# Requirements

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

*April 24, 2017*

*Senior Capstone, Oregon State University*

**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 three distinct sub-projects, specifically, implementing/modifying an image processing algorithm facilitating the classification and alignment of solar eclipse images before they are stitched into a movie, a developer pipeline, and a location-based eclipse simulator.

| | |
|---|---|
| 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, Eclipse Image Processor Developer Pipeline, and the 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 three products: an Eclipse Image Processor, an Eclipse Image Processor Developer Pipeline, and an Eclipse Simulator. The Eclipse Image Processor will ingest images and identify the sun and moon. The Eclipse Image Developer Pipeline will download images to be processed by the image processor, build and run the image processor on these images, generate an HTML document with the results, and upload the processed images and HTML document to Google Cloud Storage. The Eclipse Simulator will provide users with a 2D visual representation of the eclipse from a specified location within a 3 hour time range.

## 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.

**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, Eclipse Image Processor Developer Pipeline, 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, Eclipse Image Processor Developer Pipeline, and the Eclipse Simulator, are all components of the larger Eclipse Megamovie Project. The Eclipse Image Processor will be a binary that processes images, finds the sun and moon in each image, and exports metdata about the images. The Eclipse Image Processor Developer Pipeline is an interface that allows easy experimentation and development with the image processor via a test script that builds and runs the processor, can download images, and uploads an HTML document to Google Cloud Storage with nicely formatted output. The Eclipse Simulator will be a standalone JavaScript module that can be added to an existing webpage.
2) The Eclipse Image Processor and Image Processor Developer Pipeline do not directly interface with the user, the pipeline will function as a developer tool and the Image Processor will eventually function as a backend system for file uploads on the Eclipse Megamovie website. The Eclipse Simulator does interface directly with the user. It should function on most modern internet browsers (Chrome, Firefox, Safari, Edge). 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, a zoom feature, and a play button.
3) This system does not interface with hardware.
4) The Eclipse Image Processor and Image Processor Developer Pipeline will be designed to run on Ubuntu 16.04. It is necessary for these applications to be compatible with this operating system because the machines that will be running the them also run 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 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 about 1.5 hours before maximum eclipse to about 1.5 hours. 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) The Eclipse Image Processor application will ingest eclipse photos and find the sun and moon in these images. In addition to identifying circles in the images the image processor will export data about its run to an output directory. This function will allow developers to imporove upon the application as it matures.
3) The Eclipse Image Processor Developer Pipeline will enable easy experimentation and testing of the image processor. This includes collecting images from Google Cloud Storage for the Image Processor to process, invoking the Image Processor with these images as input, and collecting the output of the Image Processor and uploading it in a nicely formatted HTML document to Google Cloud Storage.

### 2.3 User characteristics

1) The Eclipse Image Processor application will be used by the members of this project.
2) The Eclipse Image Processor Developer Pipeline application will be used by Google Engineers.
3) 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 1.5 before the eclipse to 1.5 hours after it.
   a) Time can be advanced via a draggable slider or clickable buttons.
4) Users will be able to ”play” the eclipse and advance through the available time domain automatically by pressing a play button.

### *3.1.2 Eclipse Image Processor*

1) The Image Processor will be compatible with Ubuntu 16.04 and will include instructions to install all dependencies and a makefile to build the binary.
2) The application will accept the following input as command line arguments:
   a) Required: images_file
      i) Absolute or relative (to the directory the binary was invoked from) path to file containing a list of image files to process. Each line of this file should correspond to one image file. Paths to image files must be absolute paths.
   b) Optional: mode
      i) Mode in which to run the image processor pipeline. Valid modes: ”window”, ”batch”
   c) Optional: output_dir
      i) Required in batch mode. Directory in which to save exported images and metadata when run in batch mode.
   d) Optional: hough_dp
      i) dp parameter to cv::HoughCircles.
   e) Optional: hough_param1
      i) param1 parameter to cv::HoughCircles.
   f) Optional: hough_param2
      i) param2 parameter to cv::HoughCircles.
   g) Optional: hough_min_dist
      i) min_dist parameter to cv::HoughCircles.
   h) Optional: hough_min_radius

i) min_radius parameter to cv::HoughCircles.

i) Optional: hough_max_radius

i) max_radius parameter to cv::HoughCircles.

3) The application will accept JPEG (.jpeg/.jpg) and PNG (.png) image files.
4) The application will write the following output to the output_dir directory:

a) metadata.txt

i) File containing one line per image processed with the following values (| separated):

A) processed_image: processed image filepath (absolute)

B) found_circle(s): circles found by the image processor, format: c(cente_x, center_y, radius)

C) execution_time(s): time(s) taken for various 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.

#### *3.1.3 Eclipse Image Processor Developer Pipeline*

1) The developer pipeline will accept the following command line arguments:

a) Required: $DIR, the directory to use for 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 summarizes the image processor output info.

2) Required: $GCS_BUCKET, the Google Cloud Storage Bucket that contains the images to process.
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
4) Optional: $PIPELINE_FLAGS, arguments to pass to the image processor when it is invoked.

### 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.
2) The simulator location will be restricted to the United States.
3) The simulator environment will darken as the eclipse progresses.
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 sun's movement.

5) The simulator will be mobile friendly.

#### *3.2.2 Eclipse Image Processor*

1) The image processor will be implemented as a class that will be easily inheritable / modifiable by developers.
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*.
3) The image processor will identify the circles of the sun/moon in the images it processes.
4) The image processor will record the number of seconds (wall clock time) needed to complete various portions of each image’s processing.

#### *3.2.3 Eclipse Image Processor Developer Pipeline*

1) The developer pipeline will be able to download images from Google Cloud Storage, if requested.
2) The developer pipeline will build and invoke the image processor on the requested images using batch mode.
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.
4) The HTML file created by the developer pipeline will summarize the data included in the metadata.txt file exported by the image processor.

### 3.3 Performance Requirements

#### *3.3.1 Eclipse Simulator*

1) All simulator resources will load in less than 700ms 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.

#### *3.3.3 Eclipse Image Processor Developer Pipeline*

1) The image processor developer pipeline will download/upload images from/to Google Cloud Storage in parallel.

# 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 |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |