Final Project Report

Gigapixel Image Rendering

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

Table of Contents

Introduction

With continued advances in space exploration, the need to efficiently process the large quantity of data received from space continues to grow exponentially. One important piece of data is the massive images of other planets, usually several thousand to several million pixels in size. The current methods of viewing these images is by either building pyramids or by creating tiles which can often take anywhere from a few minutes to a few days. Building pyramids also consumes a massive amount of disk space. We plan to provide researchers with a method of quickly viewing these images and a tool in which to view the images. The method we will provide is a stochastic sampling algorithm and the tool will be a lightweight image viewer. Our image sampling method, combined with our image viewer, will provide researchers the ability to view these massive images in a manner of minutes instead of hours.

Our system will be used not only by USGS but by any company in industry that wishes to use GDAL. Our system will greatly reduce the amount of time that GDAL users take to do something as simple as looking at an image. Some key risks that we had to face included hardware incompatibilities, learning new softwares and systems, and image quality loss.

Process Overview

During the development of our system we stuck to a waterfall method of approach. This means that we initially started the project by gathering requirements and working on the design before we started developing the software. The different tools that we used during the course of the development include but are not limited to:

- Github: Github is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.
- Trello: Trello is a collaboration tool that organizes your projects into boards. In one glance, Trello tells you what's being worked on, who's working on what, and where something is in a process.
- Slack: Slack is an instant messaging and collaboration system that allows teams to communicate with each other as well as providing other functionality such as Github integration and file sharing.

There was four primary roles in our team which were the release manager, the customer communicator, the architect, and the team leader. Daniel Garcia-Briseno was the release manager, Matthew Ostovarpour was the customer communicator, Thomas O'Brien was the architect, and Douglas Peterson was the Team Leader. While the lines between roles were definitely blurred at some times during the course of our project, everyone always contributed as best as they could. Our team met at minimum once a week to work on the project and different deliverables for the class and also to check up on each other and see our progress.

Requirements

Functional Requirements

The functional requirements for our product are as follows:

1. Input
    1. Terminal Input
        1. User can enter the following parameters via command line:
            1. file to open
            2. window width
            3. window height
    2. GUI Input
        1. User will be able to interact with the window with the following controls:
            1. Enter/Exit zoom mode
            2. Enter/Exit pan mode
2. Usage Modes
    1. Zoom mode
        1. User may zoom in on the image.
        2. User may zoom out from the image.
    2. Pan mode
        1. User may use the “WASD” keys to move the image while it is zoomed.
    3. Idle mode
        1. Image quality will improve over time in this state.
3. Output
    1. Create a window to display the image.
    2. Interactive controls for viewing the images.

Environmental Requirements

1. Cross Platform
    1. Ability to run on Windows
    2. Ability to run on Mac

3. Ability to run on Linux

2. Software Libraries and Programming Languages
    1. Software Libraries
        1. OpenGL for cross platform functionality
        2. GLEW (GL Extension Wrangler) for modern OpenGL Functionality
        3. GLFW (OpenGL framework) for cross platform windowing
        4. GDAL for reading image files
    2. Programming languages
        1. C
        2. C++
3. Other external constraints
    1. Our team will only be using free and open source software (FOSS) to create the reader and viewer.
    2. Supported image formats (courtesy of GDAL)
        1. http://www.gdal.org/formats_list.html
        2. http://www.gdal.org/ogr_formats.html

Non-Functional Requirements

The reader must perform faster than readers that are currently available.

The most important part of our reader is that it needs to render gigapixel images much faster than the current standard. The current method is to build pyramids to render the images slowly after which they are available to be viewed. We are using random sampling (stochastic sampling) to drastically increase the speed in which images are rendered.

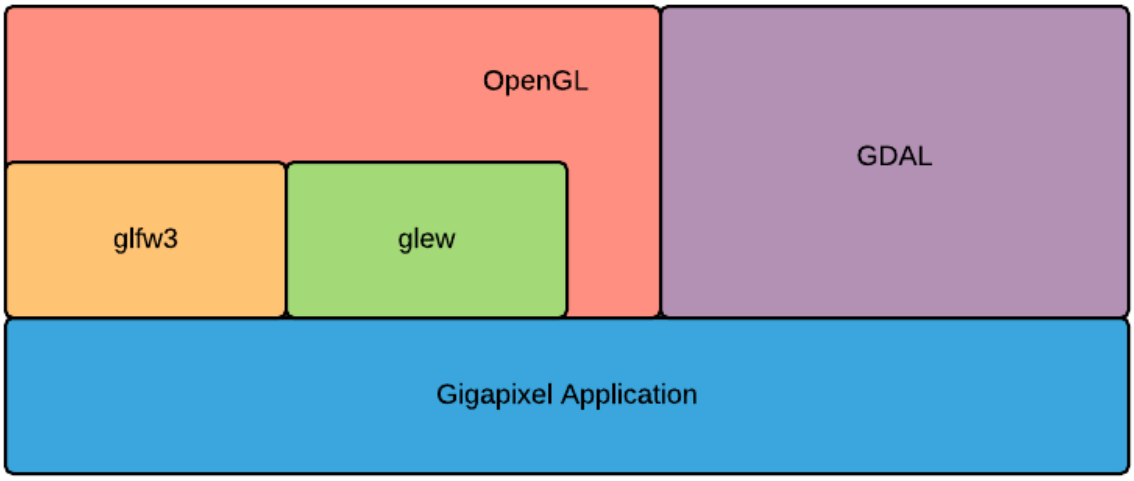
The reader must produce a visible, quality image.

Our reader must also produce an image of an acceptable quality. The pyramid method of rendering already produces high quality images which are to our sponsors standards. We don't expect our reader to produce an image of the same quality, but we do expect it to meet the standards that our sponsor desires. Stochastic sampling with a small sample size could result in a rendered image with very poor quality. Our reader must have a minimum sample size where the quality is at its lowest acceptable standard. The emphasis of our reader is on speed, not quality, but both of these factors are important for the project.

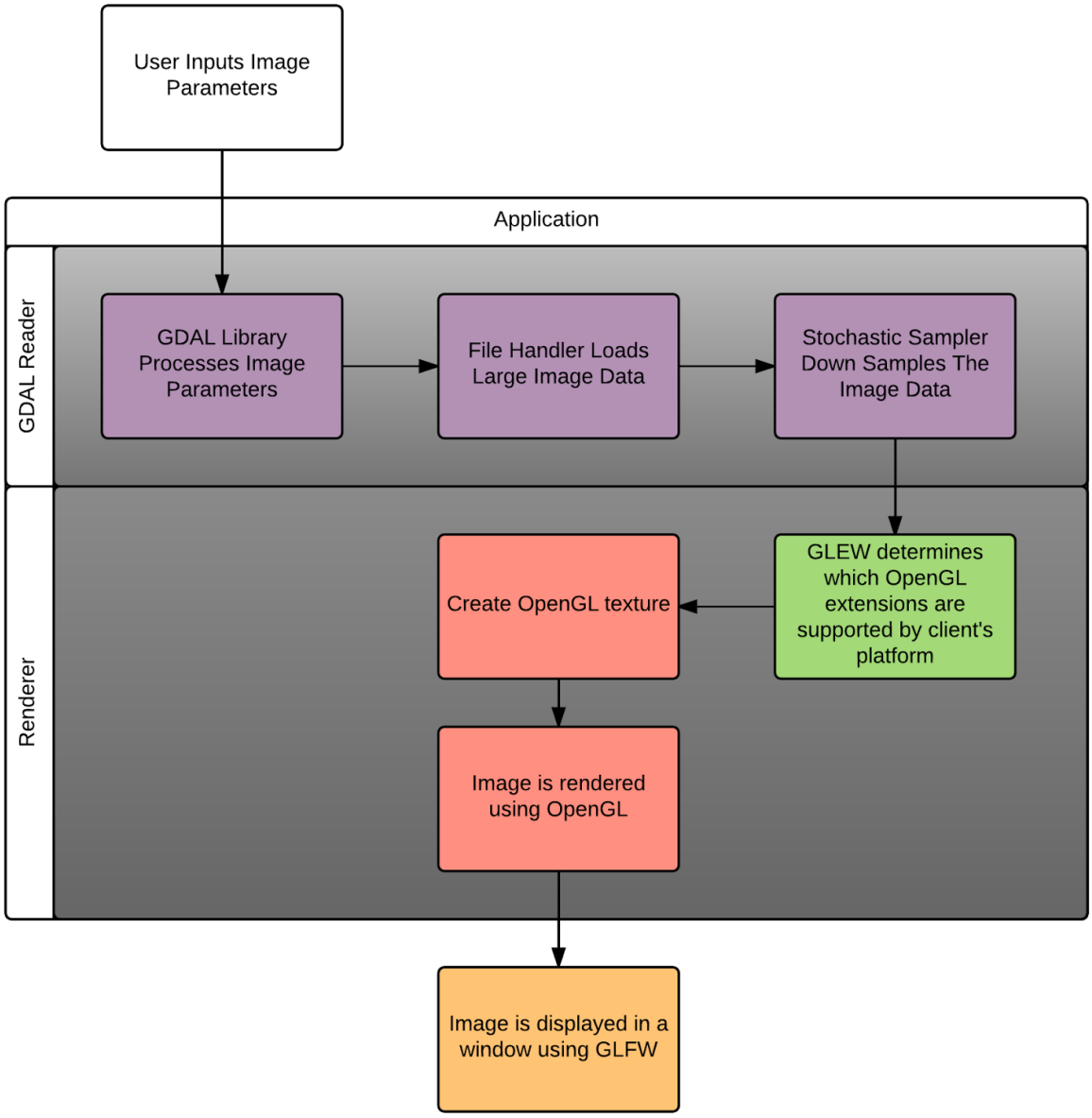
Architecture

The application was designed in a simple pipe and filter style architecture. The technologies used were: OpenGL, GLFW3, GLEW, and GDAL.

Technology Stack



Pipeline Diagram



Testing

For testing both our algorithm and Image viewer 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 focused our testing mainly on Integration testing and Usability testing. Integration testing allowed us to ensure that our algorithm works well with our image viewer and that the components that make up the interview are resilient. As for Usability we worked with our sponsor to ensure he was satisfied with the quality and smoothness f our manipulation features.

Stochastic Algorithm Test

For this test, we took the output from our stochastic algorithm and wrote it into its own image file. This is necessary in order to use some existing tools that come with the GDAL library. We then ran other algorithms built in to the GDAL library already (Nearest Neighbor, Average, Convolution, Bilinear, Cubic) and created another file based on the outputs from these algorithms. Then using gdal_calc, and gdal_info we measured 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

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 was expected. 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.

Usability Testing

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 focused 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.

Since the overall functionality of the viewer and manipulation tools works and the display for the quality of the images is within acceptable standards our testing has been prosperous. We have taken the information gained and modified the speeds for the manipulation tools to the best approved rates. These tests have resulted in finely tuned interface features that allow for precision and speed when the user is manipulating the image.

Future Work

Because of the amount of progress we have made on our project future work would be more focused on refactoring what we have already done. Re-writing our code base would increase readability and reusability and adding a few tweaks to our stochastic algorithm implemented inside of GDAL would possibly make it more efficient. Another part would be to implement our modifications that we made to GDAL into the main GDAL repository.

Appendix

Glossary

GDAL: A computer software library for reading and writing raster and vector geospatial data formats. Used in our project for image library and file handling along with where our stochastic sampling algorithm is located.

GLEW: The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library. Used in our project to load the latest OpenGL extensions according to the user's platform.

GLFW3: An Open Source, multi-platform library for creating windows with OpenGL contexts and receiving input and events. Used in our project to create the window in which the image is displayed to the user.

OpenGL: A cross-language, cross-platform API for rendering vector graphics. Used in our project to render the image.

Stochastic: A system that is unpredictable due to the effect of a random variable.