Progress Report

Totality AweSun

Bret Lorimore, Jacob Fenger, George Harder

February 8, 2017

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

2.1 Image Processor

2.2 Image Processor Manager

2.3 Eclipse Simulator

3 Problems and Possible Solutions

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 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.

5.2 Winter Break Week 2

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

5.3 Winter Break Week 3

- Built reverse geocoding proof of concept using Google Maps.

5.4 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.

5.5 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.

5.6 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.

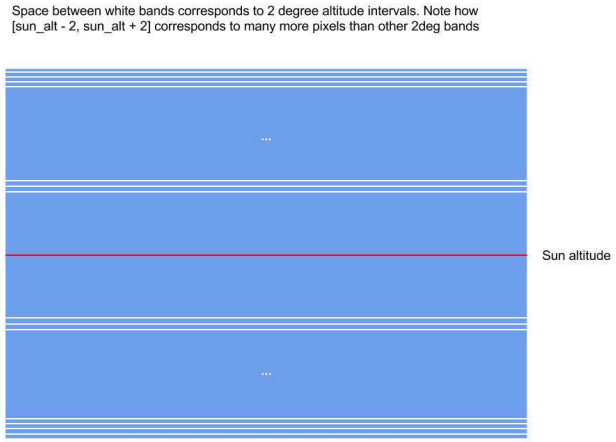
5.7 Week 3

- 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.

5.8 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:



5.9 Week 5

- lorem

5.10 Week 6

- lorem

6 Retrospectives

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