Design Document

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

May 15, 2017

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 Developer Pipeline, and Eclipse Simulator. The sections of this document are broken into subsections corresponding to these components as applicable.

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 and identify circles which correspond to the Sun and Moon in the eclipse so 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

Circles corresponding to the Sun and Moon should be found for each image.

1.1.5

The image processor needs to be able to be called by an image processor developer pipeline with appropriate input data.

1.2 Image Processor Developer Pipeline

1.2.1

Pipeline must collect images from Google Cloud Storage to be run through the image processor processor.

1.2.2

Pipeline must invoke image processor with images downloaded from Google Cloud Storage.

1.2.3

Pipeline must aggregate results of image processor run in user friendly HTML file.

1.2.4

HTML file should include all information required to re-create any run of the pipeline, i.e. image processor git revision, dataset, pipeline command line parameters.

1.2.5

All assets required to display HTML file must be uploaded to Google Cloud Storage. All links in generated HTML file must correctly reference these assets.

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 view the eclipse via a draggable slider, clickable time step buttons, and a play functionality.

1.3.3

The user interface should be visually appealing, easy to navigate, and align with the theme of the larger project's site.

1.3.4

Simulator will only support locations within continental United States.

1.3.5

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: 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.9

Analytical Methods: This view is defined by whether or not we are achieving desired processing speeds for our eclipse images.

Viewpoint Source: George Harder

2.1.2 Accuracy

Concerns: 1.1.1, 1.1.4

Elements: 4.1.1, 4.1.5, 4.1.9

Analytical Methods: Accurately finding the Sun and the Moon in each eclipse image is very important as it is pertinent to the construction of the Eclipse Megamovie.

Viewpoint Source: George Harder

2.1.3 *Input and Output*

Concerns: 1.1.5

Elements: 4.1.2, 4.1.6, 4.1.7, 4.1.8

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 Developer Pipeline

2.2.1 *Image Collection*

Concerns: 1.2.1, 1.2.2

Elements: 4.2.1, 4.2.4, 4.2.5

Analytical Methods: Developer pipeline must be able to use images from a specified Google Cloud Storage bucket.

Viewpoint source: George Harder

2.2.2 *Output Generation*

Concerns: 1.2.3, 1.2.4

Elements: 4.2.1, 4.2.2, 4.2.3, 4.2.6, 4.2.7, 4.2.8, 4.2.9

Analytical Methods: Developer pipeline must produce a well formatted and visually appealing output document in HTML that includes the information specified in sections 1.2.3 and 1.2.4.

Viewpoint source: George Harder

2.2.3 *Output Uploading and Storage*

Concerns: 1.2.5

Elements: 4.2.1, 4.2.4, 4.2.5

Analytical Methods: Developer pipeline must upload the output document to Google Cloud Storage so that is easily viewable by the developer and any collaborators and must produce working URL's to all referenced assets (i.e. images and the document itself).

Viewpoint source: George Harder

2.3 Eclipse Simulator

2.3.1 *Interface*

Concerns: 1.3.2, 1.3.3

Elements: 4.3.1, 4.3.2, 4.3.3, 4.3.5, 4.3.8

Analytical Methods: Interface should be appealing to the user as well as being responsive and fast. The look and feel of the user interface will be evaluated by the sponsor and deemed complete when it meets their desired specifications.
Viewpoint source: Jacob Fenger

2.3.2 Performance

Concerns: 1.3.2, 1.3.5

Elements: 4.3.4, 4.3.7, 4.3.8,

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, 1.3.4

Elements: 4.3.4, 4.3.6

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 making it very easy for anyone to inherit from the base processing class so one can manipulate specific parts of the processing pipeline. The interface we are using for the input and output of the processing easily allows another application, such as the image processor developer pipeline, to build and run it.

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 the Sun and/or Moon in an eclipse image. The other concerns associated with the design of the image processor are near meaningless if the application is not capable of identifying circles that correspond to the Sun and/or Moon.

To address this concern, we are designing this application with accuracy as a primary goal. To 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.

To ensure a smooth interface with the image processor developer pipeline we are designing the image processor to accept a well defined set of command line arguments including a list of images containing their paths to be processed, the running mode (Window or batch), the path to an output directory (If ran in batch mode), and other optional paramaters involving the cv::HoughCircles function call. 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 the specified output directory.

3.2 Image Processor Development Pipeline

3.2.1 Image Collection [Governed by viewpoint 2.2.1]

The image processor developer pipeline seeks to provide developers with an easy to use interface to the image processor. As such, it is integral to the design of the system that it can interface with Google Cloud Storage and thereby provide the developer with access to any bucket of test images. In order to provide this functionality to developers the image processor developer pipeline will accept as an argument the name of a Google Cloud Storage bucket. The pipeline will then download the images in this bucket and run the image processor over the downloaded images. This streamlines the process of testing the image processor and gives developers fine grained but simple control over runs of the image processor.

3.2.2 Output Generation [Governed by viewpoint 2.2.2]

The image processor developer pipeline shall produce an easily readable and well formatted HTML document that details information about the run and the run itself. For this system to be effective we need to give developer the ability to easily reproduce runs and then compare the results of these runs. With this in mind, we are designing the output generation portion of the pipeline to parse the result of the image processor, which are in a text file, and display them, along with metadata like a timestamp, git hash, and scoring metrics in an HTML document so that a developer can quickly diagnose what impact their changes to the processor had on the results.

3.2.3 Output Uploading and Storage [Governed by viewpoint 2.2.3]

Our image processor developer pipeline must not only produce useful output but in addition store that output in a convenient location. To fulfill this goal the image processor developer pipeline will upload its output to Google Cloud Storage and produce a working URL to the output. This way, a developer can share the results of a run with collaborators. Designing the image processor pipeline in this way further facilitates continued development of the image processor.

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 positioned based on the time and location the simulator is set to. In addition to the Sun and Moon, the user interface will contain stylized background imagery of a hillside. The simulator's controls will be split into a top bar and bottom bar. The top bar will have a location entry box, an expandable map, and a zoom button. The bottom bar will have a time slider, time up and down buttons, a play/pause button, and a menu to select the play speed.

3.3.2 Operating Performance View [Governed by Viewpoint 2.3.2]

The simulator will have low loading times to ensure smooth performance for most users. The simulator will be responsive and quickly respond to user input.

3.3.3 Eclipse Accuracy View [Governed by Viewpoint 2.3.3]

The simulator shall display an accurate dipiction of the eclipse from 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. This accuracy shall be evaluated by our client and will be marked as adaquate when it meets their expectations.

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.

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 *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 developer pipeline without difficulty.

4.1.7 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 directory.

4.1.8 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.

4.1.9 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 [1]. We will be basing part of the image processor off of this algorithm.

4.2 Image Processor Development Pipeline

4.2.1 Bash Script

Type: System

Purpose: The image processor developer pipeline will consist, in part, as a Bash script that handles upload, download, calling the processor, and calling the output generator.

4.2.2 Python

Type: Class

Purpose: In order to parse output of the image processor and produce an HTML document the image processor developer pipeline will call a Python script that handles this functionality.

4.2.3 Jinja 2

Type: Library

Purpose: Jinja 2 is an HTML templating library for the Python language. We will use it in our output generation script to fill the HTML document with information and results from the run.

4.2.4 Google Cloud Storage

Type: System

Purpose: Google Cloud Storage is a system that we will use to store test image datasets and the results of runs of the image processor.

4.2.5 GSUtil

Type: System

Purpose: GSUtil is a command line application that handles interaction with Google Cloud Storage. We will use it in our Bash Script as an interface to Google Cloud Storage for image download and output upload.

4.2.6 HyperText Markup Language (HTML)

Type: System

Purpose: HTML is a language for formatting and aranging data. We will use it to format the results of a run of the image processor.

4.2.7 Javascript

Type: System

Purpose: Javascript provides functionality to HTML documents, we will use it to handle sorting of the rows in the table that holds the output in the output HTML document.

4.2.8 Cascading Style Sheets (CSS)

Type: System

Purpose: CSS handles the styling, look, and feel of HTML documents. In our context, it is used to make the output HTML document visually appealing and easy to read.

4.2.9 Material Design Lite (MDL)

Type: Library/Framework

Purpose: MDL is a framework that provides built in CSS for HTML elements like text, lists and tables. We will use it to enhance the CSS on our page and make the data easily readable and visually appealing.

4.3 Eclipse Simulator

4.3.1 Hypertext Markup Language (HTML)

Type: System

Purpose: HTML will be used to describe the basic view structure of the simulator.

4.3.2 Cascading Style Sheets (CSS)

Type: System

Purpose: CSS will be used to style the simulator HTML. This includes layout, color, shape, fonts, and more.

4.3.3 Scalar Vector Graphics (SVG)

Type: System

Purpose: This element will be used to position the Sun/Moon within the simulator scene.

4.3.4 MeuusJS JavaScript Library

Type: Library

Purpose: This library is used to compute the position of the Sun and Moon in the sky at a given time/position.

4.3.5 View

Type: Component

Purpose: The simulator view will be responsible for managing any dynamic components of the simulator front end. This includes manipulating the HTML DOM, updating styles, and updating SVG elements.

4.3.6 Model

Type: Component

Purpose: The model will perform complex computational tasks in the simulator. This will include providing a wrapper around the MeuusJS library, and implementing other computations based on the information returned by this library.

4.3.7 Controller

Type: Component

Purpose: The controller will coordinate interaction between the view and the model. The controller will also provide the entrypoint for the simulator.

4.3.8 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

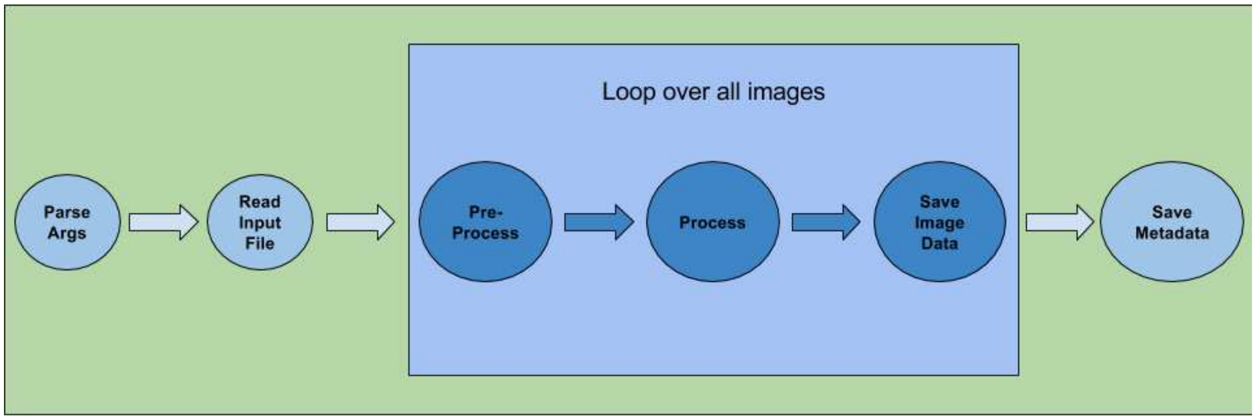

Fig. 1. Image processor basic dataflow

5.2 Image Processor Developer Pipeline

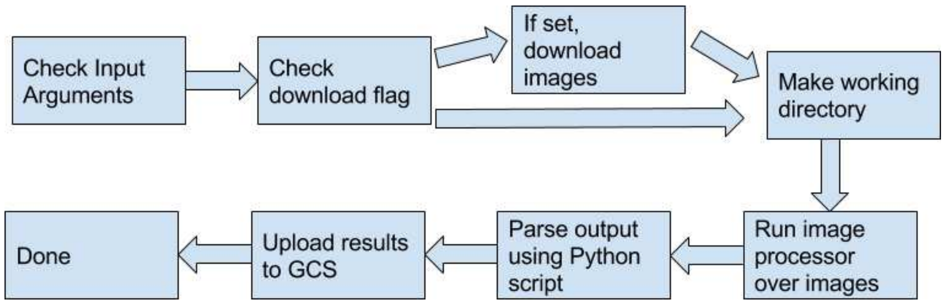

Fig. 2. Image processor developer pipeline basic dataflow

5.3 Eclipse Simulator

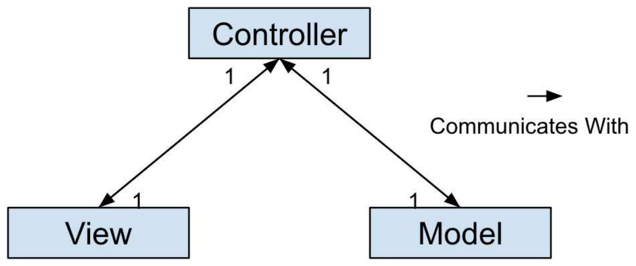

Fig. 3. Eclipse simulator MVC architecture

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 Developer Pipeline

We chose to compose the various compose the various elements of the developer using a bash script for several reasons. First, it allowed us to use the gsutil command line utility to download/upload images from Google Cloud Storage. This was desirable as gsutil already implements error handling, authentication, and concurrent downloads, meaning that these were not things we had to worry about implementing. Using bash also allowed us to build the latest version of the image processor using make, and to then easily invoke it with the input images. Bash also allowed us to use command line utilities like find, grep, and cut to dynamically create a input images file for the image processor without having to write any code.

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. The simulator is a rather complex application. One of our main goals was to design it such that iterative development was streamlined.

We decided to utilize the MVC architecture to allow for increased separation of application logic. 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, we determined that the results of both of these libraries were unsatisfactorily accurate and opted for another choice: MeuusJS. The Meuus library provides us with Sun/Moon positions that meet the requirements for accuracy of this simulator. The separation of application logic afforded by the MVC architecture means that switching between these libraries would be very easy and touch only a very small portion of the codebase.

We chose to utilize SVG for the Sun/Moon positioning as it implements a simply x, y coordinate system and is very fast.

7 Design Languages

1) UML Version 2.5

8 Appendices

8.1 Appendix I: Change History

1) November 30th, 2016 – Document Created
2) May 15th, 2017 – Document revised based on updated requirements

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.

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 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. We are using a JavaScript library, meuus.js, as our ephemeris.

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.

SEISS: Solar Eclipse Image Standardisation and Sequencing. This refers to an algorithm developed by Krista et al. [1] 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.”