A Final Report for
The USGS 3D Pipeline for Planetary Surfaces

Team
Space Blender

Team Members
Andrew Carter
Eric Ghazal
Jason Hedlund
Terence Luther

Due Date:
5/1/2014

[Document Purpose]
A final document that covers all aspects of our project.

[Version 1.0]

Table of Contents

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.

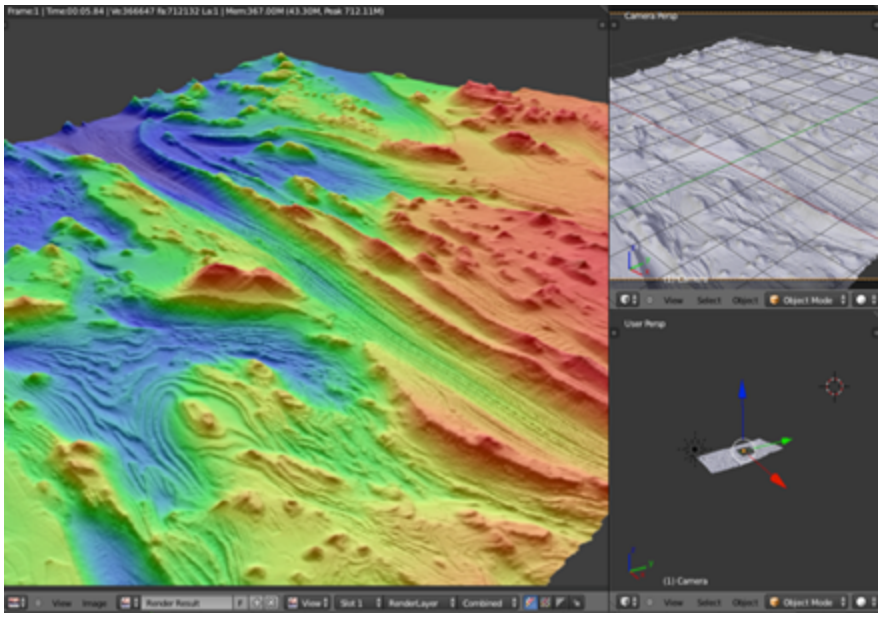


Diagram 1: A processed DEM using Blender

Process Overview

For the development of our Blender plug-in we used a variety of tools to help keep our project organized. We used the Scrum development process along with an online management tool called Scrum-Do to organize and prioritize tasks. We used a Git repository along with an application, SourceTree, to manage version control of our software. Our project involved integration of GDAL tools with the 3D modeling application Blender.

Eric Ghazal led the team and was in charge of task management and weekly meetings. Andrew Carter was in charge of the applications architecture and making sure that implementation was in line with the design of the architecture. Terence Luther took the role of Lead Programmer making sure that the implementation of the Flyover, and other modules met coding standards and performed as intended. Jason Hedlund was responsible for communication between the team and the project sponsor Trent Hare, as well as facilitating communication with the project mentor Dr. James Palmer. Jason was also in charge of the layout and updating of content on the team's website. Email communication and weekly meetings were a vital part keeping everyone working on the project on the same page.

Requirements

1. Functional Requirements

1.1. System Requirements (User Application)

1.1.1. The system shall provide a simplified method for processing DEM images.

1.1.2. The system shall be able to process a single DEM image at a time.

1.1.3. The system shall have an automated method to batch pipeline to process DEM images in GDAL and pipe them to Blender.

1.1.4. The system shall provide a simple method to help define an optimized fly-over track for the 3D model for rendering an animation.

1.1.5. Simplified installation kit that installs all necessary software (GDAL, Blender, etc…)

1.1.6. All coding should be done within Python with limited shell execution (if possible).

1.2. GDAL/OGR Script

1.2.1. GDAL shall be able to process .img, .tif, or .dem formatted images (DEMs)

1.2.2. GDAL shall use existing GDAL binaries to create hill shades.

1.2.2.1. Shaded areas (black regions from hill shade) should have specific handling for blender. Should be zeroed out in shading or marked in some special way with Blender.

1.2.3. GDAL shall be able to use existing GDAL binaries to create color-relief maps

1.2.3.1. Color choices shall mostly be colorblind friendly.

1.2.3.2. Color scales shall allow for user choice from a set of color patterns that will be provided by sponsor.

1.2.3.3. Color patterns should include, but are not limited to:

1.2.3.3.1. Grayscale (8-16 bit grays, Lunar)

1.2.3.3.2. Blues & Whites (default)

1.2.3.3.3. Brown & Blue (Earthlike)

1.2.3.3.4. Brown & Red (Mars)

1.2.3.3.5. Rainbow (not colorblind friendly)

1.2.4. GDAL shall be able to scale images to a size that is usable in Blender.

1.2.4.1. If the size exceeds 10,000 or -10,000 then rescale the image by a factor of 10 or an aesthetically appropriate value.

1.2.5. GDAL shall be able to integrate the GDAL raster library to load Digital Elevation Models (DEMs) into Blender.

1.3. Blender

1.3.1. Blender shall be able to render the image processed through GDAL/OGR script.

1.3.2. Blender shall be able apply the hill shade to the DEM image.

1.3.3. Blender shall be able to apply the color relief map to the DEM image.

1.3.4. Blender shall not render 'non-data' areas

1.3.5. Blender shall be able to render a 3D flyover animation of the DEM image.

1.3.5.1. Will have preset flyover animation that exists in a straight line.

1.3.5.2. Height of flyover will be preset to 100ft-1000ft over the highest rendered object in a simple linear direction by default at an attractive skewed angle.

1.3.5.3. (Optional) Pretty paths algorithm, which will create a flyover path that focuses on hotspots or areas input by user or hotspot algorithm.

1.3.6. Blender shall be able to export an .mp4 video file of the 3d flyover animation.

1.3.7. Blender shall be able to export an .x3d 3D model for WebGL-capable browsers

2. Environmental Requirements

2.1. The system shall be cross platform compatible with OSX, Windows, Linux.

2.2. The system shall require Python 3.2 installation on the host computer.

2.3. The system shall require Blender 2.68a installation on host computer.

2.4. The system shall require GDAL package to be installed on the host computer.

3. Non-Functional Requirements

3.1. Compliance

3.1.1. Open source standards.

3.2. Usability

3.2.1. Color schemes should be supported for appeal and colorblind.

3.2.2. Should have either a GUI, Command-line, or Blender Extension interface.

3.2.3. Should be usable by scientific community (USGS).

3.3. Performance

3.3.1. Processing shall not take longer than 24 hours to perform.

3.3.2. Efficiency should not be a concern as it is gated by GDAL and Blender.

3.3.3. Should print out status of process execution to the chosen UI.

3.4. Extensibility
- 3.4.1. Support additional filters being added to the pipeline.
- 3.4.2. Must be well-commented, so future coders can add additional functionality in seamlessly.

3.5. Scalability
- 3.5.1. Use single threaded multi-tasking instead of threading.
- 3.5.2. Support future cluster (for Linux due to USGS)

3.6. Portability
- 3.6.1. Operating System Portability
  - 3.6.1.1. Cross platform support (Linux, OSX, Windows)
  - 3.6.1.2. Python powered for additional portability of code.
  - 3.6.1.3. At least Linux support should be provided.
- 3.6.2. Data Interface
  - 3.6.2.1. Input data must conform to what GDAL supports for DEMs
  - 3.6.2.2. Input data scaling must conform to Blender.
  - 3.6.2.3. Data must be able to be piped to other filters in system.

3.7. Documentation
- 3.7.1. All changes to requirements should be documented. This will not have a traceability matrix due to time cost.
- 3.7.2. All documentation will conform to standards laid down in the team foundations document.
- 3.7.3. Meetings with the sponsor will be stored in a drop box folder labeled meeting_minutes_[date].
- 3.7.4. All documents should be stored in the .doc or .docx format.

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 easier to manage.

Discussion

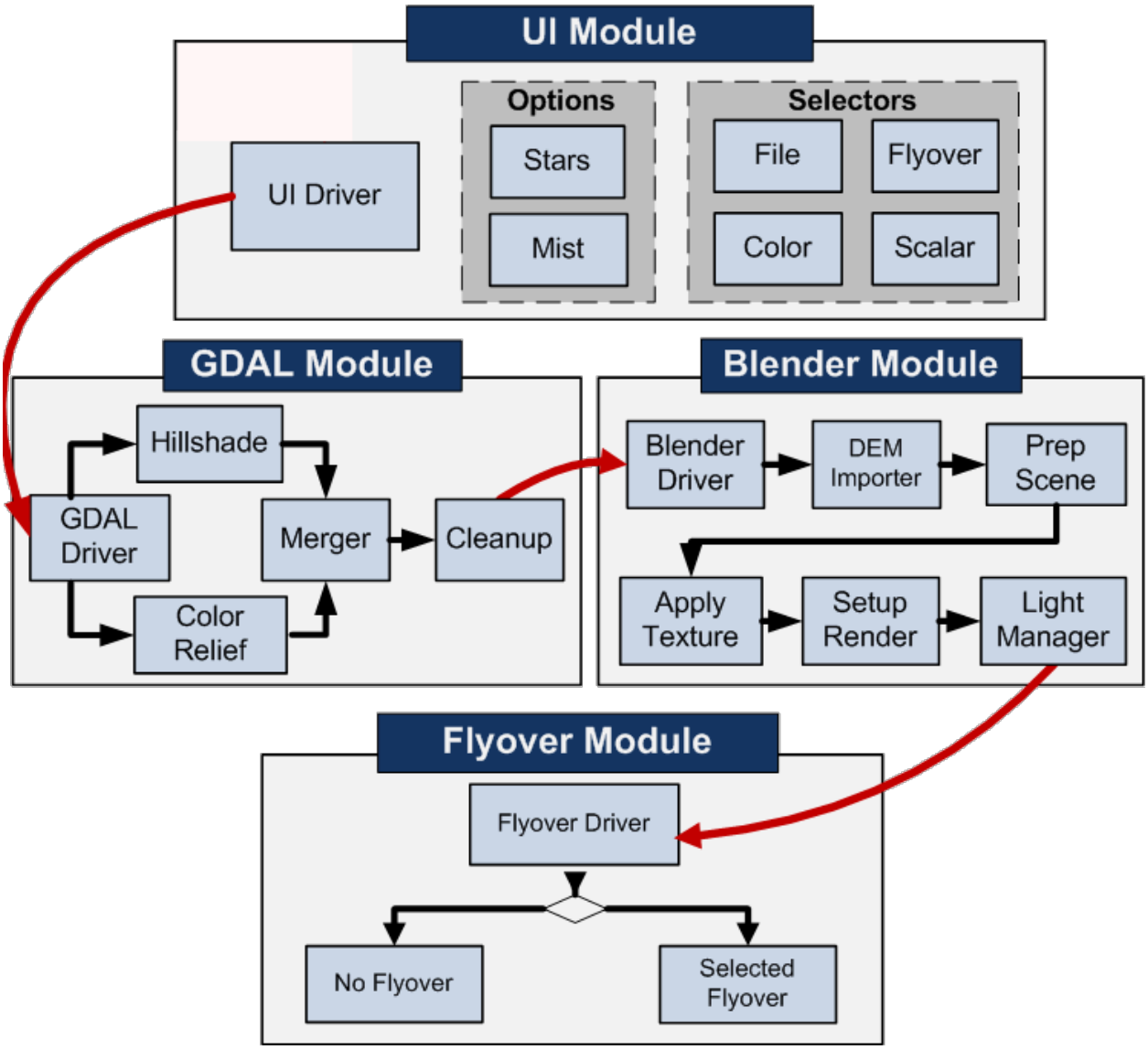
The **Plugin launcher**, which is no longer contained in the architecture, initiates the whole process. It contains data related to the plugin, creators, open source licenses, and a checkbox to initiate the system. It adds the script name to import options in Blender. Once selected it initiates the UI 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** initiates the UI driver, which contains drop down lists, check boxes, and the file selector. The file selector, color pattern, flyover pattern, and scalar/resolution components are abstracted in the UI module. The stars and mist components represent checkboxes in the UI, which allows the user to apply them for the rendered image. Once these options have been selected and the user clicks “Import and Space Blend DEM”, the input data is piped to the GDAL Module.

The **GDAL Module** runs the master module, the GDAL Driver, which contains the hill shade, color relief, merger, and cleanup functions. The hill shade and color relief components represent the GDAL libraries, executing the gdaldem commands in the console. Windows, and OSX/Linux have different ways of executing the sub processes that run GDAL, because windows uses a package called OSGEO4W. The hill shade creates a color map based upon height values. The color relief creates a color map based on the selected color file. This is then merged with an external library called hsv_merge.py. Hsv_merge combines the color and hill shade maps into a single color texture. The cleanup method then runs, which purges the color and hill shade maps. The texture is then piped to the Blender Module and later applied.

The **Blender Module** initiates with the blender driver that is based on an open source script from HiRise plugin. The first component that is executed is prep scene, which deletes the cube in the scene. The DEM can be loaded using the HiRise importer logic, which will load the DEM into the user context as a mesh. The previous texture is then applied on the imported mesh. The renderer is set up and finally a light source is created at the top of the mesh to see the world. This completes all action in the blender module. Except for use selection data, no data is piped to the flyover module. The flyover module design was changed at the last minute to support an entirely modular design.

The **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: (1) create no flyover, or (2) create a selected flyover. If the user selected no flyover, a default camera is setup for the rendering. If the user selects a flyover pattern, the appropriate geometry will be constructed for a video render. The geometries that were in the final flyover were: circle, diamond, and linear patterns. This will generate the geometries for a flyover path and attach a camera to them. That completes the descriptive narrative of our architecture.



Our prescriptive architecture of a pipe-and-filter style was easy to maintain and follow. The amount of any architectural drift was minimal as our descriptive architecture was essentially the same.

Testing

This is a test plan for the USGS 3D Pipeline for Planetary Surfaces. It is the final written document entailing the high level testing processes and objectives for the USGS project. As a high level testing outline, this document does not necessarily cover the specific test cases used, instead it focuses on the general process and objectives of the testing effort. The project is Blender and Python powered, and the testing effort is modularly designed using Python's built in unit test tightly organized as a collection of Python modules. Since this is a Blender project, most

of functionality is graphical in nature, and will be confirmed by the user. However, the functions and interactions can be thoroughly tested with Python's built in unit test in isolated modules. The document includes an expansive description of the three different testing activities as they apply to our four main modules.

The four main modules correspond to our pipe and filter architecture, and these modules are the UI, GDAL, Blender, and Flyover modules. The testing activities include unit testing, integration testing, and usability testing. Most of our effort will be focused on the GDAL and Flyover modules, since these are unique to our system and can be properly unit tested. The UI module will be thoroughly tested in the usability section. GDAL, Blender, and the Flyover modules will be tested in the unit testing section. The integration testing section will focus on how these components interact with each other and within the respective sub-modules of our primary modules to ensure data flows as expected. It is important to remember that this project belongs to a graphical system, so while unit and integration testing are important, the majority of our testing efforts are concerned with user testing and user acceptance of the product. Trent Hare is allowing us to run our usability tests at the local USGS facility, this ensures that the sponsor is getting the product he expected and wanted.

Unit Testing

Unit testing for this project will focus on the modules that call external functions as well as having return values. These modules are our GDAL, Blender, and Flyover modules. We will test for the desired return values from functions in all of these modules. The focus will be directed on functions that we have created, not existing ones taken from related projects, such as hsv_merge and DEM mesh importer logic.

Unit Testing Frameworks

We will be using two frameworks for the unit testing in our project:

1. Built in Python Unit-test Framework
2. Python Built in Unit-test Framework

Testing Procedures

All unit testing procedures will be defined in their own Python module to keep the testing suite for the project modular. By keeping all the testing code separately from the rest of the project we will be able to individually test functions independently of dependencies that other functions may have on them. This will also allow us to run the entire testing module by using one command in the command line, making it useful for generating reports as we will only have to be concerned with the output from one set of tests.

Definitions for Testing Input

1. Good input
    a. Proper format DEM image (.IMG)

    b. Correct file path to image.
2. Bad input
    a. Improper format image - any format other than .IMG
    b. Incorrect file path to image.

Items to be Tested

The GDAL module has four main functions that shall be tested:

1. gdal_hillshade
    a. This function will be tested with incorrect input to make sure that it returns a system exit command with a return code of 1.
    b. This function shall be tested with proper input to make sure that external command succeeds and a return code of 0 is returned upon success.
2. gdal_color_relief
    a. This function will be tested with incorrect result to make sure that it returns a system exit command with a return code of 1.
    b. This function shall be tested with proper input to make sure that external command succeeds and a return code of 0 is returned upon success.
3. hsv_merge
    a. This function will be tested with incorrect result to make sure that it returns a system exit command with a return code of 1.
    b. This function shall be tested with proper input to make sure that external command succeeds and a return code of 0 is returned upon success.
4. gdal_cleanup
    a. This function shall be tested with proper input to make sure that external command succeeds and a return code of 0 is returned upon success.
    b. This function shall be tested with proper input to make sure that external command succeeds and a return code of 0 is returned upon success.

The Blender Module has several functions that will be tested including:

1. create_stars
    a. This function will be tested to ensure the properties in the context are in fact updating as expected when rendered. We will iterate through the context to make sure it is updating correctly.
2. create_mist
    a. This function will be tested to ensure the mist properties are being set correctly in the context, so that it will act as expected when rendered. We will iterate through the context to make sure it is updating correctly, but only when toggled.
3. load
    a. The load function will be tested to make sure it is returning true under appropriate conditions. No other drop down lists will be selected for these tests; just the default mesh loading module should be run.

b. Test to see it is returning false if the image failed to load correctly. Ideally this should not happen, but we need to know if it fails when expected.

The Flyover Module has several functions that will be tested:

1. no_flyover
    a. This function will be tested to ensure that if no flyover is the selected option that a camera for rendering an image based off the center position of the image is created.
2. get_linear_path
    a. This function will be tested to ensure that when a linear path is the selected flyover that the list of points returned from the function corresponds to the actual points we desire the function to return.
3. get_circle_pattern
    a. This function will be tested to ensure that when a circle path is the selected flyover that the list of points returned from the function corresponds to the actual points we desire the function to return.
4. get_diamond pattern
    a. This function will be tested to ensure that when a diamond path is the selected flyover that the list of points returned from the function corresponds to the actual points we desire the function to return.
5. check height
    a. This function will be tested by testing predetermined height values against the list of height values the function actually returns.

Code Coverage

1. Code coverage shall be tested during unit testing using the PyCharm IDE's built in code coverage testing tool.

Items not to be Tested

This project integrates existing toolsets whose failures are beyond our control. These toolsets include GDAL, HSV_MERGE, and Blender.

1. GDAL
    a. If our calls to GDAL are made successfully with good input we cannot guarantee that the results GDAL produces will be the desired result.
    b. We will display the error codes that GDAL returns in the console window.
2. HSV_MERGE
    a. We will not be testing the results of HSV_MERGE other than making sure that calls to the program are correct with good input.
    b. We have no method of verifying if the results of HSV_MERGE are actually correct.

3. Blender
    a. We will not be testing Blender module beyond the scope of our new code. Faults or system crashes caused by the original DTM mesh loading process are assumed to be thoroughly tested and stable. Faults in Blender are also beyond our control.

Unit Testing Pass/Fail Criteria

1. All unit test cases must complete for over 90% of all test cases.
2. All test cases with good input in the GDAL module shall return a return code of 0.
3. All test cases with bad input in the GDAL module shall raise a system exit command and return code of 1.
4. All test cases in the Blender module shall return code of 0 if good or 1 if false.
5. Values of points returned in Flyover function tests shall be with plus/minus 5% of pre-calculated desired results.
6. Code coverage shall be at least 90% for all code covered during testing.

Integration Testing

Our integration testing will follow the same framework as our unit testing section, but focusing on testing each module and ensuring they are working correctly with one another. Because our project has a pipe and filter architecture, we will be able to drop in each module and test the output for accuracy. We will be taking the approach of a bottom up integration test; focusing on the smaller modules, functionality, and working our way up to the major modules. While we are verifying the lower modules, we will be using our unit testing information from our last section to verify the modules.

Example Outline of Integration Test

We will give an outline of our overall procedure of integration testing and our assumptions in the next part of this section. For our project, we divided our modules into two types of tests to be completed in order to satisfy our bottom up approach. In the following test, sub-module, module, and integration, we will outline the overall process of how we are going to test said components.

Architecture Division

- Major modules are the largest abstraction of our architecture.
    - Examples: UI, GDAL, Blender, Flyover module.
- Sub-modules are our smallest abstraction of our architecture.
    - GDAL: hill shade, color-relief, merge.

Sub-module Test

1. Identify sub-module from architecture.
2. Identify the methods that correspond to the sub-module.

    a. Isolate sub-module for unit like testing.
    b. Document the correctness of sub-module.
        i. Support with unit testing from our last section.
3. Test sub-module for correctness.

Major Module Connections

1. Identify major modules from the architecture.
2. Identify the sub-modules and their connections to form the major module.
    a. Test the sub-module connections for correctness.
3. Test major module for correctness.

Integration of Major Modules Tests

1. Start with the first module and filter in the pipe and filter architecture.
2. Test correctness with subsequent modules in the architecture.
    a. Add in modules one by one till all modules are done.
        i. Unit like test should be done each time a module is added.
3. Document overall correctness based on previous tasks and previous subsections of testing.

Testing Specifications of Major Modules

- **UI**
    - Document input and output of each UI input.
        - Example: Selectors; File, Color, Scaler, Flyover.
    - Document where the output of the UI is sent and check for correctness.
- **GDAL**
    - Document commands that call GDAL, ensuring that GDAL is called.
        - The GDAL library does not need to be tested.
            - Out of scope for our project.
        - HSV_Merge does not need to be tested.
            - Correctness has been tested by third party.
- **Blender**
    - Document all functions that call Blender utilities to ensure correctness.
        - DEM importer does not need to be tested.
            - Correctness has been tested by third party.
- **Flyover**
    - Document each flyover sub-module for correctness.

Measurements of Success

- **UI**

    - All selectors have the correct input check to ensure system will work with user input.
    - All selectors call/pass data to the correct modules.
- **GDAL**
    - GDAL functions and their outputs are correctly mapped to locations specified by our design.
- **Blender**
    - Blender functions are mapped correctly for input and output.
- **Flyover**
    - Flyover sub-modules are mapped to the selector in the UI.
    - Flyover sub-modules work on blender scene (blender environment).

Outcome
We will have increased the confidence in our systems functionality. Coupled with data generated by our unit testing, the overall correctness of our system should be ideal for use by the USGS.

Usability Testing
Our group will create scenarios that have specific tasks for the participant to complete and at the end of each scenario an evaluation will be completed along with questions directed towards the participant.

Participants
The selected participants are a diverse group with different levels of computer knowledge. In this group, there are a few students from the NAU Computer Science program and a few employees from the USGS. The small group from the USGS will comprise of someone that is extremely familiar with Blender and GDAL, someone that has a good, general sense of computers, and lastly someone that does not have strong working knowledge of computers. Besides the previously mentioned criteria, there are no other background skills that a participant must obtain. The participants will be asked to complete a set of tasks representing a scenario and to provide feedback and the usability of the interface.

Procedure
The usability tests will take place at the USGS campus and the NAU Engineering building. A minimum of two team members will be observing each test. At the beginning of each scenario, one observer will read out the specific tasks needed to complete what is being asked and a printed copy of the tasks will also be available to the participant. After the scenario is over, a short exit questionnaire will be provided to the participant to answer questions pertaining to the experiment.

The Test

The test will be developed by the team members and will contain a few different scenarios aiming towards a different output. The extent of our tests will vary, however, each test will attempt to complete three objectives. These objectives are listed below.

Purpose 1: Can the user successfully navigate the interface from start to finish.
Purpose 2: Is the result of a scenario the desired result.
Purpose 3: Was the scenario completed within a given amount of time.

The extent of our different tests will vary depending on the user's computer knowledge and background. The most basic participant will only test against the interface and its output. Meaning that before the participant sits down, Blender will already be up and running and the user interface will be installed. As the participants' knowledge and level of comfort increase with using computers, the test becomes more abstract. These levels of abstraction may include:

1. Blender will be running but the participant will have to find and install the interface.
2. The participant will have to open Blender, find and install interface.
3. Run Blender through the Command Line.
4. Run the entire process through command line arguments.

Measurement

Measuring the different tests will be a timed measurement against the participant and their specific scenario. The completed time of each participant will be compared to a skewed time of one of the team member's. Because the team can navigate the interface with ease, a small amount of time will be added to the end of their finished time. The success of completion will also be measured along with a stress level measurement evaluating the stress of the participant through a questionnaire.

Evaluation and Questions

The evaluation and questions will be directed towards the participant at the end of each test. The evaluation will measure how much success each participant had during their test; this measurement will be timed. There will be a few short questions directed towards the participant's thoughts and feelings about the test. These may include stress level, how comfortable were you during the test, what should be different about the test, and other similar questions.

Reporting
The results of each participant will be carefully reported and studied. The results will help the team adjust their interface and any other features to better the future users and the plugin.

Implementation of Test Plan
Our user testing was performed at the USGS with the help of our project Sponsor Trent Hare. Trent arranged for us to test our product with four USGS employees. We prepared two different tasks for our users to perform:

1. Render a flyover video from a DEM
2. Render a 3D image from a DEM

The users we tested were novice and expert users. All users tested completed the tasks assigned with minor to no errors. The users followed the user guide posted on our website to assist in doing the tasks they were given. Three of our four users completed the tasks given in less than three minutes. We were satisfied with these results as the original process took roughly 30 minutes to complete. See appendix for usability testing artifacts.

Our code was also tested extensively through unit testing. The unit tests performed conforms to the tests described in our testing plan. See appendix for unit test artifacts.

Integration testing was performed as per the process described in the testing plan.

The testing results that we received gave us more confidence in our software and its overall stability, usefulness, and end goals. The code that we are delivering was tested extensively to ensure that the final product worked as intended.

Deployment

The USGS is going to include our plug-in as part of their image processing software package.

We are going to submit our plug-in as an official plug-in to be included in the Blender application

Our plug-in is Open Source and available for other developers to modify or add additional features.

Glossary

Blender - Open Source 3D modeling Application

DEM - Digital Elevation Model

GDAL - Global Data Abstraction Library

UI - User-Interface

USGS - United States Geological Survey

Appendix

Links

Team SpaceBlender Website

Plug-in User Guide

UGRADS Poster

Attachments

Testing Artifacts