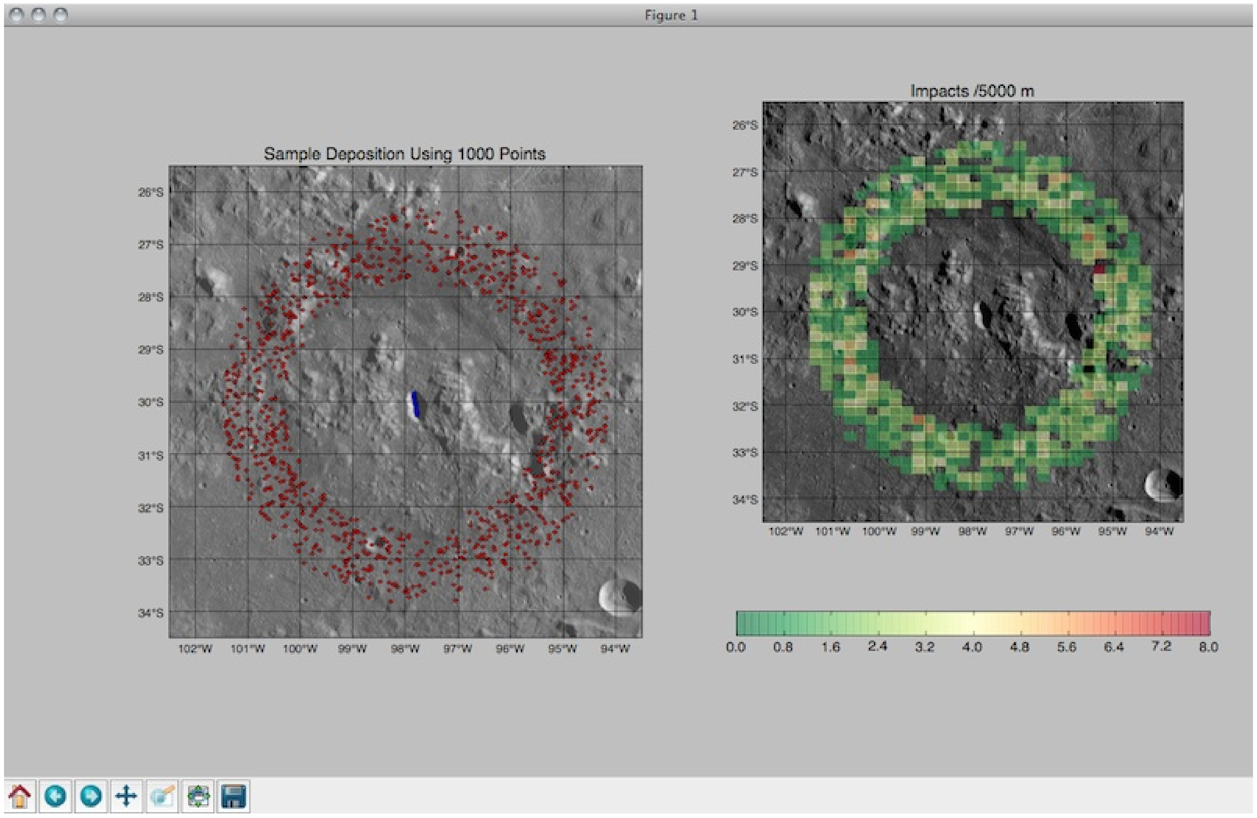
Strombolian Eruption Simulation Documentation

Release 0.1

Jay Laura

December 13, 2012

CONTENTS



This model seeks to provide a visualization of a strombolian ejection on the lunar surface. These are relatively simple models that do not account for the type of ejected material and need not account for atmosphere.

CHAPTER

ONE

INSTALLING THE STROMBOLIAN VOLCANO SIMULATOR

1.1 SVS Dependencies

SVS requires Python and depends on several other freely available Python modules. Prior to installing SVS, you should make sure its dependencies are met.

Table 1.1: SVS Dependencies

Dependency	Requirement
Python 2.7+	Required
NumPy	Required
GDAL	Required
GDAL Python Bindings	Required
Matplotlib	Required
Python Imaging Library	Required
Basemap	Required

Note that SVS is not tested with Python 3.x.

1.2 Installing

SVS is distributed as a stand alone script and therefore does not require installation. This documentation is distributed locally, with SVS. Simply place the script in a convenient directory.

Note that SVS does have a number of dependencies. These facilitate data processing, shapefile output generation, and topographic profile extraction. They are required and should therefore be installed prior to attempting to run SVS.

1.2.1 Installation on OS X

Python

Python ships with Mac OS X. It is not necessary to install a different version. Should you wish to, numerous online tutorials cover the installation of an additional python installation. As of 10.6 (possibly earlier) the default Python installation should be 64bit.

Numpy

Numerical Python is available for OS X via either pip or easy_install:

```
$ easy_install numpy
$ pip install numpy
```

NumPy, PIL, MatPlotLib

Alternatively, you can install NumPy, along with PIL and Matplotlib via binaries. These are compiled and made available by KyngChaos via his

OS X GIS Ports

Simply download the binares and install them. While you are there, you might grab SciPy, it will be useful sometime soon on some other project!

GDAL & GDAL Python Bindings

If you are using a package manager (Fink, MacPorts, Homebrew) install GDAL and the python bindings via that. Otherwise, KyngChaos provides precompiled binaries for installation.

GDAL Complete

Basemap

This is slightly more complex and an OS X DMG will be forth coming. The simplest methods, installation via automated source compilation is documented by the NASA Modelling Guru. In short, if you have macports, fink, or homebrew installed you can utilize one of the following, respectively. Otherwise, you are going to need to build via source...:

```
$ fink install matplotlib
$ port install py-matplotlib-basemap
$ brew install basemap
```

1.2.2 Installation on Windows

Windows installation

NumPy, PIL, Basemap, Matplotlib, GDAL Python Bindings

Christopher Gohlke has made a large number of binaries available for windows users. All dependencies save the core GDAL libraries can be installed via his site.

Windows Python Packages

Warning: Install the GDAL core package before installing the gdal python bindings.

GDAL Core

Installation of GDAL Core is slightly more complex. First, download the binary package from GIS Internals. This is gdal-##-####-core.msi. If you are an ArcGIS user, you likely want the MSVC 2008 version. Install this package as you normally would.

Two tutorials will be of assistance in getting GDAL setup in your PATH. Either tutorial covers the installation process.

1. My tutorial will exist as long as my Penn State account stays active
2. This USU tutorial also covers installation as a pdf.

1.2.3 Installation on Linux

The simplest installation for users who are likely comfortable with more complex installations!:

```
$ sudo apt-get install python-pip
$ sudo apt-get install gdal-bin python-gdal
$ pip install matplotlib
$ pip install numpy
$ pip install basemap
```

Note: If for some strange reason python is not already installed, you will be asked to install python running the first command, above.

CHAPTER

TWO

USING SVS

This page documents the implementation and usage of SVS. The model is constantly evolving and the documentation may not cover all pertinent features. See the doc strings in the code as these are the most up to date indication of functionality. We are current as of 0.1.

2.1 Usage

The basic usage of the script displays the iterative deposition of particles in a matplotlib gui window. After the deposition has completed it is possible to save the window to a PNG.

Two plots are displayed, the first is a movie that tracks where particles are deposited. The second, populated at the completion of deposition, is a density map, showing points per grid cell.

2.1.1 Running the iterative deposition

To get help, and see available parameters:

```
$ python simulation2.py --help
```

To run a basic model with all default ejection parameters:

```
$ python simulation2.py
```

Note: The default model emplaces 500 points with ejection velocity between 300 and 325m/s and ejection angle between 30 and 60 degrees. Azimuth of ejection is random.

To provide a fixed ejection angle:

```
$ python simulation2.py --angle 45
$ python simulation2.py -a 45
```

To provide your own range of ejection angles:

```
$ python simulation2.py --angle minAngle maxAngle
$ python simulation2.py --angle 25 75
```

To provide your fixed ejection velocity:

```
$ python simulation2.py --velocity 400
$ python simulation2.py -v 400
```

To provide a range of velocities:

```
$ python simulation2.py -v 350 425
$ python simulation2.py --velocity 275 450
```

2.1.2 Altering the grid size

By default, the model utilizes a 1000m^2 grid cell to track deposition density. The color bar indicating deposition scales with the data. To alter the grid cell size you can define Pixel Per Grid Cell as :math:: x * 100m, where x is some scalar multiplier. For example, when $x = 10$, grid size is $1000m^2$. When $x = 50$, grid size is $5000m^2$. This can be altered using:

```
$ python simulation2.py --ppg 50
```

2.1.3 Writing to a shapefile

Users can write the resulting model to a shapefile using the following:

```
$ python simulation2.py --shapefile SOMENAME.shp
```

Note: The resulting file is 'timestamped' with an iteration number. It is therefore possible, as of ArcGIS 10.0 to utilize the time slider to explore the temporality of deposition. We suggest setting the time specification to years when using this type of visualization.

2.1.4 Combining Arguments

Combining arguments is completely valid,for example the following would define a custom ejection angle, velocity, and output the results to a shapefile.:

```
$ python simulation2.py -a 35 80 -v 375 425 --shapefile MyTestRun.shp
```

2.1.5 Speed

Iterative deposition in a live matplotlib windows is suitably fast for low number of *n*. When *n* > 10,000 we suggest using the *–fast* flag. This flag bypasses the interactive visualization and writes directly to a shapefile. In this way, it is no longer possible to visualize the results in matplotlib. The *–fast* implementation is multicore enabled and will utilize all available processing capability on your machine.:

```
$ python simulation2.py --fast MyFastShapefile.shp
```

Warning: It is possible that python will warn you that your output is not a directory. This is because a shapefile with the same name already exists in the directory. Either change the name of your output or delete the old output.

2.2 Location

Currently this model is focused around the dark ring material eruption (Mare Orientale). The underlying basemap is hardcoded as is the output shapefile. With relatively trivial alteration to the code, it will be possible to extend the model.

2.3 Ejection Model

We are modeling this ejection using the most basic, atmosphere free, trajectory model. Briefly, here are the steps we take to calculate the position of the deposited material.

1. Randomly determine the ejection angle, θ
2. Calculate θ^2
3. Randomly determine velocity from the given range, v
4. Compute v^2
5. Compute total possible theoretical distance:

   distance = $(v^2 * \theta)/g$, where $g = 1.62m/s$
6. Randomly determine some ejection azimuth between 0 and 360.

Next we calculate the idealized height of the projectile over the body at 100m intervals. This interval was selected as it is the nominal resolution of the WAC DTM. We therefore know: total theoretical travel distance, azimuth, height above a planar surface at 100m interval. Using this information we then convert from geographic coordinates to pixel space and:

1. Extract the underlying topography along the line of travel. This is stored as an nx1 array of elevation data.

   Note: We are using a nearest neighbor interpolation. It is possible that the line of travel 'clips' the corner of a pixel. We are utilizing the spot elevation at that pixel in our calculation. This *should* be relatively safe due to Tobler's first law and the resolution of our data.
2. Normalize the trajectory elevation to the elevation of the ejection point. To do this we are subtracting the elevation at ejection from each index of the trajectory vector.
3. Add the absolute value of the elevation vector to the the elevation vector. Here the goal is to have all values $>= 0$. This allows for the vectorized computation of elements and is done to speed program computation.
4. Subtract the elevation vector from the trajectory vector to calculate the distance above our modified topography.
5. Add the negative offset to the elevation back to the newly generated 'distance above the topographic surface' raster.

   Note: This will need to be modified to test for sign should the ejection point be higher than 0m.
6. Return the index of impact point $*100$. This is the total distance in meters that the projectile travelled in pixel space and an estimation (within 99m) of total travel distance.