Test Plan

Gigapixel Image Rendering

Daniel Garcia-Briseno
Matthew Ostovarpour
Douglas Peterson
Thomas O’Brien

Version: 2.7

Table of Contents:

Introduction

This document provides information on how we plan on testing both our algorithm and Image viewer. Due to the nature of these programs, it is difficult to write unit tests, for example, our algorithm needs to run fast, and it needs to produce a viewable result. We can write code to make sure it runs faster than a set amount of time, but we cannot write a test to determine if the output will show well on the screen

Similarly for the Viewer, there is not much that we can do in terms of unit testing, The Image viewer simply takes the output from the algorithm and renders it to the screen. There is also a mode to zoom/pan and interact with the Image, however zooming and panning depend on user input and do not always produce consistent results. The only consistent result is that the image will be zoomed or panned, some values will be changed, but overall there are no functions in the program that perform testable duties.

Because of these challenges, we are focusing our testing mainly on Integration testing and Usability testing. Integration testing will allow us to ensure that our algorithm works well with our image viewer and that the components that make up the interview are resilient.

For Usability testing, we will work closely with our sponsor to ensure that it meets his expectations.

Unit Testing

This section should discuss your plans for creating unit tests aimed at ensuring that key methods and/or procedures function correctly. If you are not conducting detailed unit testing for each method and/or procedure, begin by outlining your rationale for only focusing on a subset of your system. Also discuss what unit test libraries you are using to streamline your testing activities and any test-related metrics you will be determining (test coverage, for example).

Continue by presenting a detailed plan for testing your code: For each unit of code you're testing, specify the equivalence partitions and boundary values you've identified and present selected inputs from these partitions and boundary values. Also make sure you include cases with erroneous inputs addressing the robustness of your code.

Due to the nature of this project, it is difficult to create a list of unit tests that could test all the functionality for this application. A majority of the application is rendering the image and the other half is generating image output into a buffer. We will not write a unit test for testing successful rendering, however we will write one to measure the output of our sampling algorithm.

Stochastic Algorithm Test

For this test, we will take the output from our stochastic algorithm and write it into its own image file. This is necessary in order to use some existing tools that come with the GDAL library. We will then run other algorithms built in to the GDAL library already (Nearest Neighbor, Average, Convolution, Bilinear, Cubic) and create another file based on the outputs from these algorithms. Then using gdal_calc, and gdal_info we can measure the difference between the outputs. This happens using these steps:

1. **gdal_calc**
    a. We will run gdal_calc against two image files, the first is the output from our stochastic algorithm, the second is the output from another algorithm in the GDAL library. The output from this command will create a new file containing the difference in pixel data between the two image files.
2. **Gdalinfo**
    a. We then run gdalinfo on the output file, this will return some statistics such as the minimum, maximum, mean, and standard deviation values. This is the minimum difference in pixel colors, the maximum difference, etc.
    b. Using this data we can create graphs that give an idea of how different our image is compared to other algorithms.

Algorithm Speed Test

A major aspect of our project was to decrease image render times compared to the currently utilized rendering algorithm. We devised a speed test where the image would be rendered several times in a row on large images supplied to us by USGS. The time taken to render the image was recorded and compared against the competition. The results of these tests are shown below:

LROC Image Test

Image size: 10 GB.
Resolution: 92160 x 40448 pixels

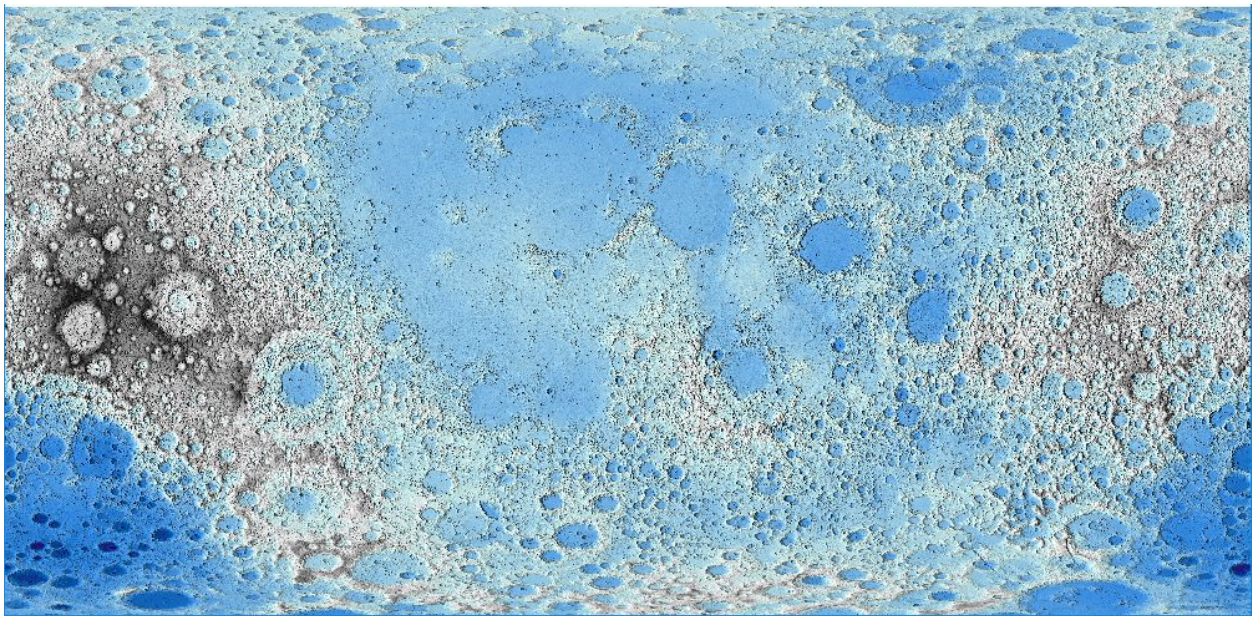
Figure 1: LROC image

Results:
The results of the LROC image speed testing are as follows:

Algorithm Used	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**	**Test 7**	**Test 8**	**Test 9**	**Test 10**
Stochastic	2.972	3.158	3.181	3.312	3.314	3.314	3.232	3.327	3.325	3.321
Nearest Neighbor	7.098	7.078	7.111	7.103	9.957	16.542	11.855	7.107	7.122	7.101

Figure 2: LROC rendering times, measured in minutes

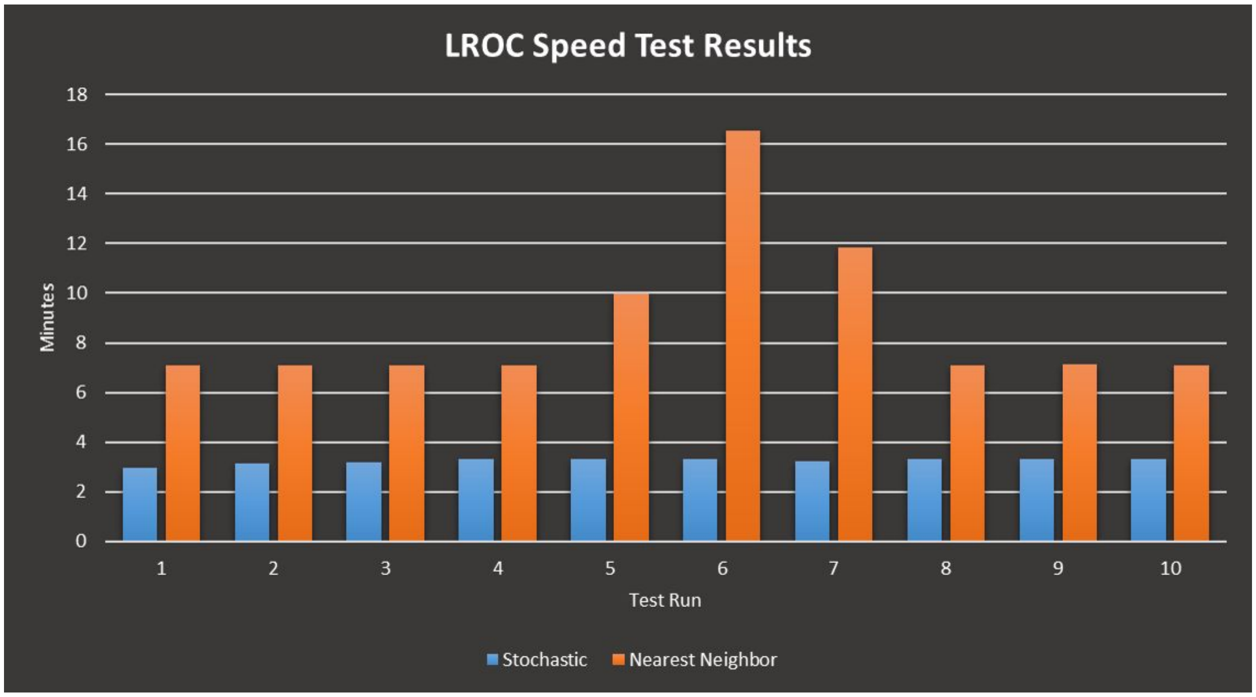

FIgure 3: LROC render times in graph form

The sampling rate was higher in this test to ensure that image quality was maintained. Our algorithm was still consistently rendering the image in under half the time of the nearest neighbor algorithm.

Kaguya Image Test:

Image Size: 500 GB
Resolution: 1474593 x 737297 pixels

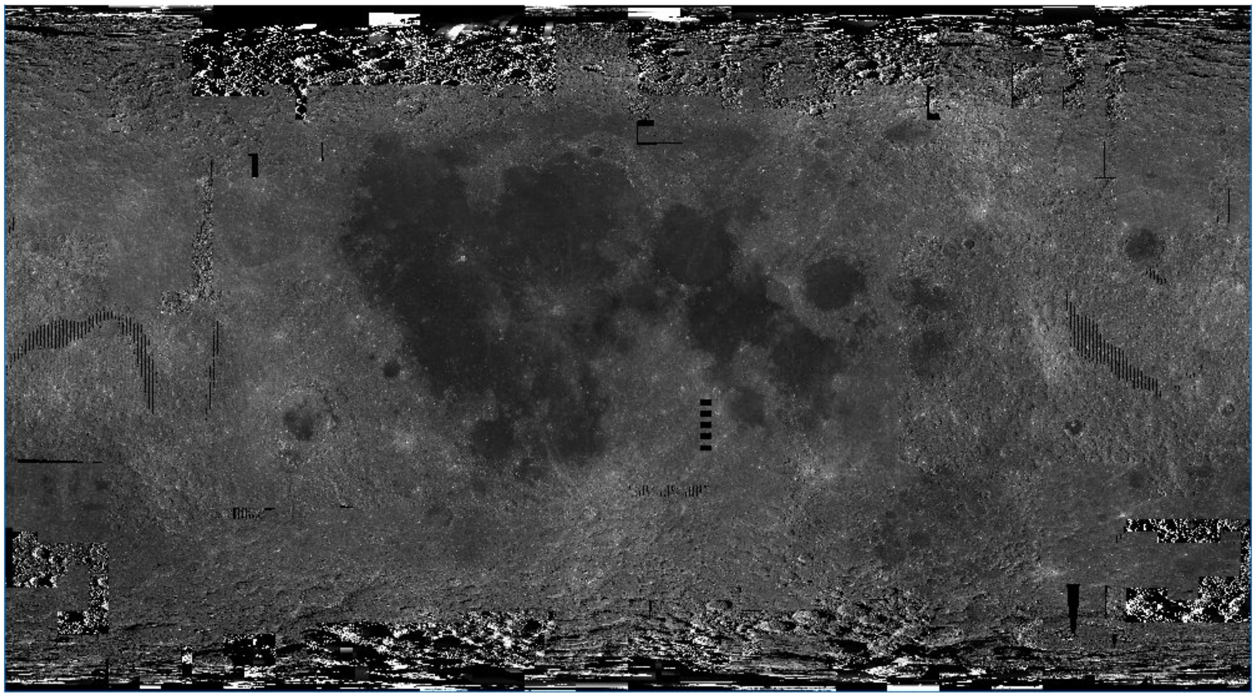

Figure 4: Kaguya Image

Results:

The results of the Kaguya image speed testing are as follows:

Algorithm Used	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**	**Test 7**	**Test 8**	**Test 9**	**Test 10**
Stochastic	4.862	4.698	4.721	4.733	4.710	4.725	4.736	4.73	4.737	4.713
Nearest Neighbor	45.751	45.709	45.671	45.696	45.701	45.701	45.723	45.727	45.715	45.723

Figure 5: Kaguya rendering times, measured in minutes

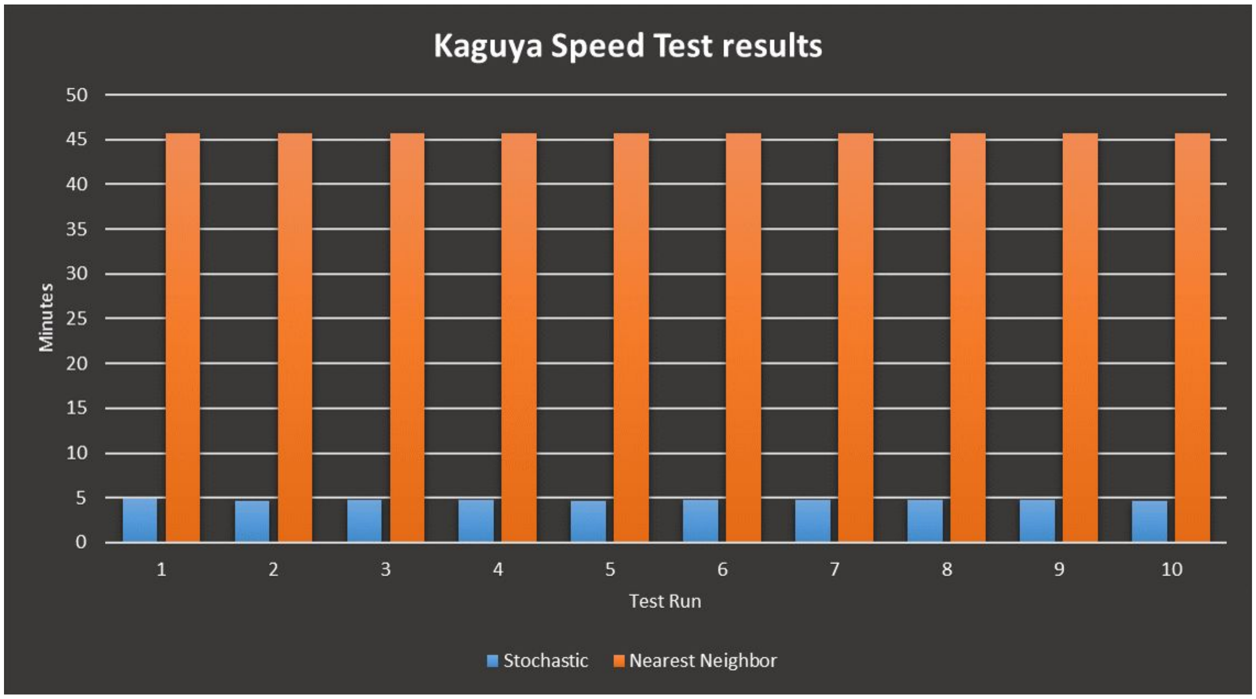

Figure 6: Kaguya render times in graph form

The stochastic sampling rate was lower for this test than the LROC test. The image produced was still within acceptable standards for our sponsor, and we were able to reduce the rendering time considerably, producing a rendered image nine times faster than the nearest neighbor algorithm.

Speed Test Conclusions

We consider this a successful test in that it demonstrates the superior image rendering speeds of our stochastic sampling algorithm. Image quality was significantly reduced in both tests, which will be explained in more detail in the user testing section of this document. One of the largest advantages of our algorithm is that we can change the sampling rate to find a balance between image quality and render times. This means that these render times could be even lower if the rate were adjusted. These tests results have helped us conclude that our algorithm is far superior when it comes to image rendering times.

Integration Testing

Integration testing is focused on the interfaces between major modules and components, and focuses on whether the interactions and data exchanges between modules take place correctly. At a very simple level, for example, while unit testing may focus on whether a method call returns the correct result, integration testing is focused on whether the data for parameters and return values is exchanged correctly. Even more simply, integration testing usually focuses on the "plumbing" of a system and if everything is wired together right. You should focus your integration testing on the "boundaries" between important modules of your system, such as those allowing access to databases, elements involved in the data exchange of a web-based interface, or network-based communication.

Present a detailed plan for testing the integration of major modules in your code: For each integration point, specify what test harnesses you used and how you verified that modules integrate correctly, ensuring that interactions take place correctly and all contract assumptions (such as data needed to invoke a function and data required to be returned) are respected.

Usability Testing

Usability testing is focused on the interactions between the software system and the end user, and is intended to ensure that users can effectively access the functionality provided. This type of testing examines the overall quality and understandability of the user interface exposed by your system and the workflow that your system embodies.
Present a detailed plan for conducting usability testing of your system, which will vary based on the specific needs and focus of each project. Discuss the appropriateness of your thoughts on this type of testing with your mentor. You might, for example, leverage focus group techniques to gather qualitative data about the usage of your system, user studies where you record and analyze user interactions with your software, or expert review.

There are not very many features of our application that we are able to subject to usability testing. Our main focus for this section are the implemented features of the image viewer. These features are image zooming to increase or decrease the image size in the viewer and image panning to allow movement of the image after zooming in.

Our plan for the usability testing of the Image Viewer and its features is utilizing paired testing groups. This would allow for appropriately modified code and ensures that the speed and quality of the features was captured while guaranteeing that the image manipulation functions worked in the Image Viewer. Including a pair of ourselves we had several users test the performance and usability of the image manipulation features. In our pairs we would focus on the transition rates as well as the fluidity of how well the pan and zoom functions worked. We tested the speed of each feature to ensure that their use would not only be simple but also make sure that the user wasn't hindered in their examination of the image. We also tested to ensure that image when using either of the features would not only keep original quality but take the feature of increasing while waiting in its use like the base image results. This testing resulted in finely tuned interface features that allow for precision and speed when the user is manipulating the image.

We took this testing a step further by working with our sponsor to ensure that these features meet their needs and are to their specifications. We were provided with a hard drive from our sponsor to test even larger images that our current machines couldn’t process due to space constraints. With that we could test these images that were actually gigabytes in size to see the quality of the image in comparison to the competing algorithm within our viewer. This resulted in a satisfactory display that our sponsor could accept to see if the image had been properly stitched together.

Quality and Image Manipulation Testing

Quality Testing

With results shown we based future testing off of how these were received by our sponsor. Then with that we could build a stronger more effective viewer.

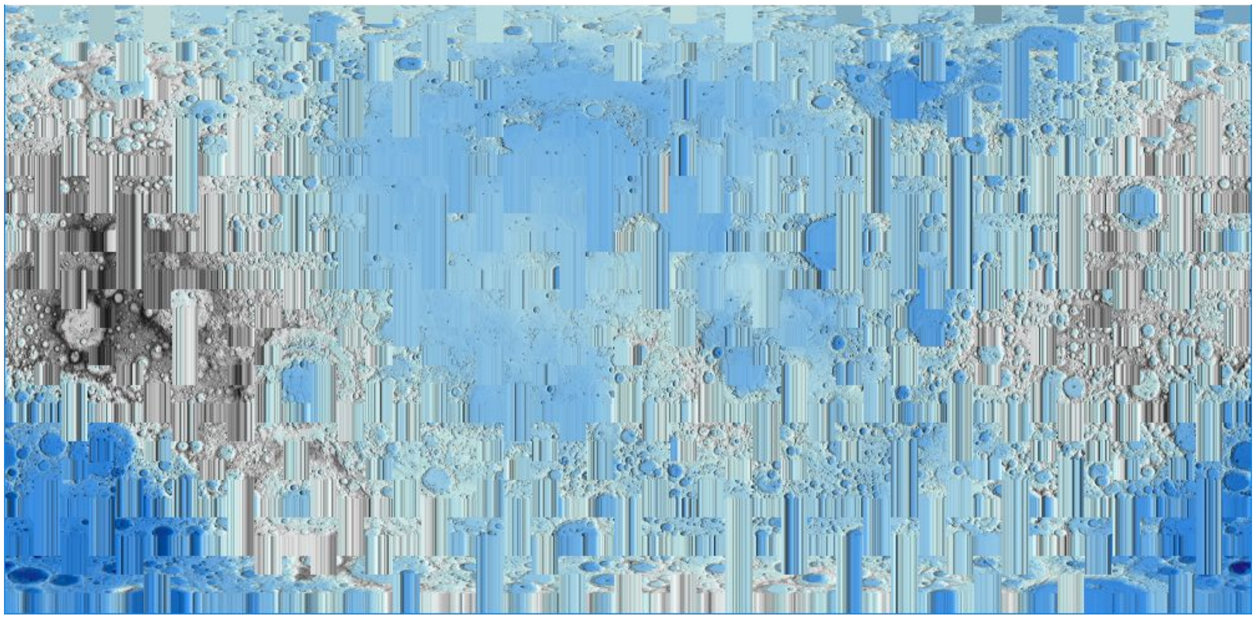

Figure 7: LROC Image using our viewer

The overall image quality is lower in comparison to the nearest neighbor image shown in Figure 1 yet this quality was deemed acceptable so for future tests we worked to determine what could be the lowest sampling rate that was acceptable to display.

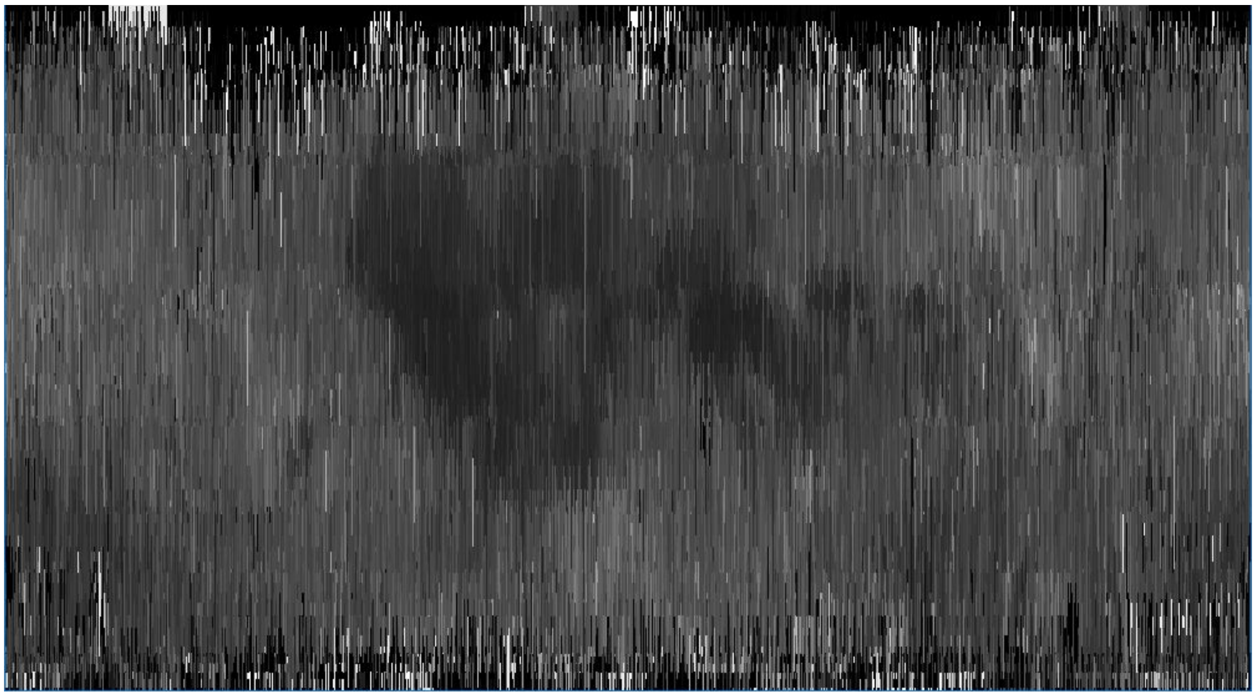

Figure 8: Kaguya Image

As with the LROC image, the Kaguya testing produced a lower quality image than shown in Figure 4, however an outline of the some of the major formations can still be seen and has been determined to be of acceptable quality for our purposes.

Image Manipulation Testing

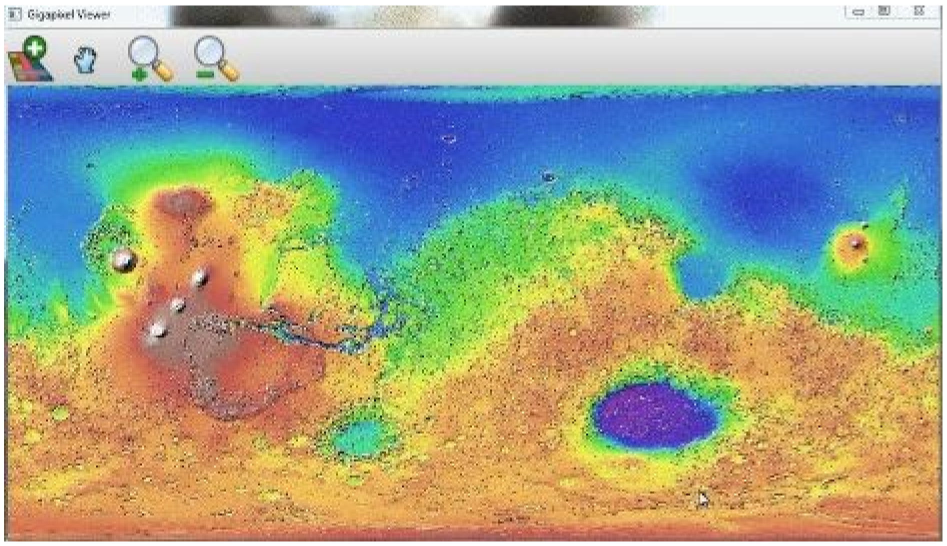

Figure 9: Example of image with viewer

This image was used as a basis for using the manipulation features as well as the overall viewer look.

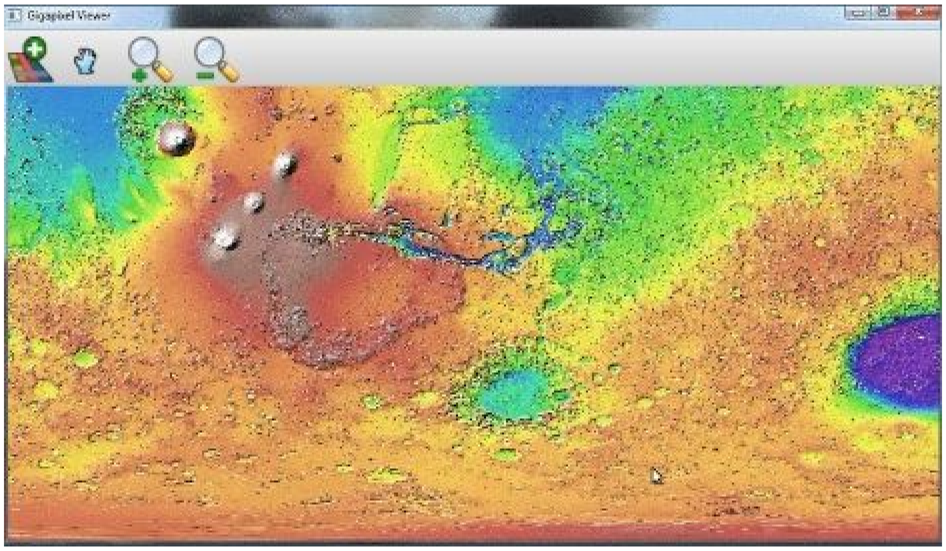

Figure 10: Same image as Figure 9, zoomed in to demonstrate features

This is a base test of how the image manipulation features work and from how it ran when in use we had the groups test and see what speed felt the most comfortable as well as if the quality improved while using the features. From the results we made plans to edit and configure the speed rates and continue seeing what made users feel best with the feature.

Future Quality Testing Plan

Since our sampling algorithm allows the sampling rate to be modified, we can utilize this in our quality testing with potential users.

The plan is to show users the image first in standard quality where they get an idea of what the image is supposed to look like. Then we show them the same image rendering with the stochastic algorithm with a very low sampling rate. So low that it is almost unrecognizable. From here, the sampling rate should be slowly increased until the user is able to again recognize the image. The sampling rate at that time will be recorded