setting up this authentication is simple and is done by setting a special environment variable to hold the path to a credentials file, obtained from Google [16]. As this client library provides a native Python API, integrating it with out application would simply require importing the necessary modules and embedding API calls directly in the

source code via Python method calls [16].

Conclusion [See summary of the above comparison in Table 4]: The Google Cloud Client Library for Python is the best option for our needs as it is easy to install, it is secure, it provides all the required functionality, and is very simple to integrate with our application source code.

TABLE 4
Comparison of technologies for downloading/uploading from Google Cloud Storage

	Provides needed functionality?	Installation procedure	Authentication	Integration with our app	Ease of installation/ integration	Secure?
`gsutil`	Yes	`apt-get`, copy service account credentials file, setup service account as default auth method using gcloud	Service accounts	Invoke as subprocess	Medium, medium	Yes
Cloud Storage JSON API	Yes	Install Python HTTP library, copy service account credentials file	Service accounts	Make HTTP requests using some Python HTTP library	Easy, hard	Yes
Google Cloud Client Library for Python	Yes	`pip`, copy service account credentials file, set environment variable	Service accounts	Native integration into source code	Medium, easy	Yes

3.2 Storing Photo Metadata

For each photo we will store a metadata record containing various information including the filename, whether or not the photo has been processed, and if it has, output information from the image processor. The image processor manager will use these records to know which photos to request from Cloud Storage (ones that have not been processed), and these records will be updated after processing to include the output of the image processor.

We will evaluate several potential technologies to facilitate this data storage/manipulation. The first of these is Google Cloud Datastore. Datastore is a simple NoSQL database solution [17]. This means that it is schemeless and typeless, so programmers must be careful to validate that the data being inserted conforms to both the desired scheme and type. Datastore is a fully managed solution so users do not need to worry about setting up virtual machines or managing any of the other infrastructure associated with running Datastore [17]. Programmers can easily interact with Datastore via the Google Cloud Client Library for Python that is

mentioned in the previous section [18]. This library takes care of authenticating requests using service accounts [18]. The API additionally provides access to Datastore transactions [18]. This functionality is important for our purposes, as we will need to request a list of image files and mark them as pending processing before any other application can read a list containing any of these same files. This behavior is necessary to ensure that multiple nodes do not pull down the same image files to process.

The second solution we consider is Google Cloud Bigtable. Bigtable is a managed NoSQL database solution and is similar to Datastore in many ways [19]. Bigtable is tailored to applications that need to store massive amounts of data and query it very quickly [19]. As such, it is a much more heavyweight solution than Datastore. It offers users much more control than Datastore does. Bigtable allows users to create their own clusters and specify the type of hardware on which they will run [20]. Users can also create multiple tables as necessary [20]. These are features that are not offered with Datastore. As part of Google Cloud, Bigtable offers a Python API to interact with it and handle authentication using service accounts [20]. Bigtable has many of the same basic features as Datastore but offers users much more control and access to much more powerful systems. Google does not recommend Bigtable if you are working with less than 1TB of data [19]. Additionally, bigtable does not support transactions [19].

The last solution we investigate is running our own MySQL servers. This solution would serve our needs as we would be able to store all our data and MySQL does support transactions. That being said, it would be a great deal of work to use our own MySQL servers. We would have to create/manage our own virtual machines, manage database replication/backups, setup some sort of load balancer to send request to the different MySQL nodes, and configure access controls which is non-trivial to do securely. One upside of using MySQL is that it has a scheme and is typed, so this relieves some of the pressure from programmers to validate data based on scheme/type. In order to connect our Python code to MySQL, we would need to incorporate an additional MySQL connector library into our application [21].

Conclusion [See summary of the above comparison in Table 5]: Datastore will serve the needs of our project best as it is very simple to setup and manage, is simple to integrate with our application, and provides all the functionality we need. Despite not supporting transactions, Bigtable could likely be made to work, however this would just be adding more overhead in both setup and creating a transactions workaround. On top of this, Bigtable is fundamentally not the right solution for our needs. It is targeted at applications that store/query much more data than ours does.

3.3 Application Parallelization

Downloading/uploading images from Google Cloud Storage and running the image processor on a set of images are both relatively high latency operations. For that reason, it is desirable for this application to do multiple operations concurrently. Here we consider various solutions for doing this in Python. In this

TABLE 5
Comparison of technologies for storing photo metadata

	Provides needed functionality?	Setup procedure	Authentication	Integration with our app	Ease of setup	Secure?
Google Cloud Datastore	Yes	Enable Datastore in Google Cloud project	Built in: service accounts	Google Cloud Client Library for Python	Easy	Yes
Google Cloud Bigtable	Not natively, no support for transactions	Create Bigtable cluster, required tables	Built in: service accounts	Google Cloud Client Library for Python	Medium	Yes
Custom MySQL servers	Yes	Create virtual machines, install MySQL, configure network access, configure access controls, setup backup/replication credentials file, set environment variable	Requires custom setup	Requires Python-MySQL connector	Hard	Yes, if done correctly

comparison, we consider invoking the image processor binary using the builtin Python `subprocess` module. This module provides a nice Python wrapper around standard `fork`/`exec`/`waitpid`/etc. C functions.

The first solution we consider is standard multithreading using the builtin Python `threading` module. This can be used to both invoke multiple image processor processes concurrently and to make concurrent requests to Google Cloud Storage and Datastore. A common issue with use of the Python `threading` module is the Python Global Interpreter Lock (GIL) [22]. The GIL is a global lock in the CPython interpreter that allows only one thread to execute code in the interpreter at a time [22]. The GIL would not cause us many issues when invoking the image processor binary, as the Python subprocess implementation releases the GIL before calling waitpid on a particular sub-process. This means that the setting up/calling of a particular invocation of the image processor binary would be done serially, but the actual processing could be done currently as it is done outside of the Python interpreter and the GIL is released [22]. The GIL is also released when making socket calls, so while setup and pre/post-processing of Cloud Storage uploads/downloads would be serialized, the socket calls themselves could be working in parallel [22]. Implementation using the `threading` module requires explicit creation/starting/joining of `Thread` objects.

An alternative solution is to use the builtin Python `multiprocessing.Pool` class. This method sidesteps

the GIL entirely by using processes for parallelism instead of threads [23]. This method is also desirable as the API is incredibly simple - a pool of n processes can be created in a single line and then these processes can be passed a target function and data in a single other line of code [23]. The only requirement here is that all parameters passed to the process pool must be picklable [23]. This would not be an issue for us as our data will take the form of standard Python datatypes. This approach does require creating entire new processes, however on Linux this can be done very very quickly. In practice, regardless of the application, sidestepping the GIL and implementing concurrency using processes instead of threads is almost always the faster solution in Python.

The last parallelization solution we investigate is applicable only to calls to Google Cloud Datastore. This is using batch requests instead of single requests. The Datastore API has built in batch request functionality which allows multiple queries to made in a single API call, meaning only a single HTTP request is made for all grouped the queries [24]. When possible, this is desirable over multiple concurrent requests as there is reduced overhead with establishing TCP connections. This application will never be sending/receiving large pieces of data to Datastore, so the savings associated with establishing far fewer TCP connections is highly desirable.

Conclusion [See summary of the above comparison in Table 6]: The `multiprocessing.Pool` class is the best solution for us for invoking the image processor and uploading/downloading from Google Cloud Storage, as it is very simple to integrate and will almost certainly offer higher performance than using the `threading` module. However, for making calls to Datastore, batch requests are preferable to creating concurrent requests using the `multiprocessing.Pool` class as with batch requests we only have the overhead of creating a single TCP connection for all the datastore queries. So we will selectively use *both* the `multiprocessing.Pool` class and Datastore batch requests to achieve the best performance.

TABLE 6
Comparison of technologies for application parallelism

	Applicable to	Integration Overhead
`threading`	Image processor invocation, GCS/Datastore requests	Medium: requires explicit thread creation/starting/joining, requires consciousness of shared data
`multiprocessing.Pool`	Image processor invocation, GCS/Datastore requests	Medium: simple to invoke, requires pickleable parameters, cannot rely on shared data unless we use IPC
Batch requests	Datastore requests	Low, requires coalescing multiple requests into single request

4 Eclipse Simulator

The eclipse simulator will be a standalone JavaScript module enabling users to ”preview” the eclipse. It will be designed in a stylized, 2D manner. The simulator will incorporate a time slider to allow users to simulate the eclipse in a time window spanning from 12 hours before the eclipse to 12 hours after it. As users drag the time slider, the eclipse will animate in the simulator window. The view of the eclipse which users are presented will be specific to a location that the user enters. Additionally, the time will be displayed in the simulator based on what location the user enters and the positioning of the time slider.

4.1 User Interface

The user interface for this web based simulator will use 2D animated depictions of the Sun and the Moon as they appear at a user specified time and location. Additionally, the user interface will contain background imagery such as a city or hillside landscape. There will also be a time slider, a location input, and a time display for user's to interact with or view.

For the first possible method, the most basic approach would be to utilize HTML, CSS, and JavaScript for output and input. Altering the locations of the sun and moon would be done by repositioning images of the sun and moon based on location data. While this approach can be simple, repositioning images and making the interface look appealing to the user can be difficult or inefficient.

Another method would be to utilize HTML5's Canvas graphics API. Canvas is capable of rendering graphics directly to the screen via JavaScript. This is otherwise known as ”immediate mode” graphics, which means that the rendered graphics are not saved [25]. Additionally, Canvas draws individual pixels to the screen. For this simulator, the only image objects that we need display differently depending on user input are the Sun and Moon images. Since these are relatively small compared to rest of the simulator, re-rendering the Sun and Moon most likely will not result in poor performance.

The last technology for displaying the simulator would be to use another HTML5 component called Scalar Vector Graphics (SVG). This is a shape based method for describing 2D graphics via XML. SVG utilizes objects to describe images. The browser is capable of re-rendering shapes automatically if attributes to an object are changed [26]. Adding sliders or round buttons to the simulator can also be a straightforward approach with vector graphics.

For the user interface, speed is the most important criteria. All graphics and other simulator resources will need to load in less than 500ms given a 1-10 Mbps internet connection. Additionally the animation used in the simulator should be around 60 frames per second.

Conclusion [See summary of previous discussion in Table 7]: SVG seems to be a better technology to use when creating the user interface because of the interactivity that it has to offer for users. This is due to the fact SVG can process events for each sub-element separately while Canvas must process events for the entire canvas. Additionally, since our simulator requires a few number of objects, we will see better performance benefits than compared to the Canvas or scratch approach.

TABLE 7
Comparison of Possible Technologies for User Interface

	Model	Method	Difficulty of Implementation	Performance
Implement from Scratch	Images	HTML, JavaScript, CSS	May take time to start from scratch	Dependent upon implementation, requires a lot of bandwidth to send large files
Canvas	Pixel Based	JavaScript	Built into HTML5	Smaller surface, larger number of items
SVG	Vector Based	JavaScript, CSS	Built into HTML5	Larger surface, smaller number of items

4.2 Gathering User Latitude/Longitude Information

For the eclipse simulator to be as accurate as possible, we must gather user submitted location data for our computations. There are several options to consider when implementing this. The first of which is having the user only be allowed to enter a specific latitude and longitude when entering their location. While this would be easy to do, it provides a negative user interaction since people generally do not know the specific latitude and longitude of places they will be located for the solar eclipse.

The second option would be to utilize the Google Maps API to display a map [27]. The user would then place a pin at the location that they want the simulator to be based off of. This would require a combination of HTML, CSS, and JavaScript. Displaying a map for the user to interact act with is a much better form of input than the first method. Users don't need to know the latitude and longitude of certain locations, they just need to recognize them via a map. Implementing this would be done by adding a map to the simulator, and then adding a click event function to save the latitude and longitude for when a user clicks a location on the map.

Lastly, we could utilize the Geocoding API which is built into the Google Maps API for converting addresses into geographic coordinates [28]. The user could enter a wide range of input including specific addresses, roads, zip codes, or other options. This data would then be geocoded and will result in the latitude and longitude.

Conclusion [See summary of previous discussion in Table 8]: Overall, geocoding user entered addresses will provide the best functionality. The Google Maps Geocoding API can easily convert these addresses to latitude and longitude coordinates for computing the Sun and Moon's location. While dropping a pin on a map is simple, it requires another large space for user's to interact with and loading an entire map may degrade performance.

TABLE 8
Comparison of Possible Technologies for Lat/Long Information

	Provides Needed Functionality?	User Friendly?	Ease of Setup	API
User Inputs Lat/Long	Yes	No	Very Easy	None
Drop Pin on a Map	Yes	Yes	Medium	Google Maps API
Geocoding	Yes	Yes	Medium	Google Maps Geocoding API

4.3 Displaying Sun/Moon Based on Latitude, Longitude, and Time Information

One of the trickier aspects of the eclipse simulator is altering the positions of the Sun and the Moon based on user entered location and time. The best metric for observing locations of celestial objects is altitude and azimuth. The altitude of an object is the distance it appears above the horizon, while the azimuth is angular distance along the horizon to the object [29]. It is also possible for the Sun and the Moon to be different sizes. We may need to alter the size of the Moon based on location and time while using the simulator. The first method would be to write a JavaScript implementation from scratch for computing the positions of the Sun and Moon based on location and time. This could take significant time to ensure that all equations are correct as well as confirming accurate results.

Another method would utilize a JavaScript library called SunCalc. This can be used to compute sun position, sunlight phases, moon position, and lunar phases based off location and time [30]. This backend library is based off of an article about positions of the Sun as seen from a planet [31]. The documentation is also well written, which makes it very easy to use.

For the last method, another JavaScript library exists called Ephemeris [32]. This is very similar to the previous method except that this can be used for ephemeris calculations for a wider range of celestial objects. Additionally, the documentation for this library is very lackluster compared to the SunCalc library.

Conclusion [See summary of previous discussion in Table 9]: In conclusion, the SunCalc library seems to be the best choice. There are several examples for how to use this library located on the readme of the page [30].

Additionally, utilizing this backend to compute the Sun or the Moon's altitude and azimuth at a certain location and time requires very few function calls.

TABLE 9
Comparison of Possible Technologies for Sun/Moon Display

	Ease of Implementation	External Libaries?	Integration Difficulty
Backend from Scratch	Very Hard	None	Easy - this would be a tailored solution
SunCalc Library	Easy	SunCalc	Easy
Ephemeris Library	Easy	Ephemeris	Easy

REFERENCES

[1] "Opencv," opencv.org, accessed: 2016-12-13.

[2] "Vxl homepage," vxl.sourceforge.net, accessed: 2016-12-13.

[3] "Simplecv," simplecv.org, accessed: 2016-12-13.

[4] "Stack overflow," stackoverflow.com/tags/opencv, accessed: 2016-12-13.

[5] "What is docker," www.docker.com/what-docker, accessed: 2016-12-13.

[6] "Google cloud platform," cloud.google.com, accessed: 2016-12-13.

[7] "The hough transform," homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT6/node3.html, accessed: 2016-13-13.

[8] K. C. et al., "An efficient circle detection scheme in digital images using ant system algorithm."

[9] S. Mallick, "Blob detection using opencv (python, c++)," www.learnopencv.com/blob-detection-using-opencv-python-c/, accessed: 2016-13-13.

[10] "Simpleblobdetector class reference," /docs.opencv.org/trunk/d0/d7a/classcv_1_1SimpleBlobDetector.html, accessed: 2016-13-13.

[11] K. Larisza and S. McIntosh, "The standardisation and sequencing of solar eclipse images for the eclipse megamovie project."

[12] "Hough circle transform," docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html, accessed: 2016-13-13.

[13] "gsutil tool," cloud.google.com/storage/docs/gsutil, accessed: 2016-10-12.

[14] "Cloud storage," cloud.google.com/storage/, accessed: 2016-10-12.

[15] "Google cloud storage json api overview," cloud.google.com/storage/docs/json_api/, accessed: 2016-10-12.

[16] "Cloud storage client libraries," cloud.google.com/storage/docs/reference/libraries, accessed: 2016-10-12.

[17] "Cloud datastore," cloud.google.com/datastore/, accessed: 2016-10-12.

[18] "Google cloud datastore documentation," cloud.google.com/datastore/docs/, accessed: 2016-10-12.

[19] "Cloud bigtable," cloud.google.com/bigtable/, accessed: 2016-10-12.

[20] "Cloud bigtable documentation," cloud.google.com/bigtable/docs/, accessed: 2016-10-12.

[21] "Connecting to mysql using connector/python," dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html, accessed: 2016-10-12.

[22] J. Noller, "Python threads and the global interpreter lock," jessenoller.com/blog/2009/02/01/python-threads-and-the-global-interpreter-lock, accessed: 2016-10-12.

[23] "multiprocessing - process-based "threading" interface," docs.python.org/2/library/multiprocessing.html, accessed: 2016-10-12.

[24] "Batches," googlecloudplatform.github.io/google-cloud-python/stable/datastore-batches.html, accessed: 2016-10-12.

[25] "Svg vs canvas: how to choose," msdn.microsoft.com/en-us/library/gg193983(v=vs.85).aspx, accessed: 2016-11-09.

[26] "Html5 svg," www.w3schools.com/html/html5_svg.asp, accessed: 2016-11-09.

[27] "Google maps javascript api," developers.google.com/maps/documentation/javascript/3.exp/reference, accessed: 2016-11-09.

[28] "Google maps geocoding api," developers.google.com/maps/documentation/geocoding/intro, accessed: 2016-11-09.

[29] D. M. H. C. University, "Altitude and azimut," http://www.astro.cornell.edu/academics/courses/astro201/alt_az.htm, accessed: 2016-11-09.

[30] V. Agafonkin, "Suncalc," github.com/mourner/suncalc, accessed: 2016-11-09.

[31] "Astronomy answers position of the sun," http://aa.quae.nl/en/reken/zonpositie.html, accessed: 2016-11-09.

[32] Mikhail, "Ephemeris," github.com/mivion/ephemeris, accessed: 2016-11-09.

4.2 Tech Review Changes

Before starting on the simulator, we chose a library called SunCalc for our ephemeris computations. During work on the eclipse simulator, we found that the ephemeris computations were not very accurate and a library called Ephemeris provided better results. As development increased, we realized again that Ephemeris was not as accurate as we needed it to be. We looked to find another existing library that provided more accurate results and found a library called MeeusJs. This library is much more accurate and is used in the final simulator version found on eclipsemega.movie.

5 Blog Posts

5.1 Fall Week 3

5.1.1 Jake

Jacob Fenger's Update 1

Week 3 is wrapping up and this will be the first update for our senior capstone project (CS 461) for fall term.

Plans for the upcoming week

Next week we will be having our weekly meeting with our client as well as our first meeting with our TA for the course of the school year, Vedanth Narayanan.

Progress Since Last Week

This week, we completed our project abstract as well as have gotten our document signed by our client. This github repo was also created to be used for our project.

Problems Encountered

No problems have been encountered so far in our project.

5.1.2 Bret

Progress

Iterated on project statement following feedback from our sponsor and Kirstin.

Problems

None

Plans

- We actually included a large amount of the content needed for the technology review in our initial project statement draft, so we will incorporate that into the Technology Review once that assignment is published.
- Talk with McGrath about somewhat unique characteristics of our project, follow up with our sponsor following this conversation.

5.1.3 George

Plans for the coming week

As of right now I have two important items on the agenda for next week. First, Bret and I plan to find Kevin in his office on Monday to discuss some time frames our project sponsor has in mind for the work and how those lineup with deadlines and due dates for the capstone class. Second, we have our weekly meeting with the project sponsor on Tuesday. We plan to ask what the sponsor would like in terms of us adding him to the Github repo and to discuss the conversation Bret and I have with Kevin.

Progress during this week

This week was spent working on and revising our project statement. We drafted the statement on Sunday, went over it with our sponsor on Tuesday, and met with Kirsten for feedback on Wednesday. Our final draft was successfully signed and turned in.

Problems Encountered

Being in the early stages of the project we have not yet encountered any significant difficulties.

5.2 Fall Week 4

5.2.1 Jake

Plans for the upcoming week

Next week we will be finalizing and sending our revised problem statement to our client for it to be signed. We will also be having a meeting on Tuesday with him to discuss requirements of our project. Our meeting with our TA is Monday as well. George, Bret, and I will also be meeting to draft a requirements document that will be due next Friday.

Progress Since Last Week

This week, we revised our problem statement document. In addition, I have worked a little bit on image processing with OpenCV in C++. I worked on applying a Sobel filter to images which is used for edge detection. OpenCV makes this very easy to do.

Problems Encountered

No problems have been encountered so far in our project.

5.2.2 Bret

Plans for the coming week

This week we will be completing revisions on our problem statement, getting it signed by @dakoner, and turning it in. Additionally, we will be working on our requirements document. I feel like we have a pretty good grasp on the requirements from a high level, but we will need to discuss them in detail with @dakoner in our call on Tuesday.

Progress this week

This week we synced with @dakoner regarding progress of the Google team and reviewed rotation matrices/researched quaternions at his suggestion. We also had our initial meeting with our TA which went very well. I am looking forward to working with him as we complete the rest of the project.

Problems Encountered

I did not think we encountered any problems/blockers this week.

5.2.3 George

Plans for the coming week

Next week we have several items on the agenda. First, we will have our weekly update with our TA, Vedanth, on Monday at 2:00. After that we are going to meet back up a little later in the afternoon to start work on our Requirements Document. Lastly, we have our weekly update meeting with our project sponsor on Tuesday.

In addition to these meetings, we are going to get our revised Problem Statement signed by our project sponsor. Currently the plan is to have it in his inbox by Monday morning so it is ready to be turned in Wednesday.

Progress this week

This week we received and incorporated feedback from the instructors on our problem statement. On Friday afternoon, we met as a group to discuss the feedback we received and incorporate revisions into the document.

Problems Encountered

Fortunately, we still have yet to encounter any significant blocking issues on this project.

5.3 Fall Week 5

5.3.1 Jake

Plans for the upcoming week

Next week, we will be finishing up our requirements document as well as getting it signed by our client. Additionally, we will be meeting with our TA, Vedanth, as well as our client. We will continue to solidify our requirements, especially when it comes to classifying images once we have properly aligned them.

Progress Since Last Week

This week, we finished up the revised edition of our problem statement. We have also created a draft of the requirements document which will be turned in today.

Problems Encountered

No problems have been encountered so far in our project.

5.3.2 Bret

Plans for the coming week

Next week we need to revise the initial draft of our requirements document, review this with @dakoner, make any changes he requests, and then get it signed by him/turn it in.

Progress this week

This past week we met with @dakoner, discussed the requirements document from a high level, and created an initial draft of this document.

Problems Encountered

No problems encountered this week.

5.3.3 George

Plans for the coming week

Next week we have our two weekly meetings, one with our TA and one with our project sponsor. At these meetings we hope to discuss the draft of our requirements document and receive feedback on it. After these meetings we will revise our requirements document and prepare to turn it in.

Progress this week

This week our primary push was on a draft of the requirements document. Our first draft is completed and we are going to turn that in today, Friday 10/28.

Problems Encountered

We do not have not encountered any problems.

5.4 Fall Week 6

5.4.1 Jake

Plans for the upcoming week

Next week, we will be meeting our client to catch him up with the current status of our project. Additionally we will be meeting with Vedanth, our TA, to discuss our plans as well. We will soon get started with implementing some aspects of our project, such as circle detection for eclipse images.

Progress Since Last Week

This week, we finished up the requirements document and turned it in. We also received feedback regarding our problem statement, which was very helpful.

Problems Encountered

No problems have been encountered so far in our project.

5.4.2 Bret

Plans for the coming week

- Begin work on the technical report assignments.
- Connect with @dakoner regarding the status of open sourcing the existing codebase. Also need to ask about getting ahold of JavaScript code to calculate solar/lunar position based on location. @dakoner has said he has this and we will need it for the eclipse simulator - I do not think it will be part of the open source release of the rest of the code.

Progress this week

- This week we met with @dakoner who reviewed our requirements document and was happy with the requirements we'd come up with.
- Following a request last week from @dakoner for the Eclipse Image Processor to use GPS EXIF info when available, we proposed a scheme to do this which @dakoner was happy with. We incorporated this into our requirements document.

- @dakoner shared our requirements with another Google engineer who is working on seperate part of the project. This is exciting and means we really will need to stick to the requirements as stated in our document.

Problems Encountered

No problems encountered this week.

5.4.3 George

Plans for upcoming week

Next week we will likely begin working on the technical review assignment.

Progress this week

This week was consumed with getting the requirements document written. We met a couple times to work on and finalize the requirements document. I focused on the first two sections, however, I added to the third section.

Challenges encountered

So far we have not encountered any challenges.

5.5 Fall Week 7

5.5.1 Jake

Plans for the upcoming week

- Meet with our TA, Vedanth, on Monday
- Meet with our client on Tuesday
- Begin work on JavaScript simulator. Will most likely use SVG from HTML5 for graphics/animation.

Progress Since Last Week

This week, we met with our client regarding some additional project components. Additionally, he sent us some JavaScript code for computing the positions of the Sun and Moon at a given time/location. This utilizes some JavaScript back-end libraries called ”Ephemeral” and ”SunCalc”.

Problems Encountered

We received some additional requirements regarding the collection of solar eclipse images to be processed. We talked about this requirement and it does not add too much to our project for us to be worried about.

5.5.2 Bret

Plans for the coming week

- Begin work on design document.
- Begin work on JavaScript eclipse simulator. This will include the UI and solar/lunar position calculations.
  - Use code provided by @dakoner as reference/starting point.
- Revise requirements document based on call with @dakoner this week. We are picking up an additional project component, an application that collects/downloads user uploaded eclipse images to be processed by the image

processor application, invokes the image processor with these images as input, and collects the output of the image processor application and uploads it to Google Cloud.

Progress this week

- Met with @dakoner, where he expressed a desire for us to pick up the additional project component listed above.
- Acquired JavaScript code from @dakoner which computes altitude/azimuth of the Sun/Moon based on a latitude/longitude and date/time.

Problems Encountered

I was initially hesitant about picking up the extra project component listed above. After realizing that it is very similar to some work I did at my internship this summer I felt better about this. Once that work is open sourced by @dakoner, we will be able to base this new component heavily on this previous work. Our group talked all this over after the meeting with @dakoner and we feel confident that we can take on this extra project component.

5.5.3 George

Plans for the coming week

Next week we have our two weekly meetings, one with our TA and one with our sponsor. In addition to these, we need to turn in our technical review by the end of the day Monday. After these, we will likely begin discussing plans for the design document.

Progress this week

This week we focused on the technical review. We met during the week to divide up the pieces and technologies that we would be responsible for. After that, we individually worked on writing up the parts of the tech review.

Problems Encountered

We have not encountered any problems.

5.6 Fall Week 8

5.6.1 Jake

Plans for the upcoming week

- Meet with our TA, Vedanth, on Monday
- Meet with our client on Tuesday
- Continue work on the Eclipse Simulator. Look utilizing the Python ”Ephem” library for computing the time of a solar eclipse for a given location.

Progress Since Last Week

- Began work on Eclipse Simulator
  - Looked at computing altitude and azimuth using the SunCalc and Ephemeris libraries
  - Wrote code to detect the time of a solar eclipse given a solar eclipse (Uses Python Ephem library)
- Finished tech. review

- Began to work on the design document

Problems Encountered

We initially found some problems when utilizing the Ephemeris JavaScript library. When comparing the SunCalc and Ephemeris library, it seemed that the Ephemeris library's computations were taking much longer than SunCalc's. Due to poor documentation, our client helped us realize that the functions in Ephemeris were expecting GMT times while we were just utilizing the local time. This made a huge different when computing the altitude and azimuth of the sun/moon at a given location/time.

5.6.2 Bret

Plans for the coming week

- Work on design document.
- Revise requirements/tech review based on updated requirements from @dakoner.
- Continue work on eclipse simulator.

Progress this week

- Met with @dakoner, discussed computations for the eclipse simulator, specifically regarding how to. determine eclipse time for a given location. This is the time that the simulator will initialize to.
- Began working on eclipse simulator.

Problems Encountered

- Running into some issues computing eclipse time for a given location. @dakoner sent over some Python code that can do this accurately, we have been unable to reproduce this entirely with JavaScript yet though. We are working on converting the horizontal parallax of the moon to angular size right now.
- We have also been running into some issues with inconsistent computations between SunCalc and Ephemeris JS. We are currently investigating these issues.

5.6.3 George

Plans for the coming week

This week we are going to begin work on the design document. In addition we have our usual weekly meetings.

Progress this week

This week we made major progress in completing the technical review. Each of us spent a fair amount of time researching different technologies that we could use to implement different features of our project. In addition Bret and I began to plan out how we would build the simulator.

Problems Encountered

We have not encountered any problems.

5.7 Fall Week 9

5.7.1 Jake

Plans for the coming week

This next week will be finishing our design document, and making sure our client has seen it and has signed it for submission. In addition, we need to finish updating our requirements document as some of our requirements have changed recently. While we will be removing some requirements, we also need to add some. This won't cause many problems for us in terms of workload as everything is still pretty much the same. We will also need to plan how we want to do the final presentation. We will most likely use Powerpoint for the slides, and some sort of recording software/microphone to talk over them.

Progress this week

This past week, we met up to finalize what we were going to do for the design document. We had a little trouble understanding the IEEE standard we need to use but it should not be a problem.

Problems Encountered

We found some discrepancies regarding accuracy using the SunCalc/Ephemeris libraries and are continuing to work on these parts of the simulator to ensure everything is scientifically accurate.

5.7.2 Bret

Plans for the coming week

- Present initial design document draft to @dakoner, revise based on feedback as necessary.
- Create design document final draft.
- Revise requirements document based on recent requirement changes.
- Continue work on simulator - use existing percent eclipse computations to implement function that determines eclipse time for a given location.
- Begin thinking about final report and presentation.

Progress this week

- Synced as a group on high level design for project components - that way we are on the same page for the design document and can asynchronously work on our individual components.
- Will complete my part of design document this weekend.

Problems Encountered

Ran into some issues with accuracy of eclipse percentage computation using Ephemeris JavaScript. @dakoner said he thinks we are accurate enough for the time being as this value is used to compute eclipse time - the initialization time for the eclipse simulator. This means that a ¡2 minute difference (which is where we're currently at) will not be noticeable.

5.7.3 George

Plans for the coming week

This week I plan on working on the design document. My goal is to have my portion of the design document finished by Sunday night so that my team members can have a chance to review and revise the document as needed. In addition it will give us a chance to put all of our individual pieces together before we meet with our client on Tuesday and ask him to sign it.

Progress this week

This week we met a few times to go over what we wanted the design document to look like and to actually do some work on high level designs. These meetings were important because they gave us all a chance to think through design schemes and got us going in the right direction for writing the design document.

Problems Encountered

The only problem we had early in the week was trying to understand the format of the design document. How ever we talked in person with Kevin after class and now have a good idea of what to do.

5.8 Fall Week 10

5.8.1 Jake

Plans for the coming week

- Finish progress report document by Sunday
- Finish PowerPoint slides for progress report presentation by Monday
- Record progress report presentation
- Continue progress on eclipse simulator

Progress this week

This week, we finished our design document and got it signed by our client. We also made some progress regarding the front end of the eclipse simulator. This mainly includes working on the SVG portion and being able to move the Sun and Moon around given a certain percentage of the view available.

Problems Encountered

The main problem we encountered was utilizing the IEEE 1016-2009 standard for the design document. It is a very confusing document, but we managed to figure it out and write our document according to the specific specifications.

5.8.2 Bret

Plans for the coming week

- Continue working on Eclipse Simulator View - get *very basic* working eclipse simulator put together. This will not include eclipse time computations, i.e. the simulator will initialize to a fixed time, not eclipse time for the entered location. It will likely have a hard coded location for this simple first pass implementation. Code should be clean and extensible so that changes can easily be made in future weeks.

- Create progress report document, presentation, and video recording.
- Talk to @dakoner about weekly meeting schedule over break and about setting up a meeting time for winter term.

Progress this week

- Completed/turned in design document.
- Got Eclipse Simulator view code started.

Problems Encountered

Design document standard was challenging to interpret.

5.8.3 George

Plans for the coming week

This coming week is finals week. Our primary push is to finish the progress report by Wednesday. As things stand now I will have the document portion written by Sunday night so that we can all review each other's work. Then I will begin working on the presentation portion. This will be completed Monday so that we can record our presentation on Tuesday.

Progress this week

This week we finished the design document. This took a large amount of work from everyone involved. In addition to turning in the design doc on Thursday we met Friday to plan out the work for the progress work.

Problems Encountered

We encountered major difficulties in understanding what the IEEE standard we used to build the design document was asking of us.

5.9 Winter Week 1

5.9.1 Jake

Plans for the coming week

- Continue to revise and work on the eclipse simulator. I will specifically working on certain time slider components. One example of this is displaying the time slider value in the top right corner of the simulator.
- Show off new changes to our sponsor, David Konerding, on Tuesday.
- Add tick marks to the slider bar for better usability.

Progress this week

- George added a new map feature to the simulator.
- Added the ability to display the slider time values in the top right corner.

Problems encountered

- The MDL slider component doesn't have an easy way to add tick marks to the slider.
- Some time values for computing eclipse time are still off. Bret is looking into 2D linear interpolation to compute more accurate values for this. Bret may need to use 3D spherical interpolation for more accurate results.

5.9.2 Bret

Plans for the coming week

- Investigate whether or not accurate eclipse times can be estimated by interpolating the pre-computed points published by Nasa.
  - @dakoner requested we investigate this as the eclipse time that we currently compute ephemeris js is not accurate to the minute - it is up to 20mins off for some locations. This does not cause a problem visually, but if we want to show a timestamp it will cause issues.
  - The next step is to try interpolating while modeling the Earth as a sphere. I already tried modeling the U.S. as a 2d plane, but the interpolation only worked on the path of totality (where it had +/- 1 second accuracy).
- Connect with @dakoner regarding progress open-sourcing the current image processor / image processor manager code.
- Present updated requirements document to McGrath.

Progress this week

- Talked to @dakoner about open-sourcing existing image processor / image processor manager code. He is working on creating an external repo with this code so that we can all (him included) collaborate on it there.
- Completed documenting requirements changes from the end of last term. Got revised requirements document signed by @dakoner.
- Implemented basic 2d interpolation code in Python using scipy to see if we would be able to interpolate pre-computed eclipse time values from Nasa.

Problems encountered

- 2d interpolation code mentioned above did not work for locations off the path of totality.

5.9.3 George

Plans for the coming week

- Read Google Maps API docs about how to give users the ability to drop markers/pins on maps
- Implement the above functionality
- Verify that the addition of the map is what @dakoner had in mind
  - After talking with @dakoner last week we began wok to add a map, in our meeting next week we will show him how the work went and make sure it is what he requested
- Deliver the updated requirements to Kevin
- Connect with Vee over email about what we did over break and what we have planned

Progress this week

- Added an expanding and collapsing Google map to the eclipse simulator
  - Map was connected to the location search box

  - Markers appear on map in the location that the user entered
- Searches in the location search box are now restricted to the United States

Problems encountered

- The Google maps API has a documented issue where location searches cannot be restricted to groups of countries (i.e. US, Canada, and Mexico). This causes some complications with respect to the requirement that the simulator is restricted to North America.

5.10 Winter Week 2

5.10.1 Jake

Plans for the coming week

- Improve the slider for selecting eclipse time
- This includes added notches or values to show the time ranges the user can select
- Port the interpolation that Bret did to JavaScript (George will be looking into this soon)
- Update display of eclipse (View may be misleading for users)
- 2D interpolation for predicting eclipse totality times for locations in the US working very accurately. We still need to port this to JavaScript (Mentioned above), which may pose a significant challenge.
- Added a time display in the upper right corner of the simulator to show the predicted eclipse time when simulating the eclipse.
- Map for selecting locations for simulation (Instead of typing a location into the bar) almost fully working

Problems encountered

- Still waiting for image processor and image processor manager code to be open sourced for us to use
- Added notches to an mdl slider isn't the most trivial task
- Sun/moon display may be a little off when simulating the solar eclipse at certain locations

5.10.2 Bret

Plans for the coming week

- Review projection code that projects sun / moon onto view canvas. We think this may be inaccurate currently.
- Get eclipse time 2d interpolation (described below) working in JavaScript.

Progress this week

- Achieved very good (within 1 minute accuracy) results computing eclipse time using 2d interpolation with scipy. This accuracy extends across the United States, both inside and outside the path of totality, including in areas that were previously causing problems like Florida.
- Told @dakoner that this is our big development term, so we are hoping to get working on the image processor / image processor manager components of our project as soon as possible. He is planning to open source these as soon as possible.

Problems encountered

- Image processor / image processor manager code has not yet been open sourced. We spoke to @dakoner about this and conveyed that this is our big development term and that we would like to get working on them as soon as possible.

5.10.3 George

Plans for the coming week

- Find a viable way to port Bret's python sun/moon position interpolation code to JavaScript
- Port interpolation code to JavaScript, integrate into existing code where we currently use the ephemeris library

Progress this week

- Finished integration of an expanding and collapsing map into the simulator
  - Users can drop markers on the map to select a location
  - Simulator restricted to the US by both the search box and marker drops
  - Branch merged into master

Problems encountered

- Still waiting on image processor/image processor manager code to be open sourced so we can work on that piece
- We're having trouble adding in tick marks to the slider because of the Material Design Lite templates we are using for the slider

5.11 Winter Week 3

5.11.1 Jake

Plans for the coming week

- Work on sun/moon display in the normal view on the simulator
- On the zoom mode, center the sun on the screen and only show the motion of the moon
- Show the path of totality on the map

Progress this week

- Tick marks and time display are now shown on the time slider
- Map fully working on the simulator
- Zoom mode complete: which shows a closer view of the sun and moon

Problems encountered

- We met with our client, his boss, an a UX designer for the team this week who gave us some more features to work on. This adds a bit more work to our simulator, but these changes are necessary for better user enjoyment and overall presentation.

5.11.2 Bret

Plans for the coming week

- Explore options for improved Sun/Moon altitude/azimuth computations.
- Continue working on UI/feature improvements following feedback from @dakoner, @scrubskip, and Gonglue (UX designer).

Progress this week

- Altered view rendering of Sun/Moon y position in frame.
- Implemented simulator zoom mode.
- Met with @dakoner, @scrubskip, and Gonglue regarding simulator feedback.
- Updated zoom mode to center Sun in frame following feedback from @scrubskip.
- Verified that the reason the simulator looks inaccurate in locations like San Diego (where there is only a partial eclipse) is inaccurate ephemeris JS computations.
- Considered/brainstormed various methods for improving ephemeris computations.

Problems encountered

- Discovered that ephemeris JS is not accurate enough to render sun/moon position correctly.

5.11.3 George

Plans for the coming week

- Implement the requests outlined during meeting with team from Google (see progress section)
- Research and begin to implement the new path for the simulator
- We need to find a viable way to use the interpolation results in the simulator. This will require compressing data in a binary format and accessing it via the in-browser JavaScript code. We also need to generate the points that we will pass to the interpolation code to produce this data.
- We also need to find a means of fixing the math errors in ephemeris.js. This may require a port of python ephemeris of use of tools that will allow Node.js code to be run in the browser.

Progress this week

- Met with team from Google regarding their concerns and feedback about the simulator
- They would like to see a play button that allows users to watch the eclipse over a span of 2.5 hours at 16x speed.
- They have some questions about the accuracy of our renderings, this relates to the issues with ephemeris.js that are mentioned above.
- They would like to see the sky darken as the eclipse enters totality as well as some images of a halo/ray effect as the sun and moon move closer together and farther apart.
- They are hoping to change the toggle zoom feature to keep the sun centered while the moon moves through the viewport as if the camera is panning with the sun.

Problems encountered

- Still waiting on image processor/image processor manager code to be open sourced so we can work on that piece
- The ephemeris.js library is seeming less and less viable. Bret has documented a major discrepancy between it and the python ephemeris library. We need to solve this issue quickly.
- The simulator is taking much more work than anticipated and seems like a priority to the sponsors, this may result in a re-evaluation of the project scope.

5.12 Winter Week 4

5.12.1 Jake

Plans for the coming week

- Dive into the visual representation that we have for the Sun/Moon on the simulator
- Change background colors based on eclipse percentage at a certain time
- A state of totality should result in a very dark background and hills
- Make the front end a bit more streamlined (Prettier)
- Mobile friendly implementation

Progress this week

- Sun/Moon visually updated
- Eclipse totality time more accurate by changing libraries used.
- Went from Ephemeris to MeeusJs for the computations
- Functions written to compute the percentage of the eclipse at a given time
- Not fully functional yet as some discrepancy has been found between the sun/moon visual display and computed times from Xavier's website.

Problems encountered

- We found discrepancy between the sun/moon positions in the simulator versus Xavier's website for partial eclipse times
- In other words: the times he provides for a given location differ from our computed Sun/Moon positions at the same time

5.12.2 Bret

Plans for the coming week

- Begin implementing desktop UI incorporating design from Gonglue's mocks. Focus on building this in a mobile-first/friendly manner.

Progress this week

- Received Gonglue's design mocks from @dakoner.
- Proposed solutions to problems raised by previously mentioned mocks. See *Problems Encountered* section

Problems encountered

- UI mocks do not describe desired behavior on mobile.
- UI mocks show some behavior that will potentially cause issues.
- The hills are quite tall, if left as-is, the sun/moon will not become visible until they are at a non-negligible altitude.
- The sun/moon in the mocks are very large, much more so than in the current simulator, at least in wide mode. if left as-is, this will make the simulator show the eclipse starting much before it is supposed to.
- I have proposed some potential solutions for these issues to @dakoner and am currently waiting on feedback on these ideas. The ideas are:
  - To solve problem 1, set altitude=0 at a point towards the top of the hills. This would mean that the bottom of the hills correspond to altitude¡0. This should not cause any problems.
  - To solve problem 2, we could ”stretch” degrees at the altitudes around the sun. This would potentially enable us to still have a field of view where there are 80 degrees of altitude, but maintain a large sun/moon.

5.12.3 George

Plans for the coming week

- Implement the design changes suggested by Gonglue
  - Move search bar and zoom/map buttons to upper middle
  - Turn slider, play, and skip forward buttons into a unified tool bar
- Work on and complete the midterm progress report

Progress this week

- Added a play button to the simulator
  - Steps through the eclipse at variable speeds in both wide and zoom modes
- Percent eclipse calculation function added
- Began using new ephemeris library, fixed many of the accuracy issues we had

Problems encountered

- Still waiting on image processor/image processor manager code to be open sourced so we can work on that piece
- Some minor errors in the eclipse percentage calculations

5.13 Winter Week 5

5.13.1 Jake

Plans for the coming week

- Work on progress report as well as One Note document bundle for senior design course
- Wait for code to be open sourced for map with totality line overlapped
- Meeting with our client, David Konerding, on Tuesday morning

Progress this week

- User interface has been updated to reflect recommendations from UX designer at google
  - Time slider buttons and UI updated similarly to last week for map and zoom-in buttons
- Progress report and One Note document bundle have been started
- Play button will reset to the beginning if the time slider is at the end

Problems encountered

- Waiting for image processing code to be open sourced

5.13.2 Bret

Plans for the coming week

- Progress report / document revisions
- Investigate issue described in problems encountered section below. Maybe the issue is due to poor sun/moon position calculations? This would then imply there is a bug in our rendering. I really don't know...

Progress this week

- Implemented UI changes to top bar based on mocks from Gonglue.
- Asked @dakoner if it is possible to obtain new background assets to implement Gonglue's UI. We need an image where the sky is transparent, and the valleys on the hills are moved up a bit. @dakoner said he will forward this request to Gonglue.

Problems encountered

- Percent eclipse code is returning weird results - sometimes it says there is up to a 17% eclipse when there should in fact be no eclipse. This appears to be due to the value being returned from the function that computes the angular separation between the sun and moon. I am not sure what is wrong with this code though, as it is correct (according to the formulas available online) and it works for computing eclipse time (it likely would not work for this if it was in fact incorrect).

5.13.3 George

Plans for the coming week

- Work on midterm progress report assignment
  - Need to write change logs for documents
  - Meeting with Kirsten on Monday to discuss the requirements of the assignment
  - Finish final section about problems encountered
- Implement any requests for changes that come from our Tuesday meeting with sponsor

Progress this week

- Implemented the UI changes suggested by Gonglue/Justin
  - Changed the play speed selector to be a pop up menu

  - Color scheme of the control bar changed to grey/white
  - Time forward and backward buttons moved so they are both on the right of the slider
  - Control bar moved off the bottom so it is floating over the hills in the center of the screen
- Worked on progress report assignment
  - Added week by week summary of activities
  - Updated current status of project section

Problems encountered

- Still waiting on image processor/image processor manager code to be open sourced so we can work on that piece
- Some minor errors in the eclipse percentage calculations

5.14 Winter Week 6

5.14.1 Jake

Plans for the coming week

- Work on mobile UI for simulator
- Add assets given to us by google (Sun/moon/background) into simulator
- Work on darkening the screen when the eclipse is in a state of totality
- Meet with our client and TA

Progress this week

- Finished up the midterm progress report, progress report video, and One Note stuff
- Worked on the revisions for our other documents so they are up to date with the current status of our project

Problems encountered

- Waiting for image processing code to be open sourced

5.14.2 Bret

Plans for the coming week

- Improve mobile support for simulator based on mocks from Gonglue
- Improve Simulator UI (desktop and mobile) based on mocks from Gonglue

Progress this week

- Completed capstone progress report
- Synched with @dakoner about general project timeline, including the timing of expo

Problems encountered

- Did not have time to work on the project - too busy working on progress report

5.14.3 George

Plans for the coming week

- Extend support for the simulator to mobile devices
- Integrate new background imagery into simulator
- Pad simulator top so the sun and moon don't go behind the top controls

Progress this week

- Complete midterm progress report
  - Met with Kirsten do discuss circumstances regarding development hold on Image Processor/Processor Manager
  - Wrote status updates for Eclipse Simulator
  - Recorded video for the presentation portion of the report
- Revised documents from fall term

Problems encountered

- Still waiting on image processor/image processor manager code to be open sourced so we can work on that piece
- Some minor errors in the eclipse percentage calculations

5.15 Winter Week 7

5.15.1 Jake

Plans for the coming week

- Work on better color change for eclipse simulator
- Alter the display of the sun/moon as they are a bit small in the landscape view

Progress this week

- Simulator now has a working mobile version
- Browser support for simulator on Firefox/Edge working
- Met with UX designer from Google and talked about changes that we should make to the simulator

Problems encountered

- Waiting for image processing code to be open sourced

5.15.2 Bret

Plans for the coming week

- Implement UI modifications following feedback with Gonglue
- Discuss potential solution to Gonglue's feedback that the sun / moon should be larger with @dakoner
- Discuss first steps to begin working on image processor / image processor manager with @dakoner

Progress this week

- Talked with Gonglue about simulator feedback

- Devised potential solution to enable increased sun / moon size without compromising accuracy of sun / moon path through sky
- @dakoner created pull request for existing image processor code

Problems encountered

- Increasing sun / moon size (important priority for Gonglue) without compromising accuracy of sun/moon path through sky and and appearance of percentage of eclipse is non-trivial / potentially not possible.

5.15.3 George

Plans for the coming week

- Begin working with code Dek is pushing to Larisza's repo
- Discuss plans to solve the sun size issue

Progress this week

- Extended support to Mozilla Firefox
  - Fixed two issues with SVG cross browser compatibility
  - Updating the position of the sun and moon by accessing .attr() instead of the style directly
  - The calculation of the environment size was not compatible with Firefox, changed this so it works across browsers
- Bret added better support for mobile users
- Met with Gonglue to discuss his comments on this round of feedback about the simulator appearance

Problems encountered

- Need to figure out how we will make the sun appear large but still maintain a high level of scientific accuracy

5.16 Winter Week 8

5.16.1 Jake

Plans for the coming week

- Meet with our client, David Konerding, to get feedback regarding the simulator as it nears its final stages of development

Progress this week

- Implemented path of totality to be shown on the map display via the simulator. (Jake)
- Larger sun/moon display implemented in landscape mode of the simulator utilizing Lagrange polynomial interpolation
- Better version of color change implemented for eclipse display using CSS filters. (Bret)
- Minor bug fix for google maps search box (George).

Problems encountered

- Waiting for image processing code to be open sourced from our sponsor so we can begin development of the manager and image processor

5.16.2 Bret

Plans for the coming week

- Get final feedback from @dakoner, @scrubskip, and Gonglue regarding simulator and tweak as needed.
- Support @scrubskip as needed with integration of the simulator into the Eclipse Megamovie website.

Progress this week

- Implemented solution to enable larger sun/moon in the simulator (discussed in last week's post). This actually turned out to be very cool! And it uses Legrange polynomial interpolation, which automatically makes it that much cooler!
- Updated the color changing behavior during eclipses following Gonglue's feedback. This includes changing the the way that the color changing is actually done to the background - we now use CSS filters - and changing the overall simulator darkening behavior. For the latter, the simulator was changed so that instead of darkening linearly with the percentage of eclipse at a given time, the darkening happens as a function of the percentage of eclipse raised to the sixth power. This has the effect of keeping the simulator very bright right up until total eclipse occurs, when it gets very dark very quickly. This is the desired behavior - Gonglue described the look he wanted as "sudden darkness" when the eclipse occurs. This achieves this.
- Jake and George made some other progress on the simulator. Specifically, they added an eclipse path overlay to the Google Map, and fixed a bug with the search bar.

Problems encountered

- None

5.16.3 George

Plans for the coming week

- Begin working with code Dek is pushing to Larisza's repo

Progress this week

- Fix Sun and Moon size issue
  - Bret implemented his proposed solution of moving a zoomed in view along the path of the sun
- Fix bug with location search
  - When a user hit enter without selecting an autocomplete entry the search returned no results
  - Fixed the issue so now the text in the box is used to retrieve suggestions programmatically and then the first result is used to set the simulator location. If the user selects a suggestion then the previous behavior is used.
- Add a polygon to the map that displays the path of the eclipse
  - Jake used data points from NASA to overlay the path of the eclipse on the Google Map.

Problems encountered

- Waiting on Larisza Krista to accept Dek's pull request

5.17 Winter Week 9

5.17.1 Jake

Plans for the coming week

- We received some remarks from our sponsor, David, as well as the UX designer from Google, Gonglue, regarding our simulator. This had to do with the position of the sun in the sky and making it seem like the sun was just coming out of a sun rise.
- Continue development on the image processor for eclipse images. Our sponsor is curious to see if utilizing GPU virtual machines will result in a better performance than that of CPUs.
- Bret and George worked on the image processor so results get exported to an HTML file with links to the unprocessed and processed images.

Progress this week

- Figured out how to get Google Cloud VMs running and installing certain packages necessary for the image processor.
- As a team, we went through the image processor code that Bret worked on during his internship last summer. We will be working to ensure more accurate computations as well as performance.
- We have a few more components regarding the image processor that our sponsor, David, gave us to.

Problems encountered

- None

5.17.2 Bret

Plans for the coming week

- Final simulator tweaks following feedback from Gonglue (see bullet 2 below).
- Make it so that image processor will run through a batch of images and exports results to an HTML file.
  - This file should include the images in question (both unprocessed and processed). These images should be referenced as links to files in Google Cloud Storage.
- Run tests on image processor performance when using a VM with a GPU.

Progress this week

- Updated simulator following feedback from Gonglue. Made it so that the field of view scales to try to and make the sun's path through the frame fill as much horizontal space (up to 90% frame width) as possible.
- Received additional feedback on this latest simulator iteration from Gonglue - he is concernde that the sun is sometimes deceivingly low in the sky. This is a byproduct of the change above.
- Setup Google Cloud VM with existing C++ image processing from @dakoner. Got this code running.
- Met with @dakoner - he gave us a list of items he would like completed on the image processor by Tuesday.

Problems encountered

- None

5.17.3 George

Plans for the coming week

- Final Simulator tweaks based on feedback from Gonglue and team
- Export results of the image processor to a well formatted HTML file
- Discuss result of image processor timing with Dek in Tuesday meeting

Progress this week

- Set up Google Compute Engine VM for working on the image processor code
  - Ran through and documented set up procedure for cloud VM's
  - Went over C++ image processor code with Bret and Jake
- Prepped code for baseline analysis
  - Bret made some changes to the code so it produces nicely formatted code
  - Skeleton of a script/HTML template for producing output written

Problems encountered

- Google Compute Engine has some weird networking/firewall issues. Will ask Dek about it on Tuesday.

5.18 Winter Week 10

5.18.1 Jake

Plans for the coming week'

- Updated the image processor code to run on a GPU VM to see if there is a performance increase compared to the standard CPU VM

Progress this week

- Updated the simulator so it shows local time based on the location that the simulator was set to
- Simulator now also starts from the beginning
- Rough draft for poster created
- Final presentation slides started

Problems encountered

- Ran into some issues when creating GPU VMs

5.18.2 Bret

Plans for the coming week

- Create GPU VM, update image processor to have GPU mode, run it on the new VM. Evaluate speedup.
- Update Python script to run both both GPU mode and non-GPU mode and put speedup into HTML file

Progress this week

- Completed final changes to simulator.

- Created poster.
- Worked on progress report.
- Worked on presentation.

Problems encountered

- Waiting for quota increase to allow creation of GPU instances. Quota increase finally approved on Friday.

5.18.3 George

Plans for the coming week

- Add to the batch running scripts to prepare for comparison between running image processor on non-GPU and GPU instances
- Finish final winter term progress reports
- Prepare for remote work over Spring Break

Progress this week

- Built batch runner image processing script
  - Bash script runs the image processor and then calls a python script to format the output of the processor into an HTML document
- Final feature requests for simulator finished
  - Jake converted the UTC slider labels to a local 12:00 hour clock format
  - Handed the simulator over to the team at Google

Problems encountered

- Google Compute Engine's firewall acts weird on Google owned projects, causing inconveniences

5.19 Spring Week 1

5.19.1 Jake

Plans for the coming week

- Continue to work on the image processor code for improvement and refactoring
- Meet with our client
- Meet with Vee

Progress this week

- Bret worked on many parts of the simulator over spring break
- Simulator is now up on the eclipsemega.movie website
- Started refactoring the image processing code that Bret wrote during his internship
- George implemented a way to sort the columns in the generated HTML file for easy comparisons for eclipse images

Problems encountered

- None

5.19.2 Bret

Plans for the coming week

- Complete image processor refactor (it is nearly done)
- Build image processor subclass that uses GPUs
  - Includes getting OpenCV GPU bindings working - we encountered some difficulty with this previously.

Progress this week

- Simulator launched!! See it at eclipsemega.movie/simulator
- Began refactor of image processor pipeline to be a class that can easily be inherited from to build different image processor implementations.

Problems encountered

- None

5.19.3 George

Plans for the coming week

- Rework output.html table sorting to be faster
- Finish integration of new OO model of image processor pipeline

Progress this week

- Added ability for output.html table to be sorted by column on a click
  - User can sort in ascending or descending order on four different columns
  - Sorting algorithm is currently very naive because it made DOM manipulations easy, going to improve this
- Simulator integrated into eclipsemega.movie website

Problems encountered

- No problems this week

5.20 Spring Week 2

5.20.1 Jake

Plans for the coming week

- Integrate a command line parsing library for the image processor pipeline
- Meet with our sponsor for further plans regarding the image processor
- Turn in the poster draft on Monday

Progress this week

- Image processor pipeline refactored
- Poster draft revision sent to our client
- George implemented a faster sorting algorithm for the python script that interprets the metadate file

Problems encountered

- None

5.20.2 Bret

Plans for the coming week

- Update image processor to use standard command line argument parsing library
- Create image processor subclass that accepts certain image processing parameters as command line args
- Run image processor on all images that @dakoner has uploaded to Google Cloud Storage

Progress this week

- Completed image processor refactor
- Revised 1st poster draft
- Submitted expo model release
- Updated `run_imgproc_test` tool to upload processed images and html output file to Google Cloud Storage

Problems encountered

- None

5.20.3 George

Plans for the coming week

- Work more on output.html generation script to add in a cell to hold all of the circles
- Prep poster for second draft turn in

Progress this week

- Improved output.html sorting algorithm
  - Reworked sorting to use JavaScript Array.sort instead of DOM manipulation
  - Wrote a comparator for each column that parses each cell's content and performs a comparison
  - The rows are converted from an HTML collection to a JavaScript array before sorting, are sorted, then are attached back to the DOM
- Altered output.html generation script to handle new input file format
  - Changed some columns in the data table to work with new data

Problems encountered

- No problems this week

5.21 Spring Week 3

5.21.1 Jake

Plans for the coming week

- We will be taking our team photo to use on the poster board for the engineering expo
- Update our requirements document. This includes removing any mention of an image processor manager and instead add the html generation script
- Find ways to reduce noise when it comes to circle detection for the image processing

Progress this week

- Bret updated the image processor with the addition of Gflags (Google Commandline Flags), a command line library made by Google.
- @dakoner, our client, has been working on generating eclipse images via Blender to be used for testing the image processing. He asked us to run the image processor and a test set of images that he generated. He will continue to work on other ways to generate noise in the images so we can test the image processor in more ways.

Problems encountered

- N/A

5.21.2 Bret

Plans for the coming week

- Continue brainstorming ”jitter” method (see below)
- Take team photo for poster
- Update requirements to more closely match what we ended up doing, specifically with regard to the image processor

Progress this week’

- Updated image processor to use [GFlags](https://github.com/gflags/gflags) for command line arg parsing
- Added command line arguments for several Hough transform parameters following request from @dakoner
- Updated ’test’ bash script in ’`run_imgproc_test`’ to accept flags to be passed to the image processing pipeline. Additionally, these are added to the output HTML file so that it is easy to exactly reproduce a run of the tool.
- Ran image processor on all images that @dakoner has uploaded to Google Cloud Storage, forwarded results to @dakoner
- Brainstormed ’jitter’ method for small adjustments to circles found by ’cv::HoughCircles’
- Requested eclipse glasses from M and S team to hand out at expo
- Reviewed/merged pull request 16 from @scrubskip

Problems encountered

- None

5.21.3 George

Plans for the coming week

- Experiment with different parameters to the circle detection pipeline

- Examine output that our sponsor has been getting on his runs of the pipeline

Progress this week

- Fixed several issues and added features to the output generation script
  - Made the script tolerant to not having a hand labeled ground truth file
  - Made the script handle cases where no circles are found
  - Added a column so that all circles found in an image can be displayed if a user clicks on that cell
  - Added scores for both timing an accuracy that can be used as single metric to compare runs

Problems encountered

- No problems this week

5.22 Spring Week 4

5.22.1 Jake

Plans for the coming week

- Work on midterm report/design doc
- Continue work on the Simulator
- Submit our final poster on Monday (Upon approval from Kevin)

Progress this week

- Bret had to work on some simulator tweaks as the researchers at Berkeley found a few issues with the darkening of the sky during totality.
- Met with another group to finalize our poster and to do a peer review. In addition, we received usage statistic regarding the simulator for our poster.
- Revised our requirements document to reflect our updated project
- Bret found a bug regarding the image processor and some test batches of images that our client was providing us to test

Problems encountered

- None

5.22.2 Bret

Plans for the coming week

- Update design document to match current requirements
- Incorporate erosion/dilation into image processing pipeline
- Incorporate ”jitter method” into image processing pipeline
- Finalize simulator tweaks

Progress this week

- Experimented with erosion/dilation for image processor preprocessing
- Ran image processor on large set of automatically generated (using blender) images ground truth
- Fixed a couple bugs in the image processor
  - Circles were not scaled up when saved to 'metadata.txt' if the working image was smaller than the original
  - Circles rendered on PNGs with an alpha channel appeared transparent. Changed image processor to open images in 'color' mode, discarding PNG alpha channels
- Worked with Google/Berkeley team on minor set of simulator improvements
- Took team photo for poster, made final changes to poster, *got it approved by client!*
- Updated requirements to more closely match what we ended up doing, specifically with regard to the image processor, *got them approved by client!*
  - Our client treated us similar to an internal software dev team and told us what he'd like done in a bit of an ad-hoc fashion. This worked really well, it just resulted in or requirements having to evolve.

Problems encountered

- None

5.22.3 George

Plans for the coming week

- Work on revising our requirements document and getting it approved in preparation for the code freeze
- Work on final draft of the poster and get feedback from a peer review session
- Squash any last minute bugs in the image processor and output script

Progress this week

- Worked on documentation fixes and revisions
- Refined the output generation script based on sponsor feedback
- Image processor is now being run on more and different image datasets

Problems encountered

- No problems this week

5.23 Spring Week 5

5.23.1 Jake

Plans for the coming week

- Prepare for the engineering expo in two weeks
- Meet with our client on Tuesday
- Revise design document
- Help with creating a neural network for classifying eclipse images

Progress this week

- Completed WIRED article
- Bret made some tweaks to the simulator as well as added some features to the pipeline

Problems encountered

- None

5.23.2 Bret

Plans for the coming week

- Update design document to match current requirements
- Create totality image classifier using Google Cloud Vision and single layer neural network

Progress this week

- Incorporated erosion/dilation into image processing pipeline
  - Ran comparison of image processor performance using 'ImgProcPipelineBase' and 'DilationPipeline'. 'DilationPipeline' performed well, exceeding 'ImgProcPipelineBase' overall, with one outlier.
- Finalized simulator tweaks

Problems encountered

- None

5.23.3 George

Plans for the coming week

- Begin preparing demos for expo
- Refine elevator pitch
- Tweak output script as needed

Progress this week

- Fixed a sorting error in the sun diff comparator
  - The comparator was not accounting for cases where there was no ground truth, now fixed
- Submitted the final draft of our poster for printing
- Completed the WIRED review assignment

Problems encountered

- No problems this week

5.24 Spring Week 6

5.24.1 Jake

Plans for the coming week

- Update the design document
- Go to the engineering expo
- Potentially work on the classifier if time permits

Progress this week

- Worked on/finished the progress report
- Worked on the eclipse image classification by getting Keras, Tensorflow, and Google Cloud Vision working on a Google Cloud VM

Problems encountered

- None

5.24.2 Bret

Plans for the coming week

- Update design document to match current requirements
- Improve totality image classifier
  - Complete labeling larger totality/non-totality dataset
  - Update model to output a 1-hot vector instead of a binary totality/non-totality label

Progress this week

- Created totality image classifier using Google Cloud Vision and single layer neural network implemented with Keras/Tensorflow
- Created progress report

Problems encountered

- None

5.24.3 George

Plans for the coming week

- Final preparation for expo
- Prep for the IAB presentation
- Continue exploring image labeling with GCV

Progress this week

- Completed Spring Midterm Progress Report
  - Wrote progress report document
  - Recorded video presentation
- Bret got our simulator hooked up to his television for expo demos

Problems encountered

- No problems this week

5.25 Spring Week 7

5.25.1 Jake

Overall, it was a good experience and the engineering expo went excellent. Since the eclipse is happening on August 21st, there would be no need for a project next year since most of the work should be done by the eclipse. A lot of the skills I learned this term was becoming a better writer and being able to communicate better (In technical/non-technical terms). I also learned a lot more about web development and image processing. Since most of this project was done with source control, I got much better at using Git as well. As far as our client is concerned, I think he was definitely satisfied with our work and we are continuing to help him with the image classification.

5.25.2 Bret

The End (sort of)

Expo is over! I am really happy with how our project turned out. I am especially pleased with the simulator. This was an exciting project because we got to build it from scratch and take it all the way to launch. This was the first time I'd done something like that with such a visible product. I learned a lot through this process, about project management, coordinating a team, and working with remote stakeholders. I think it was a really valuable experience.

Despite the fact that expo is over and our simulator is launched, we are still working on the project, specifically on image classification. This was is supplemental to our project. It is not something that is described in our requirements and is something we decided to take on after the May 1st code freeze. This is because it sounded like an interesting project where there was an opportunity to learn a lot. I am glad we decided to take on this project component. It has been fun working with Google Cloud Vision and Keras.

If I were to do this project over again, I think the thing I would do differently, more than anything would be to make more reasonable work estimates. On many occasions I would underestimate the amount of time it would take our team to complete a certain task. This didn't end up getting in the way a whole lot, but it is definitely an area for improvement. *Under promise and over deliver!*

Anyway, I have really enjoyed working on this project this year and am proud of the work we've done.

5.25.3 George

Retrospective

If I were able to tell my fall term self something it would be that I should not worry and that the project will get done. One thing I worried about at the beginning of the term was whether or not I would be able to do any of the things that we would need to get done. What I did not realize is that this is an ongoing project and things change and I can learn to do things along the way. Working on this project gave me an extremely strong understanding of web development and using/learning how to use a 3rd party API. I see these skills being useful even though my job will not be web based. I loved the development of the simulator. I thoroughly enjoyed seeing the work produce a more and more mature product. I really did not like the document revisions that I felt slowed down progress on the technical aspects of the project. If I was our client I would be very satisfied with what we as a team produced.

All in all I am extremely thankful for this experience and for my team members. I am a better engineer after having spent the year working on this project.

COLLEGE OF ENGINEERING

Electrical Engineering & Computer Science

ECLIPSE MEGAMOVIE PROJECT

There's going to be a solar eclipse. Let's make a movie!

PROJECT OVERVIEW

The Eclipse Megamovie capstone project is focused on making several contributions to the larger Eclipse Megamovie project, run by the Google Making & Science team and scientists at UC Berkeley. These contributions are focused on two components of the Eclipse Megamovie project. The first of these is a web based eclipse simulator, which we built from scratch. The second is an eclipse image processor, which will run in Google Compute Engine. The image processor was built by adding to/modifying existing code that was open sourced by Google. This modified image processor code is available at https://github.com/lariszakrista/EMP.

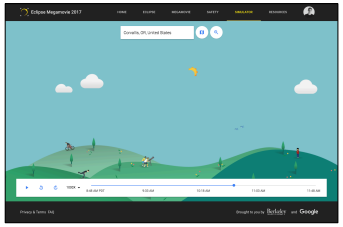


Figure 1

IMAGE PROCESSOR

The image processor is an application that analyzes photos of solar eclipses with the goal of determining whether or not a given photo shows an eclipse at totality, and if so, where the eclipse is in the photo. Figure 2 illustrates the results of running the image processor on an image of a partial eclipse - this is an image that would be discarded by the final image processor. The image processor will be used as a preprocessor in order to assemble user-submitted images into the final Eclipse Megamovie.

Development of the image processor includes creating additional tools to aid in this image processor development. One of these applications, `run_imgproc_test`, automatically pulls images from Google Cloud Storage, runs them through the image processor, and assembles the results into a nicely formatted HTML file. This HTML file is linked to the current version of the image processor source code and is uploaded to Google Cloud Storage. This makes it easy to share and reproduce image processor results. This application consists of a Python script and a calling Bash script.

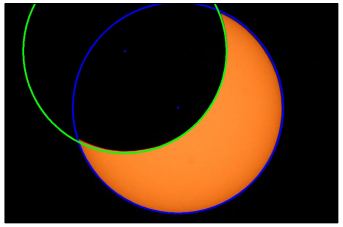


Figure 2

The image processor itself is implemented as a C++ class. This class is designed to be inherited from, so its easy for developers to change small pieces of the image processor with minimal effort. It is easy to view and share the results of these changes using the aforementioned `run_imgproc_test` tool. OpenCV is used extensively by the image processor for its image processing functions.

The image processor remains under development by Google engineers. We hope our contributions have made their jobs easier.

ECLIPSE SIMULATOR

The eclipse simulator is a web application that allows users to visualize what the eclipse will look like at any location in the United States. It was officially released on the Eclipse Megamovie website this April - see Figure 1, or the live simulator at https://eclipsemega.movie/simulator. At the time of its launch, our simulator was the second most viewed page on the Eclipse Megamovie site, behind the landing page. Today it receives 25-50 daily views each lasting nearly 2 minutes!

The simulator is built using JavaScript, HTML, and CSS, and makes use of jQuery, Material Design Lite for the Googley UI, the Google Maps JavaScript API, and MeeusJs - an open source collection of astronomical computation functions, such as Sun/Moon position.

Although we did not implement the astronomical computations in the simulator, there were several interesting computational components we did have to implement ourselves. See the *Technical Deep Dive* section for an in-depth explanation of one of these.

Technical Deep Dive

SIMULATOR VIEW TRACKING

From a UX perspective it is highly desirable for the Sun to be relatively large in the simulator window, as shown in Figure 1. This Sun size (as shown) is in fact dramatically larger than the "real" Sun size for the field of view of the simulator.

There are two ways to achieve a larger Sun size like this:

a. *Increase the Sun's angular radius*
b. *Reduce the angular field of view displayed in the simulator window*

Unfortunately, both of these methods have significant drawbacks. Increasing the Sun's angular radius results in the appearance of a partial eclipse hours before the eclipse actually begins. And reducing the angular field of view results in there being a very short time window during which the Sun is visible in the simulator. In order to overcome these issues *and* maintain the Sun's large size, we had to get a bit creative.

Our final approach was to decrease the angular field of view of the simulator and *track the Sun's path through the sky*, to keep it visible in the simulator. Notice however, that tracking the Sun's path exactly results in the Sun remaining dead center in the simulator window. Therefore, instead of tracking the Sun's path exactly, we use a wider "reference" field of view and look at the Sun's position in this field of view at the beginning, middle, and end of the simulator time window. We then use this information to track the Sun such that at these 3 points in time, the Sun appears in the same position in our narrow field of view that it would be in the wider, reference field of view.

Notice that when tracking the Sun as described above, the amount of angular movement required to track the Sun from the beginning to the middle of the time window may be much different than that from the middle to the end of the time window. Therefore, linearly tracking the Sun could result in a sharp change in tracking speed at the middle of the time window. In order to avoid this and enable smooth tracking of the sun throughout the time window, we use Lagrange polynomials to create a smooth interpolating polynomial of these three points. The simulator then tracks by adjusting its position according to this smooth polynomial.

The Project

GROUP MEMBERS

Bret Lorimore: lorimorb@oregonstate.edu
Jacob Fenger: fengerj@oregonstate.edu
George Harder: harderg@oregonstate.edu
David Konerding (sponsor)

ECLIPSE MEGAMOVIE PROJECT

The primary goal of the Eclipse Megamovie Project is to produce a high definition, time-expanded video of the total solar eclipse that will cross North America from the northwest to the southeast on August 21, 2017. The Megamovie video will be pieced together from images collected by citizen scientists at various points along the eclipse path. This will provide continuous datasets that far exceed what any one person could capture from a single location, where the longest duration of totality possible would be under three minutes. The Eclipse Megamovie, by contrast, will be about two hours long (from https://eclipsemega.movie).

Google

Oregon State UNIVERSITY

7 PROJECT DOCUMENTATION

7.1 Eclipse Simulator

The eclipse simulator is a user application designed to run in a web browser. The simulator uses the Google Maps JavaScript API so it does require a Google API Key. To obtain on of these you must sign up for a Google Cloud account. You can then create an API key from the Cloud Console on the API Manager / Credentials page. Below is a set of commands that can be executed to setup and start the simulator:

```
$ # Clone git repo
$ git clone https://github.com/hardergeorge/SeniorCapstone
$ # Enter simulator directory
$ cd SeniorCapstone/src/simulator/
$ # Create JavaScript file with your API key
$ echo 'var api_key = "<YOUR_API_KEY>";' > client-id.js
$ # Launch the simulator in your default web browser
$ xdg-open index.html
```

7.2 Image Processor

The image processor is a C++ application that will loop over a set of images of total solar eclipses and locate the eclipse in them. The application has two general modes of operation. These are the following:

- **batch mode**: this will loop through all the images without stopping, and will save images with detected circles overlayed into `output_dir`, along with a file called `metadata.txt` containing metadata collected during processing.
- **window mode**: this will loop through all the images, stopping after each one. After each image, numerous windows will be displayed including one of the original image with overlayed circles, and other intermediate images that were created during processing. Press any key to p roceed to the next image. If you wish to terminate the program before you've gotten to the end of the program, press the ESC key.

7.2.1 Parameters

The command line parameters accepted by the image processor are the following:

- `images_file` **[Required]**: text file of image filenames with no path prefix, one per line
- `mode` **[Optional, defualt window]**: batch or window (see below).
- `output_dir` **[Required in batch mode]**: directory to save image processed image and metadata files
- Hough transform parameters, see the `cv::HoughCircles` documentation for descriptions of these parameters. **These are all optional.**
  - `hough_dp` (double)
  - `hough_param1` (double)

- `hough_param2` (double)
- `hough_min_dist` (double)
- `hough_min_radius` (int)
- `hough_max_radius` (int)

7.2.2 Dependencies

The image processor requires OpenCV 3 and gflags. Installing OpenCV is not exactly trivial. If you are an Ubuntu user, Adrian Rosebrock offers a nice set of instructions here. gflags is easier to install, and can be as follows:

```
$ sudo apt-get install libgflags-dev
```

7.2.3 Building & Running the Application

The application can be built & run as follows:

```
$ # Clone the git repo
$ git clone https://github.com/bretlorimore/EMP
$ # Enter image processor directory
$ cd EMP/src/imgproc/cpp
$ # Install dependencies... See above
$ # Build
$ make
$ # Run
$ ./pipeline --images_file=/path/to.file \
  --output_dir=/path/to/dir --mode=batch [hough_params]
```

7.3 Image Processor Developer Pipeline

This tool will run the image processor and summarize the results in an HTML file. This HTML file is uploaded to Google Cloud Storage so that it is publicly available on the web. This tool will take care of downloading the images to process from Google Cloud Storage (if desired), and building the image processor. This tool assumes access to the northamericaneclipseimages Google Cloud Project. It is access to this project that grants access to the `eclipse_imgproc_output` Google Cloud Storage bucket, where the processed images and results HTML file are uploaded. If you do not have access to this Google Cloud Project and which to use another Google Cloud Storage Bucket to upload files to, simple change the value of `PROCESSED_BUCKET` on line 21 of `EMP/tools/run_imgproc_test/test`.

7.3.1 Parameters

The command line parameters accepted by the image processor developer pipeline are the following:

- `DIR` **[Required]**: The directory to use for image/data storage. The following will be saved into `DIR`:

  - If download flag is set, a clone of the `GCS_BUCKET`.
  - A directory called `output` that will contain:
    * All the processed images.
    * A metadata file that will contain the processed image names along with output information from the image processor.
    * An HTML file that includes summarizes the image processor output info.

- `GCS_BUCKET` **[Required]**: The Google Cloud Storage bucket holding the images to process. All the images in this bucket will be processed. If the download flag is set, these files will be downloaded from Google Cloud Storage. If not, they will be looked for in `DIR/GCS_BUCKET`.
- `download` **[Optional]**: A flag specifying whether or not to download images from Google Cloud Storage. *Note:* If this flag is not specified, it is assumed that the images to process are present in `DIR/GCS_BUCKET`, and that a file called images.txt exists in this directory with the full paths to the image files in it. It is not recommended to generate this file manually. First call this script once with the `download` flag, then on subsequent runs it can be omitted.
- `PIPELINE_FLAGS` **[Optional]**: is a single argument containing the flags that should be passed to the image processor. This argument should be formatted as follows: `"--flag1=value --flag2=othervalue"`.

7.3.2 Dependencies

The Image processor developer pipeline requires Python3, Jinja2, and gsutil. Python3 should come pre-installed on most systems. Find instructions for installing gsutil on the Google Cloud documentation page. Junja2 can be installed as follows:

```
$ # Install Jinja2
$ pip install Jinja2
```

7.3.3 Running the Application

The application can be run as follows:

```
$ # Clone the git repo
$ git clone https://github.com/bretlorimore/EMP
$ # Enter image processor directory
$ cd EMP/tools/run_imgproc_test
$ # Install dependencies... See above
$ ./test DIR GCS_BUCKET [download] [PIPELINE_FLAGS]
```

8 LEARNING NEW TECHNOLOGY

For most components of the project, we already had experience with the languages used and mainly relied upon documentation when we needed help. Mozilla Developer Network provided excellent help regarding JavaScript and

W3Schools is a great site for HTML and CSS. OpenCV.org had good tutorials on how to use OpenCV in C++. Google Cloud Platform documentation was also helpful in utilizing the Cloud platform service. Python.org was our primary reference for when needing to look up a specific Python functionality. Stack Overflow was occasionally useful during the year. Our sponsor, David Konerding, also provided some high level assistance which we found to be very beneficial during development.

9 What We Learned

9.1 Bret Lorimore

Overall, participating in this project was an invaluable learning experience. My learning from this project spans a wide variety of subject and domains. My biggest takeaway from a project management perspective is that things take longer than you think. I have learned this time and time again, in every internship I've had, yet it has always remained something that's difficult for me. As a general rule, I should probably estimate the time it will take to complete a task, double or triple it, and then use that value as my estimate. I know that it is always best to under promise and over deliver. I do think I was okay at this, at lease sometimes, but if you compare our original Gantt chart to what actually happened, differences are striking.

To that end, something I learned about working on projects with multiple stakeholders is the amount of time it takes to apply final revisions to a user facing product. This is also something I've experienced in internships, but I'd managed to lose some appreciation for it. We spent multiple weeks leading up to the simulator's release applying final changes / revisions. Sometimes we would present something, receive feedback that it needed to be changed, and later receive a request to change it back. Such is working with multistakeholder distributed teams.

From a more general project development perspective, I learned of the importance of maintaining a clean, documented, repository from day 1. This spring, we decided to open source our repo for this project. There are definitely some commits and general repo state in that now public git history that I am not super proud of / thrilled to have the whole world see. Live and learn I suppose.

In addition to learning about these somewhat abstract project related subjects, I learned about many other technical and non-technical subjects while completing this project. Some of these are listed below.

Technical Subjects

- Projection of 3d objects onto a 2d plane
- The correct way to animate moving objects using JavaScript (`requestAnimationFrame`)
- Performance analysis of client side JavaScript
- Description of position and apparent size of astronomical bodies
- Atmospheric light scattering
- Sky darkening behavior as a Solar Eclipse occurs
- Haar feature-based cascade classifiers

- Bilateral image filters
- Erosion and dilation image operations – how they work and why they're useful
- Keras
- Google Cloud Vision
- Composition of multiple machine learning systems
- gsutil
- Receiver operating characteristic curves

Non-Technical Subjects

- History of astronomical mathematics, i.e. Besselian Elements, etc.
- Various other mathematics history

Neither of the above lists are complete. There are certainly many subjects that are slipping my mind. The point of all this, however, is that I learned a tremendous amount on a wide variety of subjects working on this project.

Overall, I am very happy with how this project went. There is not too much that I would change if I could do it again. There are a couple things though. One thing I regret is that we did not employ the best git practices at all times, especially early in the project. I wish we had used branches more frequently, squashed commits, wrote cleaner more well documented code, and written better, more descriptive commit messages. I also wish that I had spent more time on the project throughout the entire year. For the most part I am happy with my contributions to the project, however there were a couple weeks where I did not complete as much as I would have liked. I would change this if I could do the project over again. All this said, I am proud of what we accomplished at the end of the day and I believe that completing this project was a tremendous learning experience.

9.2 Jacob Fenger

This capstone project was an excellent learning experience that allowed me to learn and try many things. This includes technical components of projects, such as programming, but also the non-technical components such as all the writing and explaining we had to do.

Technical Subjects

- C++, OpenCV
- JavaScript, HTML, CSS
- Google Cloud Compute Engine
- Image Processing Using OpenCV
- Keras with a Tensorflow Backend
- Python Generator Functions

- Astrononomical Terms (Such as azimuth and altitude of objects in space)

Non-Technical Subjects

- Writing to General and Specific Audiences
- Presenting to Audiences
- Group Collaboration
- Clear Communication Strategies
- Creation of Design Documents

Working on large projects can involve a lot more components than people initially think. At the start of this project, we all thought we knew how we were going to achieve several components of our project but realized later on that other methods were better. I also learned that many projects can involve a lot more writing. For as much time as we spent programming, we spent a similar amount of time meeting as a group or writing documents.

Managing a project can be difficult when groups are not openly communicating. We all checked our emails or group message very often and responded very quickly. This is necessary to dealing with problems as they come up in the project. Many resources are out there that can really help with this regard. Google Calendar, Hangouts, email, are just a few examples that come to mind. In my opinion, the communication between ourselves was much better than most other groups in the capstone course.

Working in a team can be difficult at times if one person seems to be doing most of the work. This can be dealt with by trying to delegate more tasks to others. This is not a one way street though, because everyone must be willing to contribute to the project. A good strategy is to be open and honest when it comes to the effort of others and to be willing to them to get their act together. A team with a good ability to communicate will always be a successful team.

There are not many things that come to mind when doing things differently. One of the main things I can think of is writing cleaner and better documented code. This can be difficult at first, but it definitely helps with readability and understanding for when others need to look at your code.

9.3 George Harder

This project came as close to an actual engineering experience as anything that I have done in college. Because of this, it constituted an unparalled learning experience and I know that I am better for it. Because this project required technical know-how, interpersonal communication, and technical writing I have found that my breadth and depth of knowledge about technical and non-technical subjects has increased. What follows is a list of topics that come immediately to mind when I consider what I learned:

Technical Subjects

- JavaScript, jQuery and HTML

- The Google Maps JavaScript API
- Basics of UX design
- Material Design Lite
- Hough Transforms
- Strategies for improving circle detection in images
- Jinja2 templating
- Google Cloud Compute Engine
- Google Cloud Storage
- Altitude, azimuth, and
- Technical writing

Non-Technical Subjects

- Schedule management
- Group collaboration and task delegation
- Communication strategies
- Strategies for group problem solving

With regard to project work, project management and working in teams I would be lying if I said I learned a ton of new information. I feel it is much more accurate to say that a lot of the things that I have learned from working in groups from classes, internships, and research experiences were reinforced by this project. While I have never worked with the same group for this length of time, I have found that my thoughts and feelings about group work from previous experiences were applicable to this project.

On thing that I have found that a group cannot function without is easy and open channels of communication. This proved to be especially true to this project. For example, if Bret found a bug in a piece of the porject I owned he never hesitated to let me know and I was always quick to respond and resolve the issue. When it came time to schedule meetings everyone was always able to respond quickly so that a time could be chosen and nobody was left hanging. The communication within our group made planning and working much easier.

Another fact of project and team management that I have found to be true and was reinforced with this project is that it is unreasonable to expect work to be distributed evenly at all times, but also important that everyone pulls their weight. Over the 8 months that we were working on this project there were times where I felt like I was doing more than others and times where I knew I was doing less. However, I never felt like one person was doing all of the work. Establishing a balanced group dynamic in which people both volunteered to do work but also understood they did not have to do everything factored into this groups success.

These are just two examples of lessons that I feel as though I had learned over the years that were reinforced by my experience over this year. While group work and project management can prove difficult even for the best groups, I feel

as though these lessons are important as I move forward and begin to work with a team of engineers everyday.

I struggle to think of things that I would change if I were to do this project all over again. I am absolutely thrilled with our finished product. I cannot find major faults in the simulator or in our development of the image processor and developer pipeline. Honestly, if I could do one thing differently it would be to spend the year with the understanding that no problems would arise because we spent so much time waiting for image processor code to be open sourced. If I went in to the project knowing that the simulator would be as cool as it ended up being I would save myself some annoyance I had early in the project about the status of the image processor code. However, this really amounts to a minor shift in thinking and I am truly happy with the results we produced.

10 APPENDIX

10.1 Essential Code Listings

10.1.1 Eclipse Simulator: Time of Maximal Eclipse Computation

```
EclipseSimulator.Model.prototype.compute_eclipse_time_and_pos = function()
{
    // Initial date/time to begin looking for eclipse time
    var date = EclipseSimulator.ECLIPSE_DAY;
    date.setUTCHours(EclipseSimulator.ECLIPSE_WCOAST_HOUR);

    // Sun/Moon angular separation
    var prev_sep = Math.PI * 4;
    var sep      = Math.PI * 2;

    // Initial time increment of 5 minutes
    var step = 1000 * 60 * 5;

    // Set time back one step, as it will be incremented
    // in the do while loop below, before its used
    var time = date.getTime() - step;

    // Doesn't matter
    var prev_time = 0;

    // Loop until we've reduced the step to a single second
    while (step >= 1000)
    {
        do
        {
            // Record previous iteration values
            prev_sep   = sep;
            prev_time  = time;

            // Update time for the current step
            time      += step;
            date.setTime(time);
```

```
            // Compute sun and moon position and angular separation
            var pos = this._compute_sun_moon_pos(date);
            sep     = EclipseSimulator.compute_sun_moon_sep(pos.sun, pos.moon);
        }
        while (sep < prev_sep);          // Loop until the
                                         // sun/moon start getting further apart

        // Back off and reduce step
        time -= (2 * step);
        step /= 2;

        // This sets the value of prev_sep
        sep = Math.PI * 2;
    }

    // Compute eclipse position
    var pos = this._compute_sun_moon_pos(date);

    // Save eclipse time in the model
    this.eclipse_time.setTime(time);

    return {
        time: this.eclipse_time,
        az:   pos.sun.az,
        alt:  pos.sun.alt,
    };
};
```

10.1.2 Eclipse Simulator: View Smooth Sun Tracking

```
EclipseSimulator.View.prototype.compute_wide_mode_altaz_centers = function()
{
    var max_time_offset_ms = (
        0.5 * 1000 * 60 * EclipseSimulator.VIEW_SLIDER_NSTEPS *
        EclipseSimulator.VIEW_SLIDER_STEP_MIN[EclipseSimulator.VIEW_ZOOM_WIDE]
    );

    // range is [-1, 1]. -1 corresponds to start of slider range, 0 corresponds
    // to time of maximal eclipse, and 1 corresponds to end of slider range.
    var time_ratio = (this.current_time.getTime() - this.eclipse_time.getTime())
                     / max_time_offset_ms;

    var poly_x = this._wide_fov_tracking_poly(
        time_ratio,
        this.sun_moon_position_ratios.x
    );
    var poly_y = this._wide_fov_tracking_poly(
        time_ratio,
        this.sun_moon_position_ratios.y
    );

    return {
        az:  this.sunpos.az  + (poly_x * this.current_fov.x),
        alt: this.sunpos.alt + (poly_y * this.current_fov.y),
    };
};
```

```
// Computes offset ratio in field of view (centered at center_angle) with fov
// width fov_width. I.e. if position_angle is at the left edge of the field of
// view, -0.5 is returned, and if position_angle is at the right edge of the
// field of view, 0.5 is returned.
EclipseSimulator.View.prototype._compute_offset_ratio = function(position_angle,
                                                                 center_angle,
                                                                 fov_width)
{
    var diff = EclipseSimulator.rad_diff(position_angle, center_angle);
    var mult = diff > 0 ? 1 : -1;
    return mult * Math.sin(Math.abs(diff)) / (2 * Math.sin(fov_width / 2));
};

// Polynomial to enable smooth tracking of sun when in wide mode.
// This is an interpolating polynomial of the following points:
//
//      p0 = (-1, ratios.start)
//      p1 = (0, 0)
//      p2 = (1, ratios.end)
//
// We compute the polynomial at a given point t by computing the Lagrange basis
// polynomials l0, l1, l2 and returning (p0.y * l0(t)) + (p1.y * l1(t)) + (p2.y * l2(t))
//
// Note: we ignore l1 since p1.y is 0
//
// For more information, see https://en.wikipedia.org/wiki/Lagrange_polynomial
//
EclipseSimulator.View.prototype._wide_fov_tracking_poly = function(t, ratios)
{
    var l0 = (t - 0) * (t - 1) / ((-1 - 0) * (-1 - 1));
    var l2 = (t + 1) * (t - 0) / (( 1 + 1) * ( 1 - 0));
    return (ratios.start * l0) + (ratios.end * l2);
};
```

10.1.3 Image Processor: Bilateral Filter Derived Pipeline

```
class BilateralPipeline : public ImgProcPipelineBase {

public:
    BilateralPipeline(int argc, char **argv) : ImgProcPipelineBase(argc, argv) {};

    virtual void preprocess(const cv::Mat &image, cv::Mat &processed)
    {
        Mat blurred, blurred2;
        std::pair<int, int> dimensions;

        time_t t = std::clock();

        // Convert image to black and white if it is not already
        if (image.channels() != 1)
        {
            cvtColor(image, processed, CV_BGR2GRAY);
        }
        else
        {
            processed = image.clone();
        }

        // Resize image to normalized size
        dimensions = getRescaledDimensions(processed, HD_MAX_W, HD_MAX_H);
        resize(processed, processed, Size(dimensions.first, dimensions.second));

        // Record BW/resized image
        this->current_image.add_intermediate_image("gray", processed);

        // Apply an unsharp mask to increase local contrast
        bilateralFilter(processed, blurred, 9, 75, 75);
        addWeighted(processed, 1.5, blurred, -0.5, 0, processed);
        this->current_image.add_intermediate_image("unsharp", processed);
```

```
        // Blur final image to reduce noise
        bilateralFilter(processed, blurred2, 9, 75, 75);
        processed = blurred2.clone();
        this->current_image.add_intermediate_image("blur", blurred2);

        t = std::clock() - t;

        // add execution time
        this->current_image.add_execution_time("preprocess",
                                               (double) t / (double) CLOCKS_PER_SEC);
    }
};
```

10.1.4 Image Processor Developer Pipeline: Metadata file parsing

```
def read_metadata(original_path, processed_path,
                  original_bucket, processed_bucket, converter):
    try:
        truth_file = open(os.path.join(original_path, TRUTH_FILE), 'r')
    except FileNotFoundError:
        truth_file = None
    if truth_file is not None:
        truth_positions = {}
        for line in truth_file:
            tokens = line.split('|')
            position = dict(sun = literal_eval(tokens[2]), moon = literal_eval(tokens[3]))
            truth_positions[tokens[0]] = position
    f = open(os.path.join(processed_path, "metadata.txt"), 'r')
    metadata_items = []
    time_score = 0
    time_count = 0
    sun_center_sum = 0
    sun_radius_sum = 0
    sun_count = 0
    for line in f.readlines():
        tokens = line.split('|')
        img_name = os.path.basename(tokens[0])
        item = dict(
            image_name = img_name,
            original   = util.gcs_url(img_name, original_bucket),
            processed  = util.gcs_url(converter.get_run_specific_filename(img_name),
                                      processed_bucket)
        )
        times = []
        comments = []
        token_count = 0
        circles = []
```

```
for token in tokens:
    if token.startswith('t'):
        tup = literal_eval(token[1:])
        times.append(tup[0] + ":\t" + str(tup[1]))
        time_count += 1
        time_score += tup[1]
    elif token.startswith('c'):
        circles.append(literal_eval(token[1:]))
    elif token_count > 1 and token != "\n":
        comments.append(token)
    else:
        None
    token_count += 1
item['times'] = times
item['comments'] = comments
item['circles'] = circles
if tokens[1].startswith('c'):
    item['found_sun'] = literal_eval(tokens[1][1:])
else:
    item['found_sun'] = "undefined"
if tokens[2].startswith('c'):
    item['found_moon'] = literal_eval(tokens[2][1:])
else:
    item['found_moon'] = "undefined"
if truth_file is not None and img_name in truth_positions:
    if truth_positions[img_name]['moon'] is not None:
        if tokens[2].startswith('c'):
            moon_center_offset, moon_radius_diff = calc_position_diff(
                literal_eval(tokens[2][1:]),
                truth_positions[img_name]['moon']
            )
            item['moon_center_diff'] = moon_center_offset
            item['moon_rad_diff'] = moon_radius_diff
```

```
            else:
                item['moon_center_diff'] = "undefined"
                item['moon_rad_diff'] = "undefined"
        else:
            item['moon_center_diff'] = "No Moon in ground truth"
            item['moon_rad_diff'] = "No Moon in ground truth"
        if truth_positions[img_name]['sun'] is not None:
            if tokens[1].startswith('c'):
                sun_center_offset, sun_radius_diff = calc_position_diff(
                    literal_eval(tokens[1][1:]),
                    truth_positions[img_name]['sun']
                )
                item['sun_center_diff'] = sun_center_offset
                item['sun_rad_diff'] = sun_radius_diff
            else:
                item['sun_center_diff'] = "undefined"
                item['sun_rad_diff'] = "undefined"
        else:
            item['sun_center_diff'] = "No Sun in ground truth"
            item['sun_rad_diff'] = "No Sun in ground truth"
        if tokens[1].startswith('c') and truth_positions[img_name]['sun'] is not None:
            sun_center_sum += sun_center_offset
            sun_radius_sum += abs(sun_radius_diff)
            sun_count += 1
    else:
        item['moon_center_diff'] = "No ground truth"
        item['moon_rad_diff'] = "No ground truth"
        item['sun_center_diff'] = "No ground truth"
        item['sun_rad_diff'] = "No ground truth"
    metadata_items.append(item)
time_score = time_score / time_count if time_count != 0 else None
sun_center_score = sun_center_sum / sun_count if sun_count != 0 else None
sun_radius_score = sun_radius_sum / sun_count if sun_count != 0 else None
return metadata_items, time_score, sun_center_score, sun_radius_score
```

10.1.5 [Supplemental] Totality Image Classifier: Simple Logistic Regression Model

```
def get_model(in_dim, out_dim):
    model = Sequential()
    model.add(Dense(out_dim, input_dim=in_dim))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['accuracy'])
    return model

def main():
    parser = argparse.ArgumentParser(description='Totality image classifier')
    parser.add_argument('--pred-type', type=str, default='onehot')
    parser.add_argument('--pred-thresh', type=float, default=None)
    args = parser.parse_args()

    # Get data
    dataset = ImageDataKFold(nfolds=10, one_hot=(args.pred_type == 'onehot'))
    in_dim, out_dim = dataset.get_dim()

    results = list()
    weights, biases = list(), list()
    for train, test in dataset.get_folds():

        x_train, y_train = train
        x_test, y_test = test

        # Build and train model
        model = get_model(in_dim, out_dim)
        model.fit(x_train, y_train, nb_epoch=10)

        # Save weights
        w, b = model.layers[0].get_weights()
        weights.append(w)
        biases.append(b)
```

```
    # Test model
    y_pred = model.predict(x_test)
    res = evaluate_predictions(y_test, y_pred, args.pred_thresh)

    results.append(res)

acc, false_pos_ratio, accept_ratio = np.average(results, axis=0)

print_results(acc, false_pos_ratio, accept_ratio,
              'Average k-fold test case performance')

# Average weights across k training iterations
weights, biases = average_weights_and_biases(weights, biases)

# Run averaged model on all images
model = get_model(in_dim, out_dim)
model.layers[0].set_weights((weights, biases))
x_test, y_test = dataset.get_all()
y_pred = model.predict(x_test)
evaluate_predictions(y_test, y_pred, args.pred_thresh,
                     title='Results for averaged model on all images')

# Print most significant weights
top10 = get_most_significant_weight_idx(weights, 10)
print_header('10 most significant weights (avg. of all k-fold model weights)')
for _, i in top10:
    print(dataset.get_feature_name(i), weights[i])
```

10.2 Images

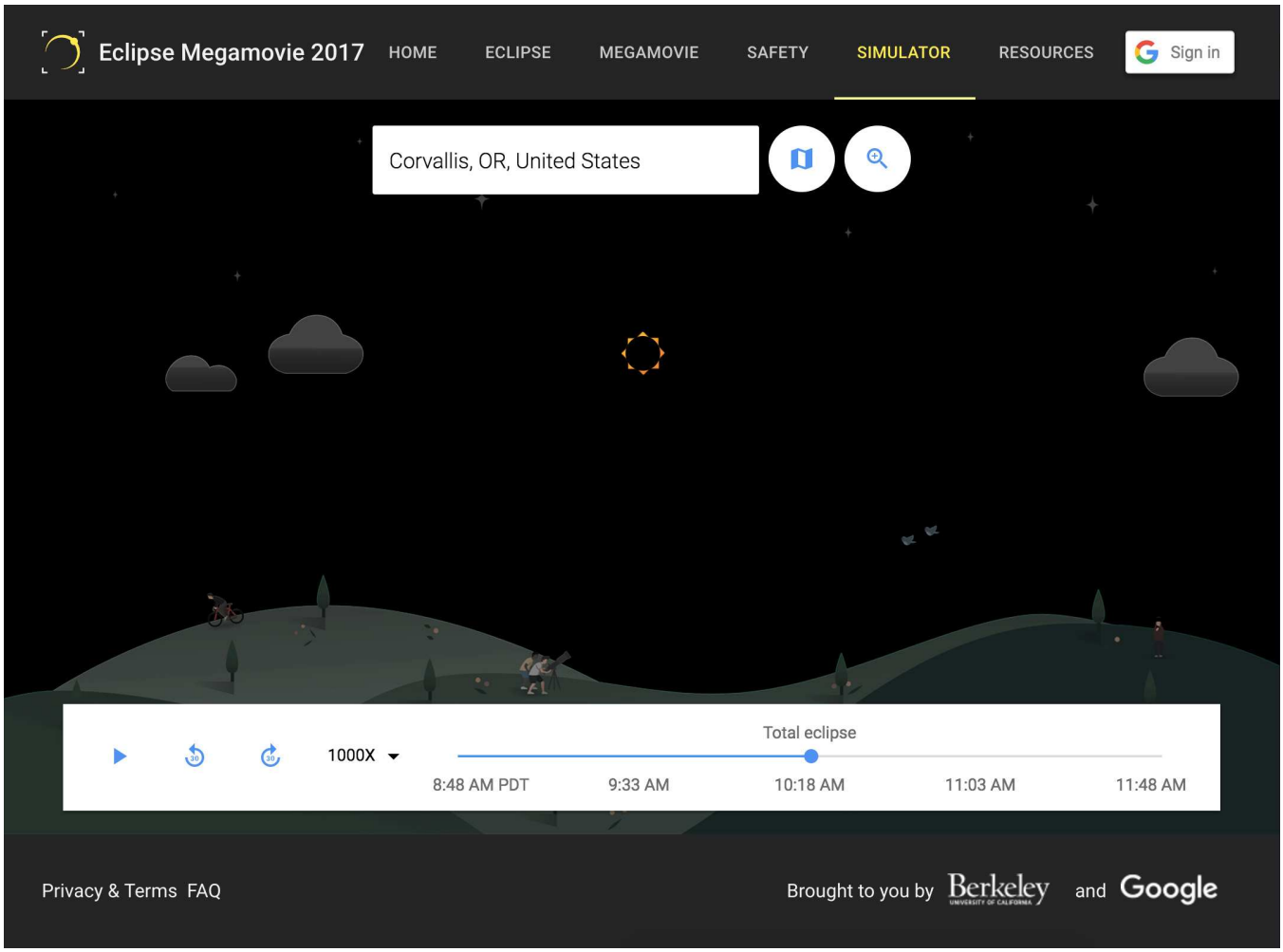


Fig. 1. Simulator in wide mode showing a total solar eclipse

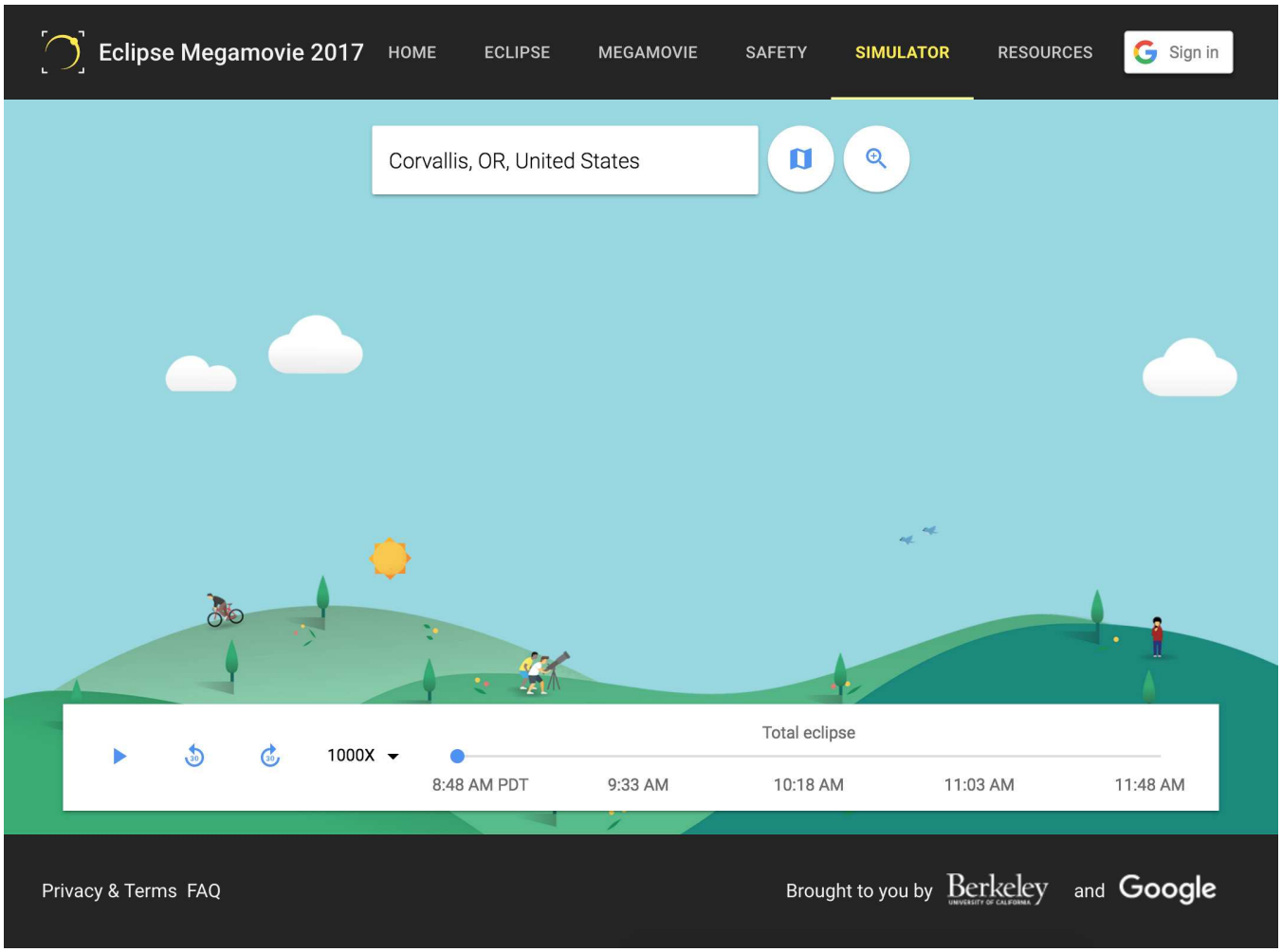


Fig. 2. Simulator in wide mode showing no eclipse

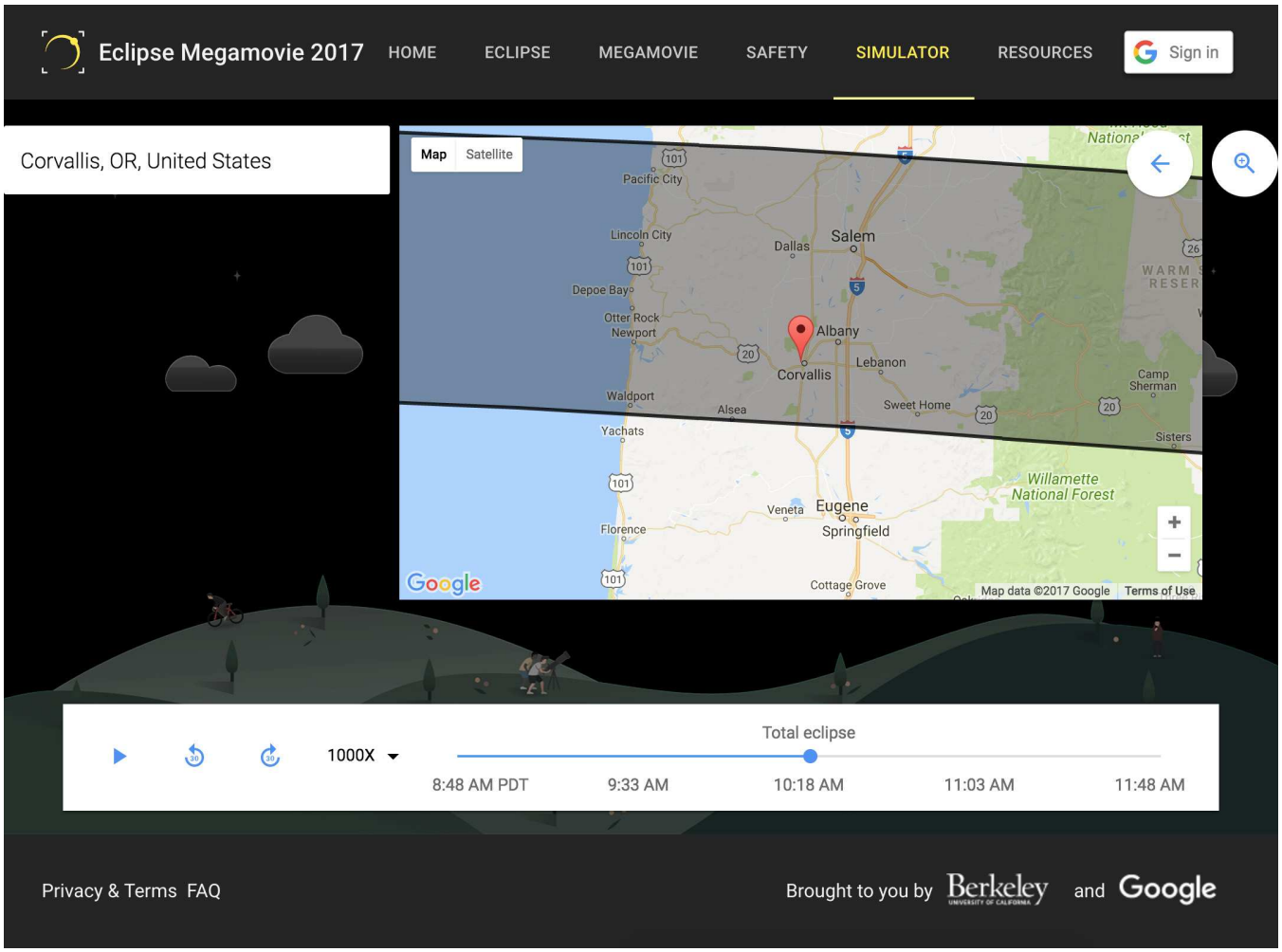


Fig. 3. Simulator with map expanded

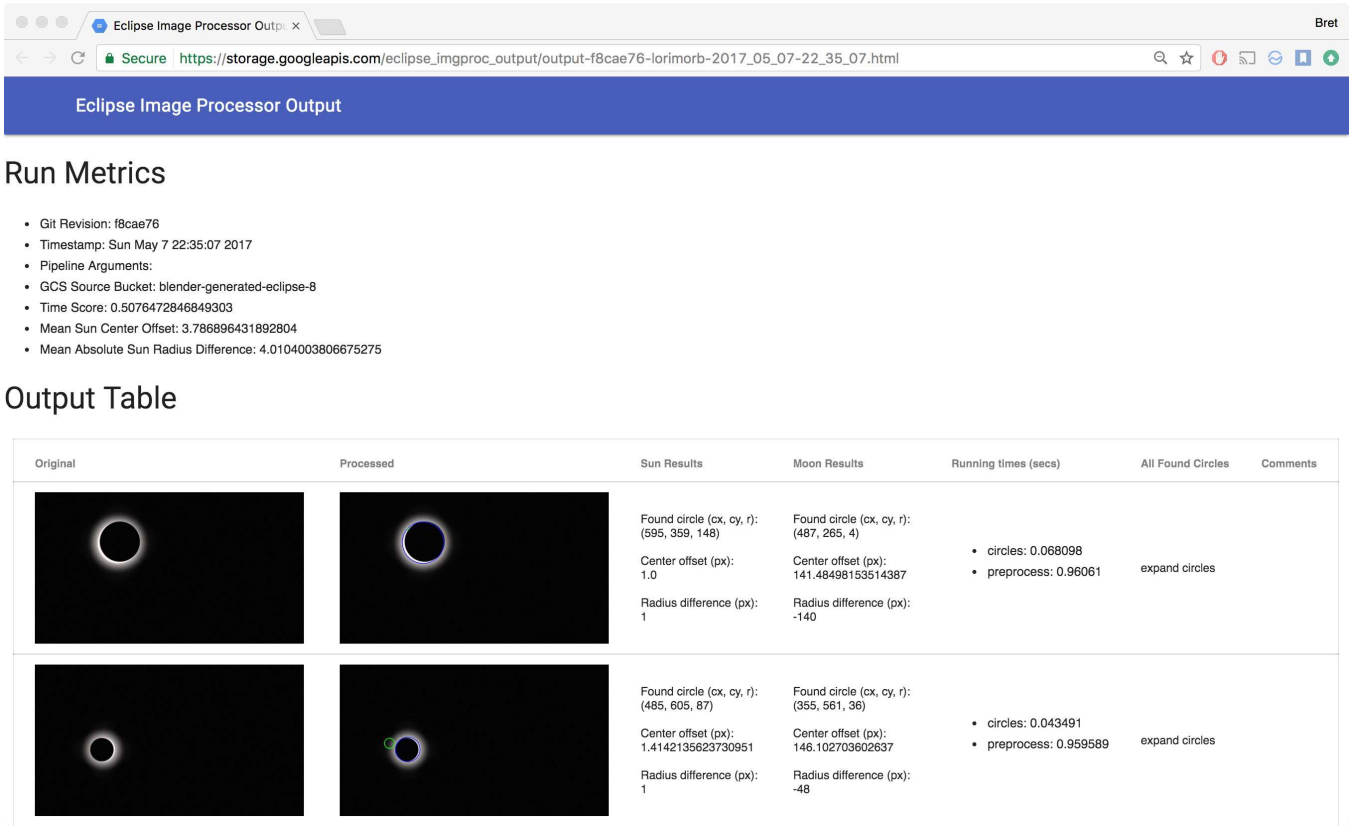


Fig. 4. Image Processor Developer Pipeline Results File