Test Plan Document for
The USGS 3D Pipeline for Planetary Surfaces

Team

Space Blender

Team Members

Andrew Carter

Eric Ghazal

Jason Hedlund

Terence Luther

Due Date:

4/10/2014

[Document Purpose]

Test plan to ensure proper implementation of our team's functional and non-functional requirements.

[Version 1.0]

Table of Contents

Introduction

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.