Design Document for
The USGS 3D Pipeline for Planetary Surfaces

Team
Space Blender

Team Members
Andrew Carter
Eric Ghazal
Jason Hedlund
Terence Luther

Due Date:
2/6/2014

[Document Purpose]
A high fidelity representation of the architectural design for the USGS Pipeline Project.

[Version 1.0]

Table of Contents

1. Introduction

The sponsor of our project is Trent Hare from the USGS in Flagstaff, AZ. Trent is the Cartographer/GIS Manager of the USGS Astrogeology Science Center that is responsible for mapping planetary surfaces using Digital Elevation Model (DEM) image data. Mapping planetary surfaces is an extremely important part of landing spacecraft on planets. Knowing the exact terrain that a spacecraft will be landing on and having accurate models of a planet's surface will help to ensure that the spacecraft is able to land safely and without damage.

The tools that are needed to process DEM images are available through the Geospatial Data Abstraction Library (GDAL) utility package. GDAL can create hill shades and color-relief-maps that can be overlaid on the DEM image to evaluate the planetary surface. The program Blender can be used to create 3D models and animations of the planetary surfaces from DEM images, and overlay the hill shade and color-relief-map images that GDAL produces. This process currently has to be done in a series of steps, in two different programs, making the processing of these images a manual task.

The software system we are developing will be a pipeline between GDAL and Blender that will streamline the processing of DEM images. This system will greatly simplify and automate many of the processes that are currently being done manually. The software system we are developing is going to be deployed as an add-on extension for Blender, a free and open-source 3D computer graphics product. Automation of these processes will lead to faster image processing, and possibly, off-loading processing tasks to a central server. 3D animations and flyover animations of images are currently features of Blender but these features are unintuitive and have many intermediary steps involved to achieve the desired result. The system will simplify the creation of 3D animations and image flyovers that Blender produces.

We will accomplish this process by integrating existing, proven, tools with the software we create into a pipeline style architecture. Using the tools available in the open source Geospatial Data Abstraction Library (GDAL), we will process DEM images to create hill shades and color relief maps. A script supplied by our sponsor allows us merge the hill shades and color relief maps to use as a texture for the image in Blender. We will make use of a third party Python script to import the image into Blender. Our job is to combine all of these individual processes with the processes we create into an automated extension accessible in the add-on menu of Blender.

The automation of these processes, in addition to the functionality we create, will simplify the processing involved with DEM images. The simplification of these processes will decrease the time needed to process images, and allow the images to be processed without having to learn how to use all the individual tools. This will help the USGS increase efficiency for its employees and allow a greater number of images to be processed in less time than it is currently taking.

Some issues that could arise during our development stage are defining non-standard data types and packaging everything as an add-on. DEM images may contain undefined areas (no-

data values) in the map because of the way the DEM is generated. These non-data areas must be identified and marked so the user does not get an incorrect surface. These issues will need to be addressed and tested thoroughly throughout implementation to make sure the product we create meets the requirements of our sponsor.

Diagram 1 shows a DEM in Blender that has been processed manually to create an image with an overlaying texture. This is the process that our team aims to automate with this project.

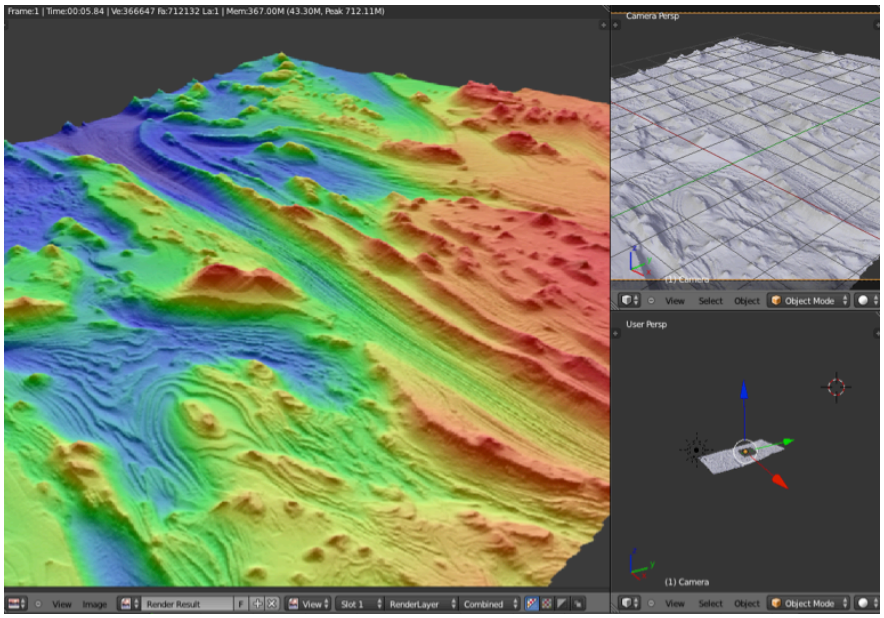

Diagram 1: A processed DEM using Blender

2. Architecture

This section is designed to provide an overall view of the architecture that will be implemented in our system. It will discuss any prominent influences that pointed to a particular architecture, along with a description of that architecture. Finally, it will discuss individual modules and their sub-components.

Description and Influence

Our architecture is influenced by the pipe-and-filter architectural style, but deployed as a plugin (add-on) within the Blender environment. Each of these filters are related to existing software packages that need to be executed within Blender's Python. We can extrapolate four major modules which group a collection of components (master module and sub modules) in the system. A visual view of the proceeding modules can be seen in the architectural illustration in diagram 2 on page 7.

- User Interface (UI) Module
    - UI Driver
    - File Selector

    - Flyover Selector
    - Color Selector
    - Scalar Selector
- GDAL Module
    - GDAL Driver
    - Hill Shade
    - Color Relief
    - Merger
- Blender Module
    - Blender Driver
    - DEM Importer
    - Apply Texture
    - Light Manager
    - Camera Manager
- Flyover Module
    - Flyover Driver
    - No Flyover
    - Selected Flyover
    - Algorithm Flyover

Each preceding module shifts the control flow of the program to the proceeding modules. Each module is driven by a core drive class which contains all of the module's components as functions. This design decision was made to decrease the necessary modules for the project, and keep the complexity of the code structure low.

Discussion

The **Plugin launcher** contains data related to the plugin, creators, open source licenses, and a checkbox to initiate the system. It then adds the script name to the drop down list of the import options in Blender. Once selected it initiates the UI bar on the left side of the screen with a simple menu. This begins the UI module and initiates the entire pipe and filter architecture.

The **UI Module** starts off with our UI driver, which contains all the data on drop down lists, check boxes, and file loading options, to give data to the GDAL Module. The file, color, flyover, and scalar components are simply UI components that are abstracted to show the function of the UI driver. Once these options have been selected and the user clicks "OK", the control will be shifted and data will be piped to the GDAL Module.

The **GDAL Module** has the master module the GDAL Driver, which contains the function definitions of hill shade, color relief, and merger. These filters belong to the GDAL libraries, specifically the gdaldem command for DEM processing. Hill shade creates a special coloring map depending on the height of the points in the image. Color relief creates a map based on user input from the color selector which contains several color schemes. This is then merged with an external library called Hsv_Merge.py, written by Trent_Hare. Hsv_Merge

combines these maps into a single map that can later be applied as a texture. Control then passes from the GDAL Driver to the Blender Module, piping the data of the merged color maps to the Blender Module.

The **Blender Module** initiates with the Blender Driver that runs an open source script from HiRise to import the DEM data into Blender. The texture that was passed in previously is then applied within this module and a light source and camera are created. This completes the action of the Blender Module, shifting control over to the Flyover Module.

The last module, **Flyover Module**, then takes over driven by the Flyover Driver, where the rest of the sub modules are ran as functions. The components do one of the following three: (1) create no flyover, (2) create a selected predetermined flyover, or (3) run the algorithmic flyover; this is performed through user selection. This will generate the nodes for a flyover path, concerning the camera focus of the newly imported DEM. The only data that needs to be forwarded is the UI data for the flyover module. That completes the objectives of the sponsor in the least costly way.

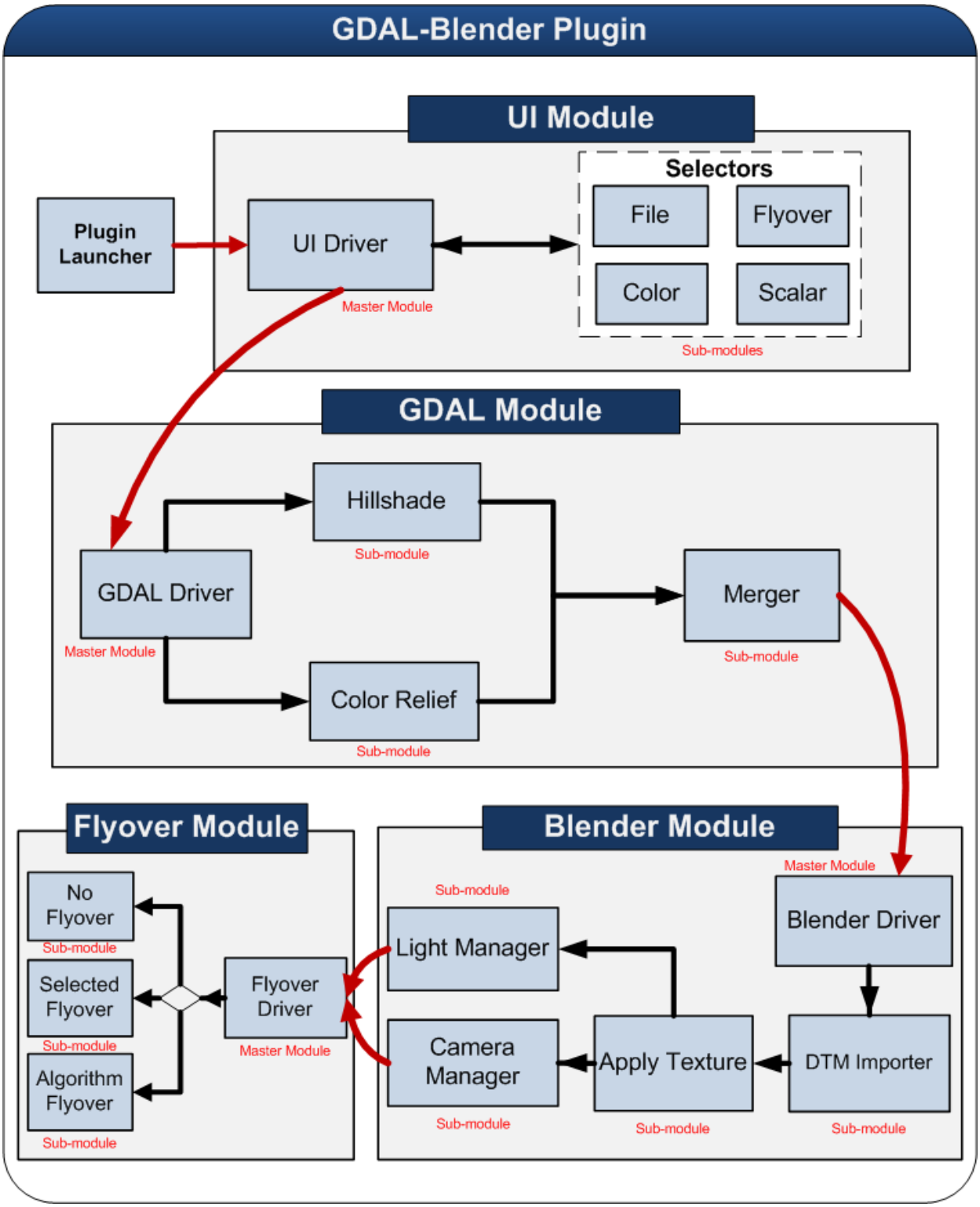

Diagram 2: USGS Pipe-and-filter Architectural Diagram

3. Module and Interface Descriptions

The module and interface description section provides detailed features of individual sub-modules. The sections below are sorted by the four main modules in the architectural pipe and filter order. Before the modules are presented, the first component that is mentioned is the system plugin. A UML class diagram containing the following modules can be seen on page 11.

System Plugin

Component Name: Plugin Launcher
Component Details: Core launcher
Responsibilities: Runs the entire pipe and filter network and passes processed data from the work flow.
Communication Mechanism: Pushes output data from other modules into their appropriate inputs.

Module: UI Module

Component Name: UI Driver
Component Details: Filter, Data Pipe: à Initiates UI in Blender
Responsibilities: Initiates UI in Blender for selecting image, color file, scaling options, and flyover details.
Communication Mechanism: Event driven by users, stores responses and from there requests data that will be fetched for other processes.

Module: GDAL Module

Component Name: GDAL Driver
Component Details: Filter, Data Pipe: à Initiates GDAL and Merge processes
Responsibilities: Initiates the process to create hill shade and color relief maps of a DEM image. Also initiates the merging of the produced hill shade and color relief maps..
Communication Mechanism: Calls GDAL functions and the hsv_merge script as sub processes in the shell. File input is received from UI Module.

Component Name: Hill shade
Component Details: Filter, Data Pipe: image à height shaded map
Responsibilities: Uses GDAL to process a DEM image and returns a hill shaded map of the image.
Communication Mechanism: Map is piped as input into the Merger.

Component Name: Color relief
Component Details: Filter, Data Pipe: image à colored map

Responsibilities: Uses GDAL to process a DEM image and returns a colored relief map of the image. The color scheme that will be applied depends on user input from the execution of this plugin.
Communication Mechanisms: Color relief map is piped as input into the Merger.

Component Name: Merger
Component Details: Filter, Data Pipe: image à merged hill shade and color-relief image
Responsibilities: Uses hsv_merge script to merge the hill shade and colored relief maps of the image. The resulting image will be applied as a texture to the DEM image in Blender.
Communication Mechanisms: Hill shade and color relief map are piped in as input to the merger. The hill shade and color relief have to be complete before the merger can execute.

Module: Blender Module

Component Name: Blender Driver
Component Details: Filter, Data Pipe: à Initiates DEM Importer.
Responsibilities: Initializes the DEM importer to bring our model into render, application of the texture component to be applied to our rendered image, and starts the light and camera manager.
Communication Mechanisms: Acts as a driver calling each component. The locations of the files are transferred to DEM importer and Apply Texture component.

Component Name: DEM Importer
Component Details: Filter, Data Pipe: → Brings in the DEM to render a landscape.
Responsibilities: Takes our DEM and sets up a mesh landscape structure for Blender to render.
Communication Mechanisms: The data for the mesh landscape is sent to Blender to render.

Component Name: Apply Texture
Component Details: Filter, Data Pipe: → Brings in the image from merge to apply a colored texture.
Responsibilities: Gives our landscape a colored texture to reflect elevation.
Communication Mechanisms: The data taken from the merge image is sent to Blender to apply our texture.

Component Name: Light manager
Component Details: Filter, Data Pipe: → Creates a light source in Blender.
Responsibilities: Creates a “sun” light source that allows the landscape of the DEM to be visible after rendering.
Communication Mechanisms: Data is sent to Blender as to the position of light sources and values.

Component Name: Camera Manager

Component Details: Filter, Data Pipe: → Sets the position of camera for image rendering.
Responsibilities: Sets a default position of the camera so that the DEM surface is within view upon image rendering.
Communication Mechanisms: Sends data to Blender to place a camera to view rendered landscape.

Module: Flyover Module

Component Name: Flyover Driver
Component Details: Filter, Data Pipe: → Initiates No flyover, Selected Flyover, and Flyover Algorithm components.
Responsibilities: This driver initializes the No Flyover, Selected Flyover, and the Flyover Algorithm components.
Communication Mechanisms: Acts as a driver calling each component. Location of the files needed are transferred to all three other components; No flyover, Selected Flyover, and Flyover Algorithm.

Component Name: No Flyover
Component Details: Filter, Data Pipe: → Does not produce a flyover motion
Responsibilities: Gives the user a simple view with no fly over, just a static view of the landscape.
Communication Mechanisms: No flyover is created, control is reverted back to user, and they can do what they want with the imported image in Blender.

Component Name: Selected Flyover
Component Details: Filter, Data Pipe: → Creates a flyover path selected by the user.
Responsibilities: Selects the path of the flyover whether it is straight, zig-zag, or a circle.
Communication Mechanisms: Takes in DEM data for determining the flyover path, outputs the flyover path on Blender's view.

Component Name: Flyover Algorithm
Component Details: Filter, Data Pipe: → A specific algorithm is used to determine a more desirable, custom path.
Responsibilities: Looks for hot spots (extreme changes in elevation) and creates a fly path that will take a look at all of these areas so that they can be seen by the camera. In addition, it will guard against the camera diverting off of the map.
Communication Mechanisms: The algorithm takes in DEM data to determine hotspots and flyover path, and outputs the flyover path to Blender's view.

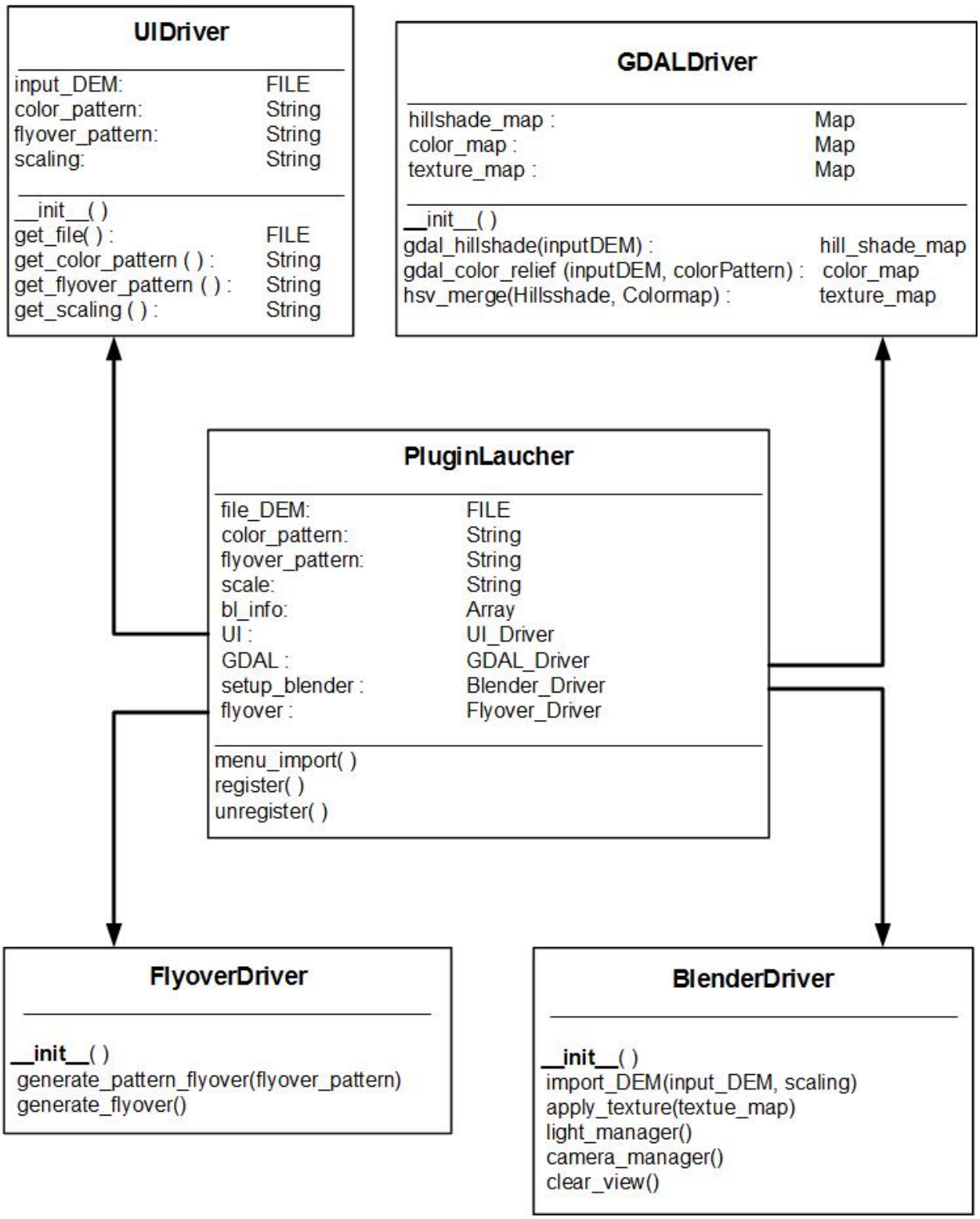

Diagram 3: USGS UML Script Diagram V1.0

4. Implementation Plan

This implementation section is designed to give an in depth look into how our team will implement the entirety of the project, through specific methods such as Scrum, a Gantt chart, and milestones.

Scrum

To begin, the team will be using Scrum as our specific agile software development framework as we feel this provides the best opportunity to efficiently and successfully complete this project given the time frame. There are two critical roles within Scrum, the Scrum Master who will be Eric Ghazal and the Product Owner who will be Jason Hedlund. To help manage this process, the team will be using the ScrumDo software tool which facilitates the product backlog, sprint backlog, sprints, burndown charts, and incremental delivery.

Gantt Chart

We have created a Gantt Chart, shown below in diagram 4, to illustrate a projected timeline of our implementation stages, documents, and milestones. Implementation can be seen in red, testing phases in green, design documents in blue, and milestones in purple diamonds. The milestones are a key part of our team's progression and if any milestones are missed that will, in return, cause a damaging ripple effect towards later milestones.

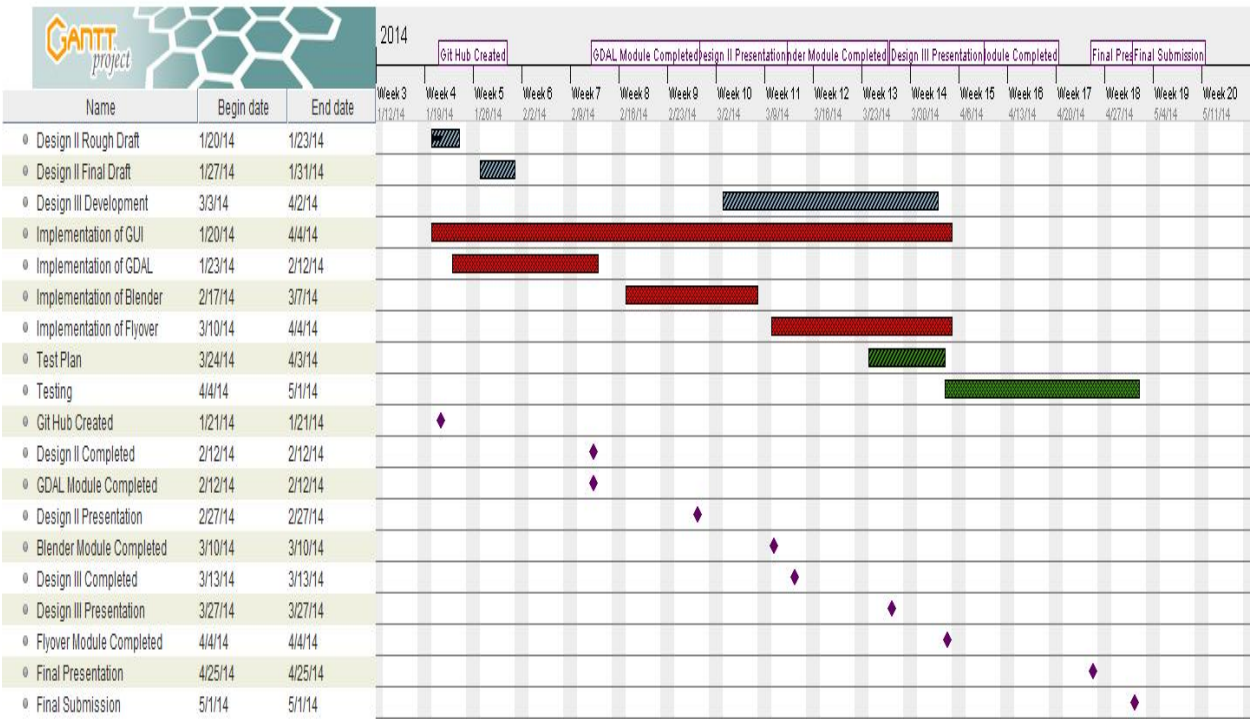

Diagram 4: A Gantt Chart representing the Spring semester progression plan.

Meetings

Every week our team will meet on Tuesdays and Thursdays at 12:30 pm to develop design documents, code modules, and unit tests. These meetings will review sprint progress, and tasks for the following sprints will be identified; individual tasks will be assigned to conform to our sprint backlog. Stumbling blocks will be identified and documented by the scrum master and will be resolved outside of the meetings.

A weekly meeting will take place with our mentor, Dr. Palmer, Thursdays at 2:20pm.