Progress Report

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

December 4, 2016

CS 461 - Fall 2016

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 partnership 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 Ephemeris. 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.

2.1 Image Processor

This part of the project is still in the design and planning phase, development work has not begun. However, we have a clear design plan and have identified several tools and technologies that we will use to build the image processor.

The image processor will use OpenCV and its built in Hough Transform as well as an existing algorithm that we plan to modify and improve upon in order to identify eclipses in images. We have chosen to use C++ to build the image processor because it gives us better control over the speed at which this application runs while still providing us with the ability to leverage the OpenCV API. We have also clearly defined the inputs and outputs of the in order to allow the parts of the system that interact with the image processor to proceed independently with their development.

2.2 Image Processor Manager

Development has yet to start on the image processor manager, but we have a very good idea of how we are going to build it. Additionally, we will have access to code that Bret wrote during his internship this summer that we will be able to repurpose to handle large parts of the interactions with Google Cloud.

2.3 Eclipse Simulator

As it currently stands, progress has been made to create an initial working demo of the simulator. We have worked with the Ephemeris library to see how accurate its computations are. There was some variance when comparing the values it returned to other 2017 solar eclipse predictions, but we have decided that the variance is not large enough to warrant worry. Work has been done to create a front end view for the simulator. Eventually Google will be providing the sprites that the simulator will use, but we are currently just using simple SVG circles to simulate the Sun and Moon. We also have the code to render these Sun and Moon SVGs on the eclipse simulator window given their altitude and azimuth (vertical/horizontal angular coordinates). We will continue working on the eclipse simulator over winter break.

3 Problems and Possible Solutions

Around week 8 of the term, we found that some results differed in the Ephemeris library calculations and eclipse predictions we found elsewhere. We brought this issue up with our client, and he said to continue progress on the simulator as the variance we were seeing was not very significant for the simulator. Once further progress has been made on the simulator, it may be necessary to dig into the Ephemeris codebase to determine any problems.

4 Interesting Code

Below are some interesting sections of the Eclipse Simulator view code that we have so far. The code shown below focuses on rendering the Sun and Moon on the screen given their altitude and azimuth.

```
// EclipseSimulator namespace
var EclipseSimulator = {

   // [...]

    View: function()
    {
        this.window    = $('#container').get(0);
        this.controls  = $('#controls').get(0);
        this.sun       = $('#sun').get(0);
        this.moon      = $('#moon').get(0);

      // temp radius values
        this.sunpos   = {x: 50, y: 50, r: 2 * Math.PI / 180};
        this.moonpos  = {x: 25, y: 25, r: 2 * Math.PI / 180};

        // Field of view in radians
        this.fov = {x: 80 * Math.PI / 180, y: 80 * Math.PI / 180};

        // Center of frame in radians
        this.az_center = 0 * Math.PI / 180;
    },
    // [...]

    // Convert degrees to radians
    deg2rad: function(v)
    {
        return v * Math.PI / 180;
    },

    // Convert radians to degrees
    rad2deg: function(v)
    {
        return v * 180 / Math.PI;
    },

    // Convert a to be on domain [0, 2pi)
    normalize_rad: function(a)
    {
        var pi2 = Math.PI * 2;
        return a - (pi2 * Math.floor(a / pi2));
    },

    // Compute positive distance in radians between two angles
    rad_diff: function(a1, a2)
    {
        a1 = EclipseSimulator.normalize_rad(a1);
        a2 = EclipseSimulator.normalize_rad(a2);

        var diff = a1 > a2 ? (a1 - a2) : (a2 - a1);

        return diff > Math.PI ? (2 * Math.PI) - diff : diff;
    },

    // Determine if angle a is greater than angle b
    // That is, if b < a <= (b + pi)
    rad_gt: function(a, b)
    {
        a = EclipseSimulator.normalize_rad(a);
```

```
        b = EclipseSimulator.normalize_rad(b);

        a = EclipseSimulator.normalize_rad(a - b);
        b = 0;

        return a > b && a <= Math.PI;
    },

    // [...]
};

// [...]

EclipseSimulator.View.prototype.get_x_percent_from_az = function(az, radius)
{
    var dist_from_center = Math.sin(az - this.az_center);
    var half_fov_width   = Math.sin(this.fov.x / 2);

    if (this.az_out_of_view(az, radius))
    {
        // Just move the body way way off screen
        dist_from_center += this.fov.x * 10;
    }

    return 50 + (50 * dist_from_center / half_fov_width);
}

// This may need some re-visiting... the current computations imply a field of
// view of 2*fov.y
// Compute y coordinate in simulator window as a percentage of the window
// height
// Assumes altitude is <= (pi/2)
EclipseSimulator.View.prototype.get_y_percent_from_alt = function(alt)
{
    var height     = Math.sin(alt);
    var fov_height = Math.sin(this.fov.y);

    return 100 * height / fov_height;
```

```
}

EclipseSimulator.View.prototype.az_out_of_view = function(az, radius)
{
    var bound = this.az_center + (this.fov.x / 2);
    var dist  = EclipseSimulator.rad_diff(bound, az);
    // Body off screen to the right
    if (EclipseSimulator.rad_gt(az, bound) && dist > radius)
    {
        return true;
    }

    bound = this.az_center - (this.fov.x / 2);

    dist  = EclipseSimulator.rad_diff(bound, az);

    // Body off screen to the left
    if (EclipseSimulator.rad_gt(bound, az) && dist > radius)
    {
        return true;
    }

    return false;
}

// [...]
```

5 Week by Week Summary of Group Activities

5.1 Week 1

- Go to class

5.2 Week 2

- Worked on initial draft of problem statement.
- Established a LaTeX workflow.

5.3 Week 3

- Got feedback from sponsor and Kirstin on initial problem statement draft. Our sponsor was happy with our initial draft. Kirstin informed us that our problem statement was actually a bit too verbose and covered a lot of material that was more relevant to the technology review.

5.4 Week 4

- Had an initial meeting with Vee, our TA.
- Researched rotation matrices and quaternions at the suggestion of our sponsor. He pointed us to this material as being relevant to our rotating images, but likely more involved than actually necessary. He still encouraged us to research the material as he thought it was interesting and good to know. It was!

5.5 Week 5

- Created initial requirements document draft.

5.6 Week 6

- Reviewed requirements document with project sponsor.
- Followed up on new information we received from our client in week 5, specifically that we would have access to GPS/timestamp EXIF info in the image processor. We considered this and proposed a scheme to incorporate it into the image processor that our client was very happy with.

5.7 Week 7

- Our sponsor added an additional component to the project, the image processor manager. Previously this had been outside the scope of the project. We were initially hesitant about taking this on, but realized that there was some relevant code from Brets internship over the summer that we can repurpose for this component of the project.

5.8 Week 8

- Began work on the eclipse simulator. We ran into some issues computing accurate sun/moon positions using JavaScript. The first library we were using, SunCalc was not producing accurate results. After switching to the Ephemeris library, we started seeing better results.

5.9 Week 9

- Met as a group and ironed out a high level design for the various components of the project.
- Individually began working on our respective design document pieces.
- Continued to see accuracy issues with eclipse calculations in JavaScript, even after switching to Ephemeris. We spoke to our sponsor about this and he recommended moving on for the time being, as he did not think these computations were off enough to warrant concern, or even render them unusable.

5.10 Week 10

- Completed and turned in design document, we struggled a bit interpreting the IEEE 1016 standard for the design document.
- Began work on the view code for the eclipse simulator.

6 Retrospectives

Positives	**Deltas**	**Actions**
Image Processor design completed		
Image Processor Manager design completed		
Began work on Eclipse Simulator	User interaction	Implement buttons to alter sun/moon positions
Have access to Solar Eclipse Image Standardisation and Sequencing algorithm, research paper and code	Existing code is written in IDL and the algorithm does not completely meet our needs at this point in time	Port code to C++ and modify algorithm to meet our needs
Have reusable code from Brets internship that is relevant to both the Image Processor and Image Processor Manager	Need to work on re-purposing this code to fit our needs and incorporate into our project	Need to work with sponsor to get this code open sourced
Sponsor impressed by proposed usage of EXIF GPS data in Image Processor		