Midterm Progress Report

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

February 15, 2017

CS 462 - Winter 2017

Abstract

This document describes the current state of the *North American Solar Eclipse 2017* senior capstone project. The document gives a brief overview of the project and its components, describes the current state of the project, describes problems that have been encountered throughout the term, shows some of the code that has been produced thus far, gives a week-by-week outline of progress throughout the term, and reflects over the term in the retrospectives section at the end.

TABLE OF CONTENTS

1 Project Overview

The North American Solar Eclipse 2017 Senior Capstone project is partnered with Google to build a set of applications that will assist the development of the Eclipse Megamovie Project. The overall project has been broken down into three components: the eclipse image processor, the image processor manager, and the solar eclipse simulator. Each will be individually outlined in the sections below.

1.1 Image Processor

The image processors primary activity is to quickly and consistently identify images of an eclipse at totality. The Eclipse Megamovie project will be collecting thousands of images from photographers around the country, and the image processor needs to identify the images of the eclipse at totality so that these can then be stitched into a timelapse movie. In order to make the stitching as easy as possible for the Eclipse Megamovie team, the image processor will add metadata to each processed image that includes spatial information about where the image was taken along the path of totality and temporal information about how far into totality the eclipse is.

The purpose of the Image Processor is not to process many images as quickly as possible. Instead, our goal is to be able to consistently and accurately process a single image at a time. As such, the image processor will fit into the larger project as an executable file that is called by the Image Processor Manager. This allows us to focus the image processor solely on a single goal, and leave parallelization and deployment to a different piece of the project.

1.2 Image Processor Manager

The image processor manager will be a Python application responsible for managing the image processor application. This includes collecting images from Google Cloud 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 to Google Cloud. The image processor and image processor manager will be deployed together in a single docker container to Google Container Engine Clusters (of VMs).

An important role of the image processor manager is that it will be responsible for ensuring that the compute resources on the host VMs are as saturated as possible. This means invoking multiple image processor processes concurrently, while at the same time downloading the next images to be processed and uploading the already processed images. The image processor manager will achieve this parallelism through process based concurrency in Python, as in Python, thread based concurrency is throttled by the global interpreter lock (GIL). We chose to use Python for this application as it will be much simpler to write in Python and we can sidestep any concurrency issues by using process based parallelism as mentioned above.

1.3 Eclipse Simulator

The eclipse simulator will be an independent JavaScript module that can easily be added to the existing Eclipse Megamovie webpage. This simulator will allow users to preview the eclipse. It will be a 2D depiction of what

the solar eclipse in 2017 could look like given a certain location. Users will be able to interact with a time slider that will simulate the eclipse in a time window spanning from 12 hours before the eclipse to 12 hours after it.

To help with the eclipse ephemeris computations, we will be using an external JavaScript library called MeeusJs. For the front end view for the simulator, we will be utilizing HTML5 SVG. We plan to implement a model-view-controller architecture for controlling the states of each component as well as handling the interactions. This architecture was chosen due to the ability to easily exchange a component without altering the whole design of the system. For example, if one wanted to create a whole new front end for the simulator, they would not need to rewrite the model or controller component of the system. They would simply need to ensure that the new view component can handle the interactions with the controller module.

2 Current Status of the Project

As a whole this project has moved from the pre-planning stages into the earliest development phase. The image processor and the image processor manager are both fully designed, while the eclipse simulator is farther along and has some existing proof of concept code.

When this project was being planned, the Image Processor was the main thrust of the project and the simulator was, in our minds, a secondary objective that could be completed quickly. However, the emphasis has shifted toward producing an extremely polished visualization of the eclipse that will be included on the eclipsemega.movie site. This shift in priorities as well as the delays with code becoming open-sourced has placed this piece of the project on the back burner for the time being.

2.1 Image Processor

Development of the Image Processor is currently on hold for a couple of reasons. First, the image processor relies on existing code from our project sponsor that needs to be open-sourced before we can work with it. Our project sponsor has been working on getting this code to us, however this process relies on approval from several levels on management at Google and is a slow moving process. Based on conversations with our sponsor we do not expect this code to become available until the end of February or beginning of March. Secondly and as mentioned above, there has been somewhat a shift in priorities for our sponsor.

2.2 Image Processor Manager

The image processor manager is currently on hold for the same reasons as the image processor, outlined above.

2.3 Eclipse Simulator

We currently have a working eclipse simulator that meets the specs outlined in our requirements document. This includes allowing users to simulate the eclipse at locations across the United States by entering their location either via searchbar of by selecting it on a map. This basic functionality was completed several weeks ago, however we have been working on tuning and adding additional features after presenting the initial MVP to our client.

Two new features that were requested are a zoom mode, allowing users to view a close-up simulation of the eclipse, and a ”play” feature, that allows users to watch the eclipse simulation like a video. In addition to these new features, we have done (and continue to do) a lot of work improving the UI based on feedback from our sponsor and his colleagues.

3 Problems and Possible Solutions

3.1 Ephemeris Computations *Resolved*

At the start of January we noticed that our computed eclipse times did not line up with the times computed by other (trusted) sources stated. Locations that were within the path of totality were merely seconds off the actual eclipse time, but as we deviated from the path of totality, the computed times were up to 20 minutes off. At the time, we were computing eclipse time by computing the Sun/Moon position at a series of times, and finding when they were closest together. We determined that the reason this code was not working was because the Sun/Moon positions we computed, using the EphemerisJS library were not accurate. At the time, we did not know of any more accurate JavaScript libraries for computing the positions of celestial objects. We looked into several other possible solutions to give more accurate results. One option was to perform a 2D interpolation over known data points from NASA and elsewhere to to approximate eclipse times anywhere in the country. We implemented a test script in Python to see if this was a viable solution and the results reflected that it was. We also considered setting up a basic REST service which clients could query for eclipse times and Sun/Moon positions. While continuing to evaluate these potential solutions, we became aware of another JavaScript ephemeris library called MeeusJs. We found that the Sun/Moon positions that this library computed were much more accurate than EphemerisJS. After feeding these more accurate data points into our elclipse time computation code, we achieved much more accurate results. Our eclipse times are now computed with < 1 minute error across the United States.

3.2 Percetage of Eclipse *Ongoing*

One of the desired features of the simulator is to darken as the Sun eclipses the Moon. In order to do this accurately, we need to compute the percentage of eclipse at a given time. This is somewhat non-trivial but is done using the angular separation between the Sun and Moon, and their respective angular radii. We are currently seeing a bug in this code, as it will occasionally report the Sun being up to 15% eclipsed when it looks like the Moon is not overlapping it at all. We have looked into this issue, and it is arising because at the times that the incorrect eclipse percentages are showing up, the angular separation between the Sun and Moon being computed reflects that they are in fact overlapping. There are several things that could be going on here. One, the angular separation code could be right, and the rendering of the eclipse could be innacurate. We believe this is not the case after extensive debugging, code review of our rendering code, and heuristic evaluations of the specifics of the scenario. Second, the Sun/Moon positions could be innacurate. This is not likely the case as we have been seeing good results with the MeeusJS library, as described above in 3.1. Third, there could be an error in our angular separation code. This seems unlikely, as this piece of code is critical to the eclipse

time computations we do, which work well. That being said, this does seem like it is the most likely place for the error to be of the three possibilities outlined above. The bottom line is, we have looked into this issue extensively, and we are really not sure what is going on. We will continue to explore this issue further.

4 Things Left to Do

4.1 Image Processor

As mentioned in the section 2.1, we are still waiting on certain items to be open sourced for development to begin on the image processor.

4.2 Image Processor Manager

The image processor manager is currently on hold for the same reasons as the image processor, outlined above and in section 2.1.

4.3 Eclipse Simulator

While the simulator is fully functional, we are working on the optimization of the user interface and Sun/Moon displays. The user experience designer from Google who is helping us with improving the front end will be providing us with better landscape and Sun/Moon images. As it stands right now, we are just using circles which do not provide the best looking visualization. Once this gets done, the simulator will almost be ready for public release on the eclipsemega.movie website.

Another thing for us to do is to create a darkening effect for when the Moon starts to cover the Sun. We have written functions to compute the percent of the eclipse at a certain time as well the ability to alter the background color based on the percentage passed into it. We just need to polish the eclipse percent computation a bit more to ensure the best visualization at totality.

Additionally, we are currently working on a mobile user-interface for the simulator. This involves rewiring our HTML code to ensure the interface is looks good as well as being as interactive as possible. The user experience designer who is helping us with the front end has provided a mock up of what the mobile interface should look like.

5 Interesting Code

Below is a function to compute eclipse time for a given location. This function looks at Sun/Moon separation and finds the time of minimal separation with 1 second resolution. We have tested this function with locations across the United States as input. Over all tested locations, the eclipse time was computed in under 30ms with fewer than 60 iterations of the innermost loop.

```
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 below
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);

    }        // Loop until the Sun/Moon start getting further apart
    while (sep < prev_sep);

    // 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: date,
            az:   pos.sun.az,
            alt:  pos.sun.alt,
        };
};
```

6 Screenshots

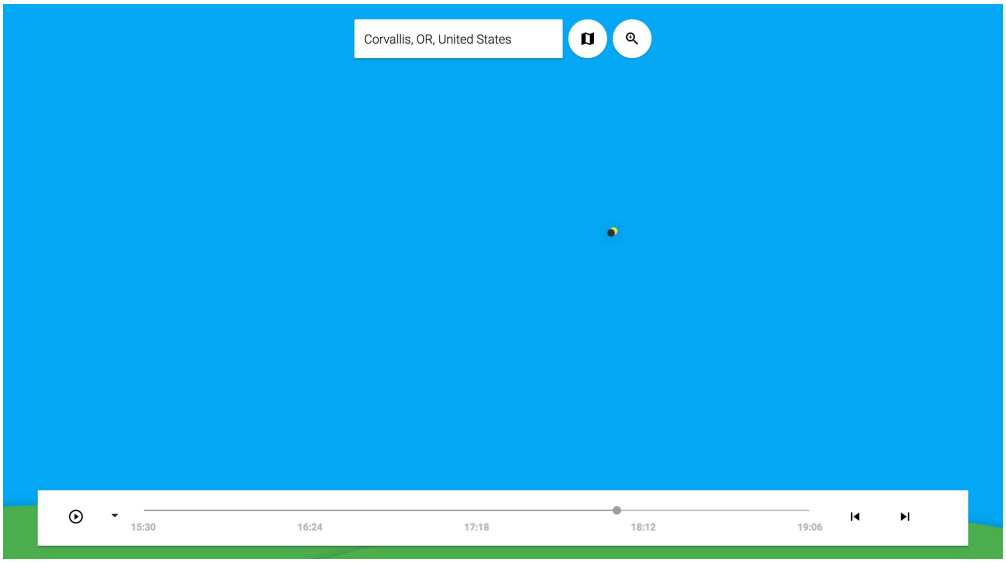


Fig. 1. Simulator wide mode

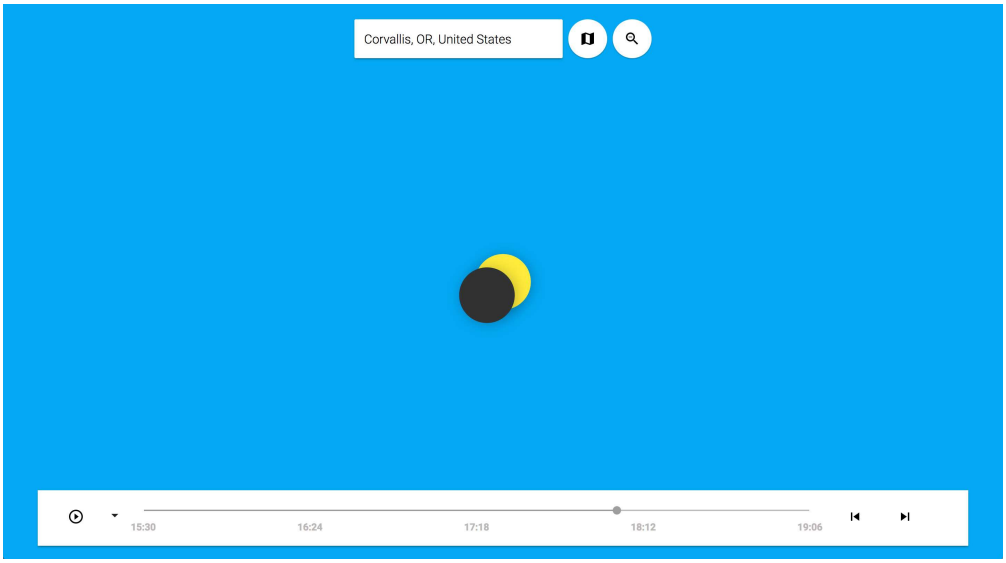


Fig. 2. Simulator zoom mode

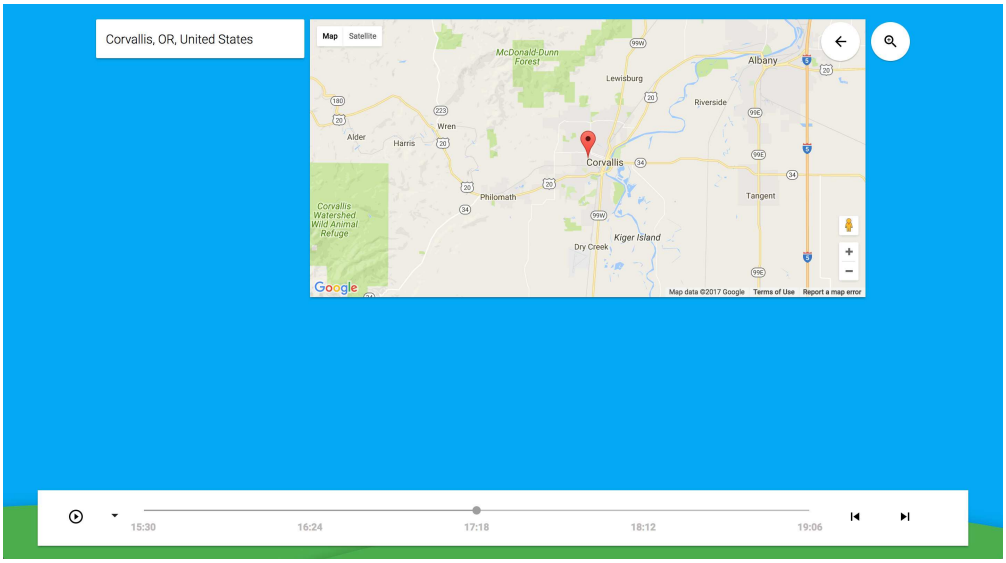


Fig. 3. Simulator with map expanded

7 Week by Week Summary of Group Activities

7.1 Winter Break Week 1

- Tuned basic simulator UI (non-functional).
- Began implementing simulator model using ephemeris JS to perform Sun/Moon position computations. Verified that these computations can be done quickly enough for our purposes.

- Built loading screen with simple toggle function. Therefore it requires only one line of code to toggle simulator loading state.

7.2 Winter Break Week 2

- Built controller to link simulator model and view.
- Made simulator UI ”prettier.” Added hills to scene.

7.3 Winter Break Week 3

- Built reverse geocoding proof of concept using Google Maps.

7.4 Winter Break Week 4

- Connected reverse geocoding code to simulator, enabling users to use a search bar to set the simulator to the location of their choice.

7.5 Week 1

- Talked to our sponsor about open-sourcing the 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 fall term. Got revised requirements document signed by our sponsor.
- 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. This code did not work for locations off the path of totality.
- Added an expanding and collapsing Google Map to the simulator that connects to the location search box.
- Restricted Google Maps search results to the United States.

7.6 Week 2

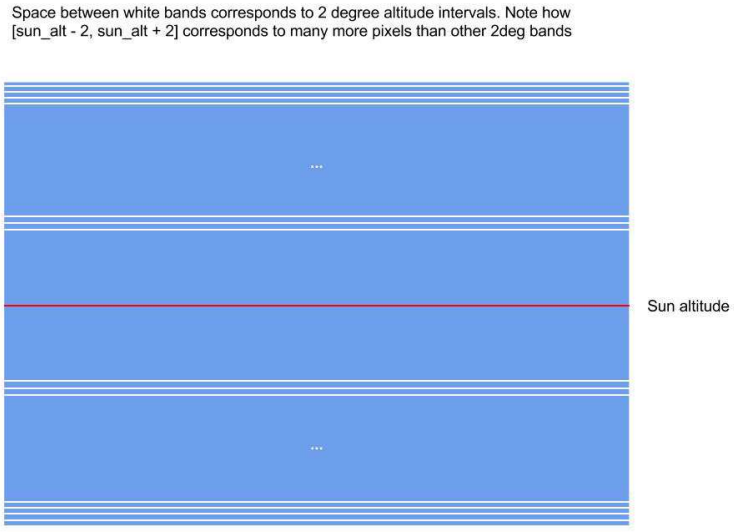
- Achieved very good (within 1 minute accuracy) results computing eclipse time using 2d interpolation with scipy. This accuracy extends across the United States, both inside and outside the path of totality, including in areas that were previously causing problems, like Florida.
- Told our sponsor that 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.
- Finished integrating Google Map to the simulator. Users can drop markers on the map to set their location or use the location search box. Marker drops and search results are restricted to the United States.

7.7 Week 3

- Altered view rendering of Sun/Moon y position in frame.
- Implemented simulator zoom mode.
- Met with our sponsor, the Google lead on the Eclipse Megamovie project, Justin Koh, and Gonglue Jiang, a Google UX designer regarding simulator design/ feedback.
- Updated zoom mode to center Sun in frame following feedback from Justin.
- 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.

7.8 Week 4

- Received Gonglue's design mocks from our sponsor.
- Proposed solutions to problems raised by the design mocks. See below:
  - The hills are quite tall, if left as-is, the Sun/Moon will not become visible until they are at a non-negligible altitude. To solve this, I proposed define 0 degrees of altitude at a point towards the top of the hills. This would mean that the bottom of the hills correspond to an altitude value of less than 0. This should not cause any problems.
  - 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 earlier than it is supposed to. To solve this, I proposed that we "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. See an illustration of this concept below:



- Added a play button to the simulator that runs through the eclipse from the time the the simulator is currently at until the end of the time range. The play function has adjustable speed.
- Replaced the epehemeris.js library with meuus.js to fix the accuracy issues with Sun and Moon position.

7.9 Week 5

- Finished the UI tweaks to the top and bottom control bars.
  - Centered the location search box and map/zoom buttons.
  - Changed the map dimensions so it expands to a large rectangle instead of smaller square.
  - All buttons are now using Material Design Icons instead of text.
  - Changed color scheme of the control bars to white/grey/black.
  - Lower control bar floats over the hills instead of sitting at the bottom of the screen.
- Asked our sponsor to talk with the Gonglue about getting us useable images of the background scene in his mocks of the simulator.

7.10 Week 6

- Completed midterm progress report
- Met with sponsor, discussed project timeline for the rest of the year.

8 RETROSPECTIVES

Positives	**Deltas**	**Actions**
lorem	lorem	lorem
lorem	lorem	lorem
lorem	lorem	lorem
lorem	lorem	lorem
lorem	lorem	lorem