GeoPointClouds Application Final Report

Clients: Dr. Zane Jobe and Nataly Chacón Buitrago
Advisor: Dr. Wendy Fisher
Team Name and Members: CSM Jobe 1 - Samuel Stech, Ethan Taylor, Mitchell Cutts
16th June 2021

Table of Contents

Introduction

Client Background

The Chevron Center of Research Excellence is a research group in the Colorado School of Mines' geology department that focuses on identifying stratigraphic architecture, scaling relationships for depositional systems, and investigating source-to-sink sediment dispersal. The applications of their research generally apply to the oil and gas industry, but many of the solutions they create can also be used in exploratory geology. Currently, the group is working on integrating machine learning solutions as well as developing software solutions to better supplement and streamline their research workflow.

Product Vision

The client had requested an application to manage and visualize point cloud data generated from drone photos of rock outcroppings. This product has a user interface that allows users to label the data through the capability to select polygons and volumes on point cloud and add a corresponding name, color, and description to the labeled data. Ultimately, the project was wrapped as an executable that is downloadable from the github repository provided by the client.

Our client placed emphasis on ease of use for geologists without programming experience as well as the creation of abundant guides and documentation for scalability of this open-source software. To accommodate this request, our team has elected to provide two forms of documentation: one for users of the program and one for developers to continue building the documentation.

To put the functionality of this product into context, an application for managing and labelling point cloud data would be useful for the client because they can perform geological investigations on the outcroppings while not in the field by being able to interpret 3D recreations of the data. A potential extension to the product is the functionality to merge labels of the same characteristics. The user would then be able to see the labels in the point cloud (visualization) and generate an output file with information of each label, such as positional, color, normal, and classification information for future use.

In conclusion, this application will be used for the client to trace geometries and polygons on rock outcropping point cloud images to better quantify the thickness and variability of how geological features are stacked in the subsurface of the Earth.

Requirements

Functional Requirements

Create a functional python package to (in prioritized order):

- Import and export point cloud data supporting several formats:
    - .txt and .csv with formats: [x, y, z, normal_x, normal_y, normal_z], [x, y, z, r, g, b], [x, y, z, intensity, r, g, b]
    - add handling for further formats including .pcd and .ply.
    - photos: .png, .jpg
- Select polygons of the point cloud and add a name and/or description in string format.
    - This implies building a graphical user interface that has interactive capabilities.
    - This implies developing or using a graphics display program that will visualize the point cloud.
- Draw lines between points on the point cloud to demonstrate geological changes over time.
    - This implies being able to select points on the point cloud in the graphics display program.
- Manage data classifications as well as their corresponding names, descriptions, and colors for display on the labelled point cloud.
- Wrap all functionality into a file executable on MacOS and Windows.

Non-functional Requirements

- Two methods of useful documentation: our program must have documentation available in Sphinx format for developer documentation as well as a github readme for user help.
- Our client requested that we repurpose code from other open source libraries to save time, as well as improve upon current code created by the client to make it more efficient and expandable.
- Perform research on related industry solutions to this problem for different use cases and new solutions. Several paid versions of this product already exist in some capacity, so our client has suggested we do research on them to understand interface examples.
- Utilize AGILE and Scrum best practices to allow for future development and working software.
- Emphasize ease-of-use and interpretability for users without programming experience.
- Develop code incrementally and utilize the client-provided github repository for development.

System Architecture

Development issues:

Over the course of this project, our team was presented with several different solutions that we could have implemented for this project:

- Utilizing Open3D and PPTK brought concern that we would be completely limited by what the libraries had to offer. While we accomplished the functionality our client requested, we were wary of these libraries at first.
- OpenGL would provide a high level of flexibility for whatever solution we ended up developing. The downside is that we would have to implement a good amount of the functionality ourselves.
- Contributing to a current open source library is a topic that none of us had experience in; it appears that many of the libraries have extremely complex architecture and development chains such that there is the possibility our developments would never get accepted for future use by our client.

We have listed this as a technical design issue as this decision greatly affected the outcome of the project. After evaluating the pros and cons of each method, we put together small demonstrations for the OpenGL and Open3D implementations. After conferring with our client, it was decided, despite our initial impressions, that we would end up designing an application utilizing Open3D instead of OpenGL.

Design Architecture and Graphics:

The architecture for the GeoPointClouds application system is outlined in *figure 1* on the next page. The components of the system are as follows:

File I/O: The import dialog and export menu handle all user I/O interactions and all requested I/O functionality through the use of flexible import and export scripts our team has written. These scripts handle all file types mentioned in the requirements.

Main Menu: The main menu has five options to choose from:

- *Edit/View Labels*: Allows the user to choose which rock type will be selected when using *Select Volume*, provides the option to create new rock types and edit their name, color and description.
- *Select Volume*: Allows the user to select bounding points on the point cloud to label all points within the boundary as a certain rock type classification.
- *Draw Line*: Allows the user to select points and draw lines between these points on the point cloud.
- *Export/Render:* Opens file export menu for user to select export options. Returns to the main menu after completion or cancellation of export.
- *View Point Cloud*: Displays the point cloud for user reference.

Help Page: Links to our Github readme and user help guide, which can be referenced here.

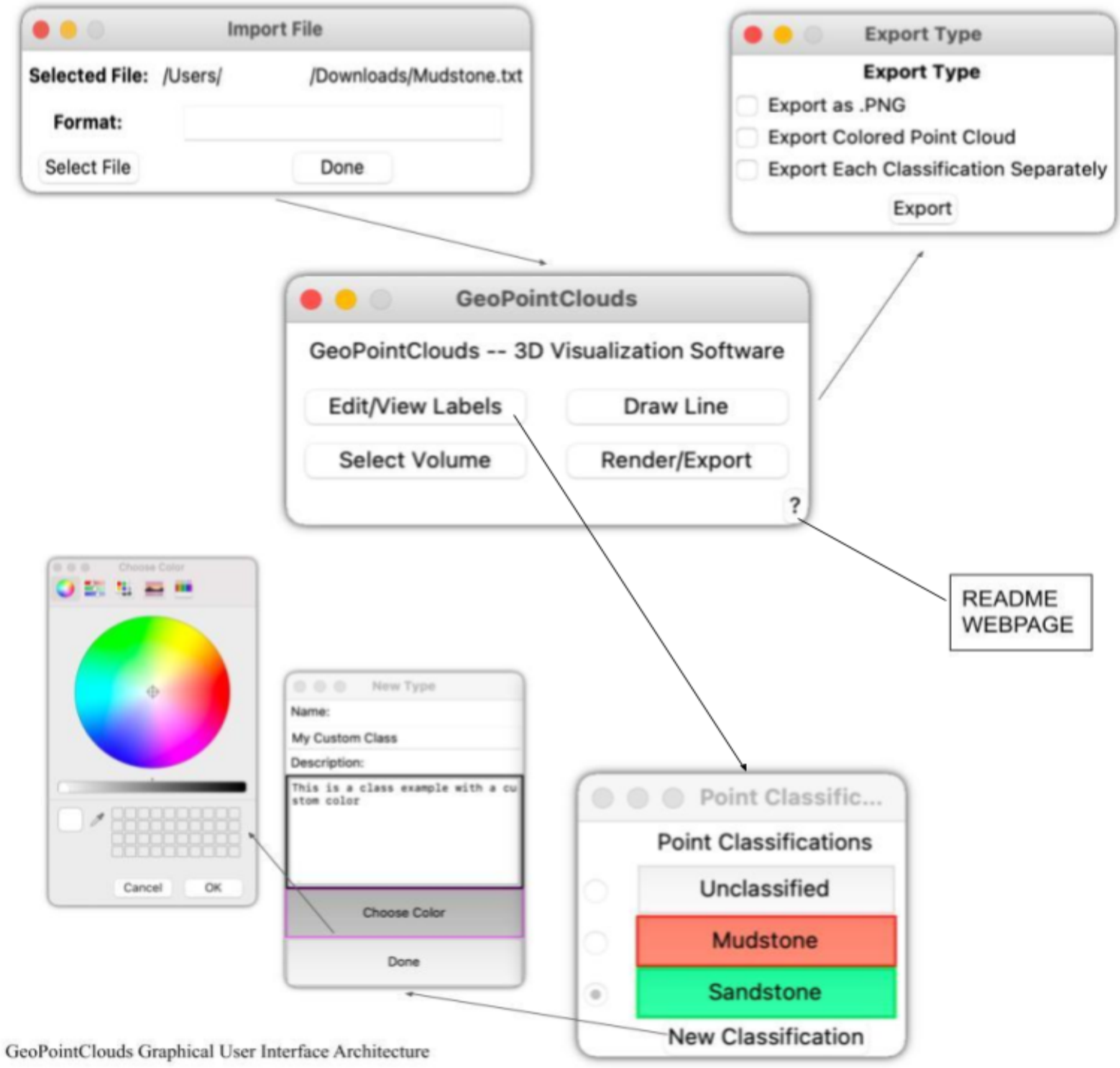


Figure 1 -GeoPointClouds Graphical User Interface and System Architecture

Technical Design

Data Management Design

In a data-heavy, interactive application, optimized data organization is vital to both performance and functionality. For the majority of our technical design, our team focused on designing data architecture that would allow the user to interact with the point cloud data in a natural way while retaining important information that will be needed for export. The two main classes for data management are shown below in the UML diagrams for PointData and Classification.

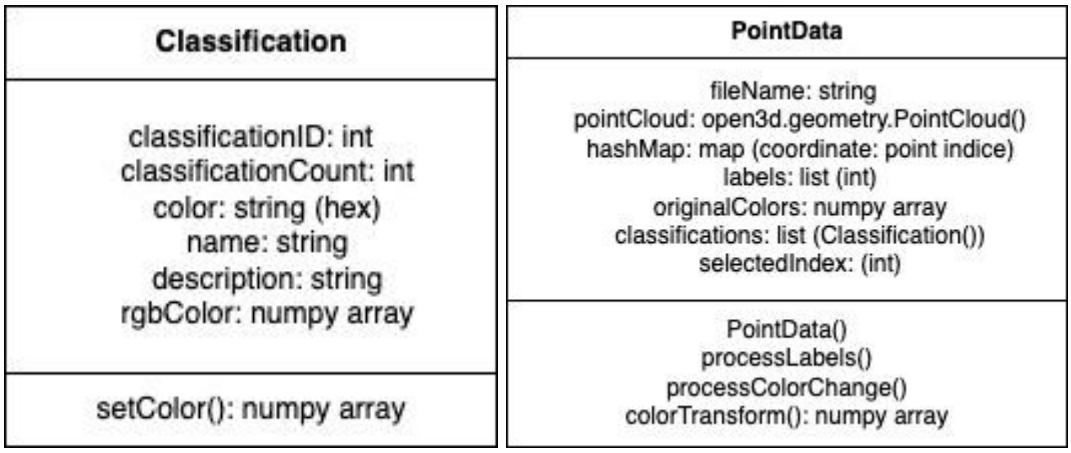


Figure 2 -UML diagram for Classification and PointData

In this application, all relevant data for displaying the point cloud will be stored in the application's PointData class. Some descriptions of the members are provided below.

fileName: the name of the file imported by the user.

pointCloud: an open3D point cloud data object, which is used to display the working point cloud in open3D. This object has members "points", "colors", and "normals", and the member manipulated by the user in this case is "colors".

hashMap: for efficient volume selection, PointData holds a hashMap that can lookup a given point's index based on the coordinate values of that point. This allows us to check if a point exists within a user-selected volume in the cloud.

labels: a list of the associated Classification a point has. This allows us to color points with different labels and retain user-selected labelling information of the class as each point's corresponding label represents a rock type through being equivalent to the rock type's index in *classifications*.

originalColors: because the user will manipulate the colors in the pointCloud, this member serves as a way to recolor the point cloud back to its original state as well as export the original point cloud in post processing.

classifications: a list to hold all classifications created by the user when labelling the point cloud. This is a list of type Classification() which is used to define a type of rock, give it a description, and assign it a

color. This information will allow us to color the different classes on the cloud, maintain information about each class, and process different labels.

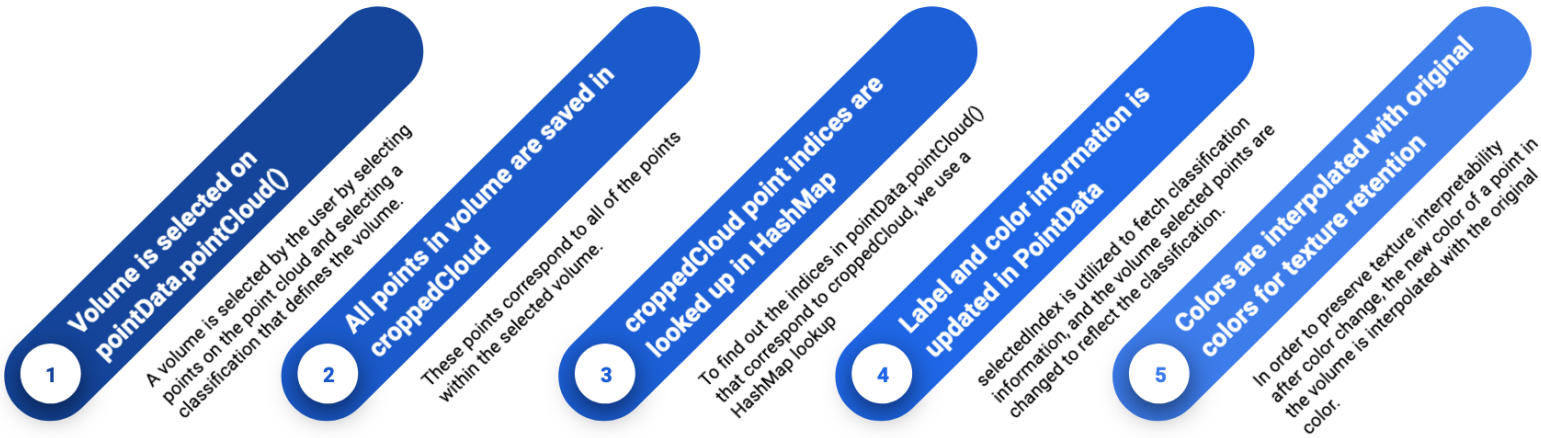
selectedIndex: the index selected by the user in that they are performing volume select on.

The members of the PointData class that are manipulated by the user of the application include fileName, pointCloud, classifications, and selected index. The others are used for data retention, logical organization, and export capabilities.

Volume Selection Use Case

Because the main function of our application is labeling of rock types through volume selection, a good example that represents our intentionality in data organization design comes through our volume selection process. When the user selects the points to form their volume in the point cloud, they will be prompted to select or create a Classification() in which to label the volume and this chosen classification and its index in the *PointData.classifications* list will now correspond to *selectedIndex*. From here, an open3D window will launch and the user will be able to select points on the cloud so that our application can form a volume based on the indices of the points chosen. This volume is based off of a bounding polygon containing a defined orthogonal axis, which for our case is generally 'Z'. After defining the volume to crop the point cloud on, the main point cloud *PointData.pointCloud* will be subsetted into a new point cloud that contains all points within the volume. From here, our program will now need to perform some processing tasks to manage this information.

Because our program maintains a single point cloud for display information (*PointData.pointCloud*), we must find which points in the cropped point cloud *croppedCloud* correspond to indices in the main point cloud *PointData.pointCloud.* To do this we utilize the *HashMap*, which performs lookups on the *croppedCloud*'s point data and returns the point's corresponding index in the *PointData.pointCloud*. Now that we have the indices of the points within the volume, we can change the points colors to the color of the classification, as well as the points labels to the index of the classification. This process is visualized below.



Quality Assurance

Code Quality & Metrics

Code quality is a team emphasis of ours as our application is very complex despite having a small codebase. To maintain quality, we intend to perform regular code reviews in conjunction with paired programming implementations to encourage clean code that is functionally and stylistically correct. Other methods that we used to encourage quality code are data format verification, unit testing, user acceptance testing, and user interface testing throughout our progress in the AGILE process. While the client has not stressed specific code style and syntax guidelines, we have defined our best practices in a style guidelines document in our repository. Moreover, due to utilizing the Sphinx library for documentation, we have also defined our comment and doc string requirements to be able to maintain documentation in the same style document. Ultimately, our style guidelines provide information on defining constants, classes, variables, commentation, function descriptions, enum types, and other naming.

Testing

Integration Testing:

- Defining GUI selection paths to test our code, Quality Testing on MacOS, Windows 10

Manual testing:

Given that we are developing an application that is mostly GUI based, a good portion of the testing performed was in the form of us going through the motions of handling the application. We hoped to uncover most of the bugs that we can through manual testing ourselves, which left the client room to give us feedback on ease of use and overall structure.

User Acceptance Testing & Code Management

Our product will be used solely by our client initially; however, our product must also be easily scalable as our client has stressed their intention to build on it further in the open source world. We were in contact weekly with our client to discuss our progress and envisioned functionality, and both parties have been realistic in defining our expectations and necessary modifications to the product. Since our user acceptance testing has been carried out incrementally, we have gotten a mutual understanding with our user about what our project's limitations, capabilities, and expected functionality are.

In terms of code management, our client has supplied us with a github repository (https://github.com/nchaconbgeo/pointcloudpackage) in which to store our code and changes over time. We regularly update this repository with our version control changes, and this helps communicate our progress to our client.

Results

Our project has been tested and is working effectively in MacOS 11.0.1 and Windows 10. We have begun tests in Ubuntu 20.10, but are still testing at the point of submission. Currently, all non-functional requirements have been fulfilled for the project, with two types of documentation available for both technical and non-technical users. Our program has also been tested on point cloud files of up to two gigabytes and handles all required functionality correctly with the exception of point cloud text labeling in Open3D. Two small remaining issues about the project is that the volume selection functionality could be improved upon to be more user friendly, and the Open3D window could be developed to be non-blocking using threads alongside the graphical user interface.

Usability testing has been conducted throughout the development of the product. All functionality was not quite in place at the time of our last meeting with the client, but the client overall seemed pleased with what we had. We are meeting up with the client during the final week of this course to deliver the final product and conduct the last usability test with the client. With that said, in the future, this product can be developed further by incorporating labelling libraries such as LabelMe for image labelling as well as adding statistical methods to the back end to better classify and sample point cloud data. Another future development could be developing methods to selectively remove points from the dataset to make the program faster (known as downsampling); this would require upsampling (retrieval of these points) in post-processing but would provide more performance without compromising interpretation capabilities. To aid these additions, we have focused on building the product with documentation and incremental development as a priority such that the client can eventually build the product in its complete vision.

Lessons Learned

- For projects in areas which you have limited expertise with, plan to spend about half of your time reading, researching and diagramming.
- When developing software on a time crunch, see if you can leverage other products to achieve your end goal in time. For example, our product would have consisted of developing a python graphics library with all required functionality from scratch if we did not find open3D after a week of research.
- Code management in python can get messy, especially when developing in a single repository. Constant clean up efforts and repository reviews are needed to maintain clean and working code.
- Before accepting a large project, be sure to clarify with your clients what exactly the project will entail. We all joined this project with differing ideas of what the project would look like; next time we would probably reach out to the client to get more information before committing to it.
- When using a library in python for development purposes, you will generally be limited to the functionality of that library and will lose flexibility in development and scalability. In our case, open3D did not provide for a great interface for volume selection and limited our ability to manipulate graphics in real time due to its blocking functions and polygon selection.
- Careful documentation and code metrics will save you from a breakdown later in your project. Our team was very careful with both of these subjects and it has saved us so much time on the back end.

Future Work

Some recommendations for future work on the product are defined below:

- Integration of machine learning methods for label generation and feature extraction would be a great addition to this project as the user would have the added capability to label and interpret data using statistical information rather than just visual volume selection.
- Improving the volume selection methods would allow for a more precise experience as the user. While the volume selection methods used in this application work, some added suggestions to improve the current methodology include utilizing nearest neighbor information, applying projection-based volume selection using camera information, and generating convex hulls or meshes to test point presence in a selected volume.
- Integration and improvement of client-written subsampling methods for point cloud data. While this is out of the scope of an application, developers could implement the subsampling methods provided by the client while leveraging our data organization to allow for scalable and efficient code for statistical analysis.
- Added functionality for outlier detection and removal in labelled dataset as well as color transformations on the labelled dataset.
- Added functionality for Jupyter Notebooks integration.
- Utilize threading to allow open3D to be non-blocking to the GUI.
- Add downsampling and post-rendering functionality to accommodate larger file sizes.

Appendix

To read more about this product or to download the GeoPointClouds application for use on MacOS 11.0.1 or Windows 10, please visit https://github.com/nchaconbgeo/pointcloudpackage. We encourage other programmers to contribute to this project as it is open-sourced and well documented.