Group Report

Antarctic Chasm One
2 March 2017

Client

Matt Polaine

British Antarctic Survey

Team Bravo

Julia Bibik, Andreea Deac, Matthew Else,
Nand Kishore, James Lomax, Valeria Staneva

1 Project Successes

We successfully built a pipeline that has the following stages:

1. Preprocessing the data

2. Storing it into a database

3. Creating an interface between the database and the renderer

4. Displaying a visualization from the renderer

This meant integrating different programming languages (Python, C++, SQL) and tools (OpenSceneGraph [1], Qt Creator [2], PostgreSQL [3], Snappy [4]).

1.1 Preprocessor

1.1.1 GeoTIFF Data

The elevation data for the area surrounding both Chasm 1 and Chasm 2 is provided in the form of a 20373×25334 px image, encapsulated as a TIFF file alongside GIS metadata. Since this is a 16-bit grayscale image, this would produce 1.03GB of uncompressed raw data. However, given the sparseness of the data, we can easily reduce the disk space required by using compression.

Multiple algorithms exist to provide the amount of compression we need. However, due to the large amount of data involved in the visualisation, we can trade off compression ratio in favour of (primarily decompression) speed. A simple benchmark [6] was constructed to determine the relative speeds of different compression algorithms. It should be noted that, for simplicity's sake, these compression and decompression benchmarks were written in Python (in all cases calling C libraries in the background [7,8,9,10]), and although some variation should be expected between implementations, these should be representative enough for our purposes.

Algorithm	**Average Decompression Time (s)**[1,2]	**Minimum Decompression Time (s)**	**Maximum Decompression Time (s)**
Snappy	0.000433	0.000193	0.017855
GZ	0.000697	0.00041	0.0019

[1] Across all 500x500px sub-images of the chasm GeoTIFF file.
[2] Performed on a quad core intel i7 with 16GB of RAM

BZ2	0.008521	0.001253	0.034648
LZMA	0.00225	0.001088	0.013187

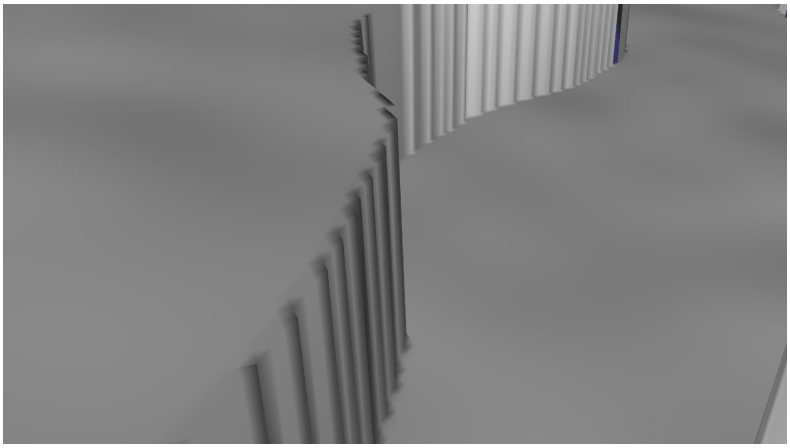
Algorithm	**Total Compressed File Size (Compressed/Uncompressed %)**
Snappy	97239049 (9.42%)
GZ	24111291 (2.34%)
BZ2	17688967 (1.71%)
LZMA	17334824 (1.68%)

1.1.2 Bedmap2 Data

Using the input from 3 files [11] - ESRI grids for the Surface (Elevation) of ice, ice thickness, and the ice bed (hard rock beneath the ice), the preprocessing script generates a .csv (Comma Separated Values) file. The output has a simple header which describes what type of information each row contains: x and y coordinates, Surface, Thickness and Bed.

1.1.3 Smoothing the edges

The uniform distribution of the data (M points per N square meters in both flat areas and edges of a crack) meant that the information density for some parts exceeded our needs, while in others it was too sparse to create a realistic visualization.



An example of side-effects of sparse data.

A solution we came up with was creating a Python script that took all consecutive pairs of heights and, detecting a big difference, added intermediate points. Such interpolation within the original height interval improved the issue of sparseness. At the same time, removing points with a negligible difference in heights solved the problem of dense data.

1.2 Store

PostgreSQL with the PostGIS extension databases store the output from preprocessing the Geotiff and Bedmap data.

For the first type of data, the table has the following columns: id, x and y coordinates and data of type bytea. Each record corresponds to a file compressed using Snappy and is stored as a binary encoding of the compressed file.

For the Bedmap2 data, data from the .csv file was used to fill the database. In addition to the data from the file, a geometry column was generated in order to be able to more efficiently answer to queries such as choosing the points which correspond to a certain tile.

1.3 Interface

The interface to the Datastore allowed the render to make spatio-temporal queries to the PostgreSql database, process the results, and return the data in a format easily useable by the renderer. The api was designed so that all the work of connecting to the database, constructing the raw sql queries, and parsing the results would be abstracted away in the interface layer, so that the renderer could focus on the graphics part of the project and not the data processing. The Datastore interface was built keeping object-oriented paradigms in mind so that it would be easily extensible for new data sources.

The final design consisted of a Datastore class that contained general methods applicable for all data-types. These methods included connect() for connecting to a database, run_query() for running an sql query against the database, and processDbRow() for processing a row retrieved from the database. These low-level methods such as connect() and run_query() were implemented using the libpqxx library, which is the official C++ client API for PostgreSQL[5].

Much of the actual implementation was in the specific subclasses of Datastore. For each data-type there would be a subclass that would contain methods specific to querying and processing that data-type. The two major subclasses included IceStore and TileStore. IceStore handled querying and processing the ice surface, thickness, and bed at any point or in a range such as inside a rectangle. TileStore handled the querying and processing “tiles” of compressed heightmap data, which after uncompressing was useable by the renderer.

The actual data-models followed a similar object-oriented design. DbData was a superclass with fields such as id, row, and column that was present in all datatypes. Subclasses of the DbData such as IceData and SquareTile contained all the fields specific to that datatype.

Using the datastore interface from the renderer is intuitive and consists of instantiating the subclass of the desired data-type, and simply calling one of the methods to retrieve the data. IceStore would query the database, parse the results into IceData, and return IceData to the renderer. Similarly, after querying the database, TileStore would uncompress the heightmap data, create a SquareTile, and return the SquareTile to the renderer.

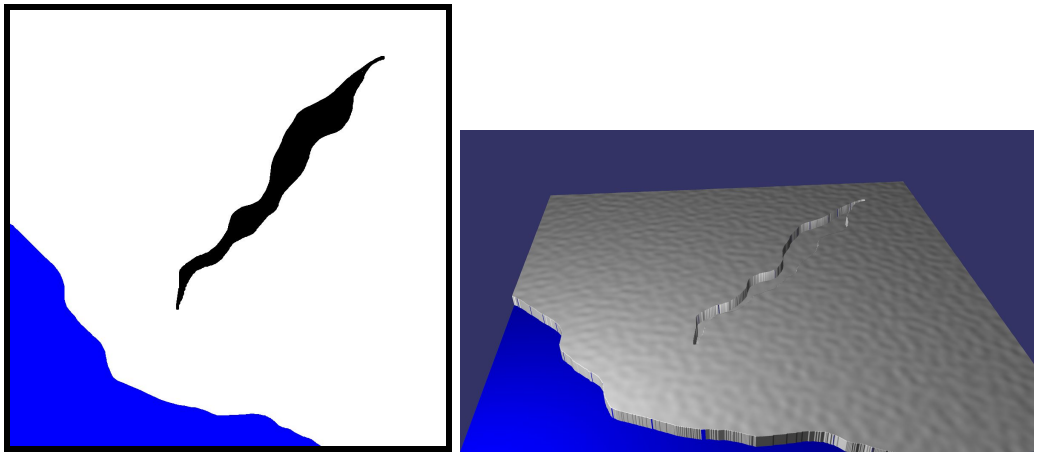
This paradigm is easily extensible since for any new data-type or table in the database, all that is required is to create another subclass of Datastore/DbData, and simply implement the desired methods. The design is also fairly flexible as users can always just use the lower-level connect() and run_query() methods to directly run any custom sql queries they desire without having to change the code.

1.4 Renderer (Visualiser)

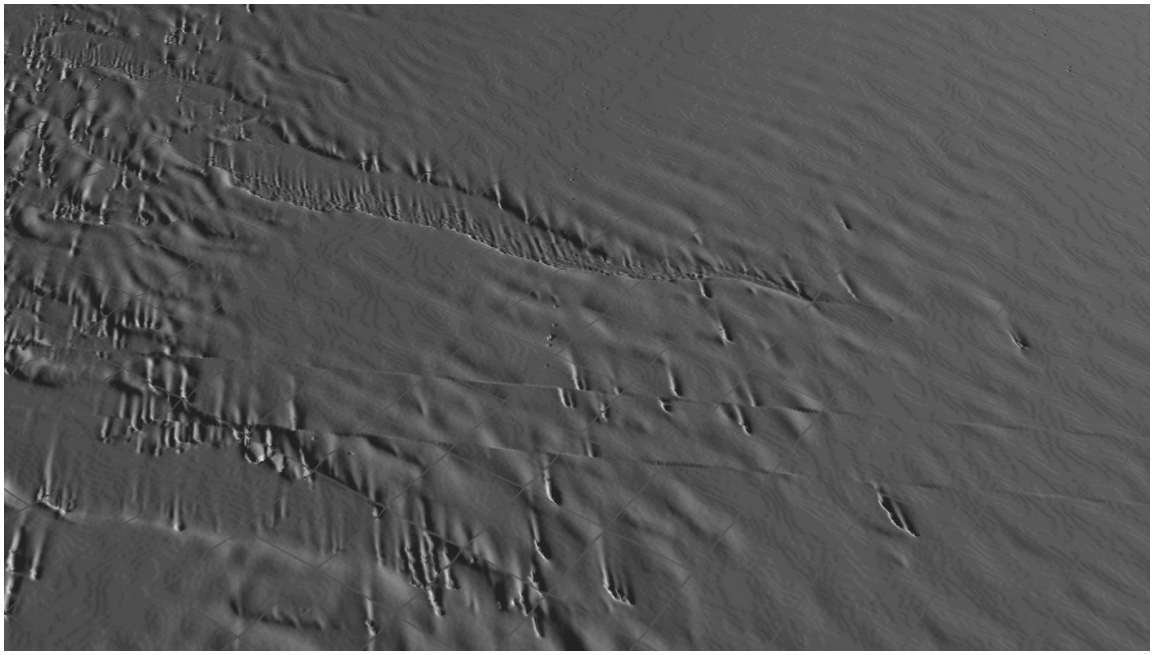
The ice scenery is rendered and navigated using OpenSceneGraph, a C++ graphics library which provides many useful abstractions over OpenGL. OSG facilitates rendering by handling the scene as a “scene graph”, which is a tree like structure on which the leaf nodes are Geometry which can be rendered, and the other nodes are used for grouping, and controlling the materials and shading techniques used. OpenSceneGraph uses this graph structure to efficiently reduce the geometry rendered in the scene, and also provides level of detail modelling. The renderer can access data from the Postgres datastore, and asynchronously load height-map data, divided into tiles, and turn that heightmap data into Geometry which is then rendered by OSG.

A TiledScene class handles the tiles of heightmap data in the scene, dynamically loading these tiles, processing them into renderable geometry and then loading and unloading this geometry as necessary based on where the player is. The tiled scene uses a TerrainUpdater class, which is contained within an AsyncJobHandler (a template class to automatically multithread a process which is divided into work units - jobs), to request updates for individual tiles. Tiles are represented by TerrainTile classes, which implement a means of processing a height map into Geometry for this node, and also handling the material for the node. Different types of tiles handle different parts of the scene, including water and ice.

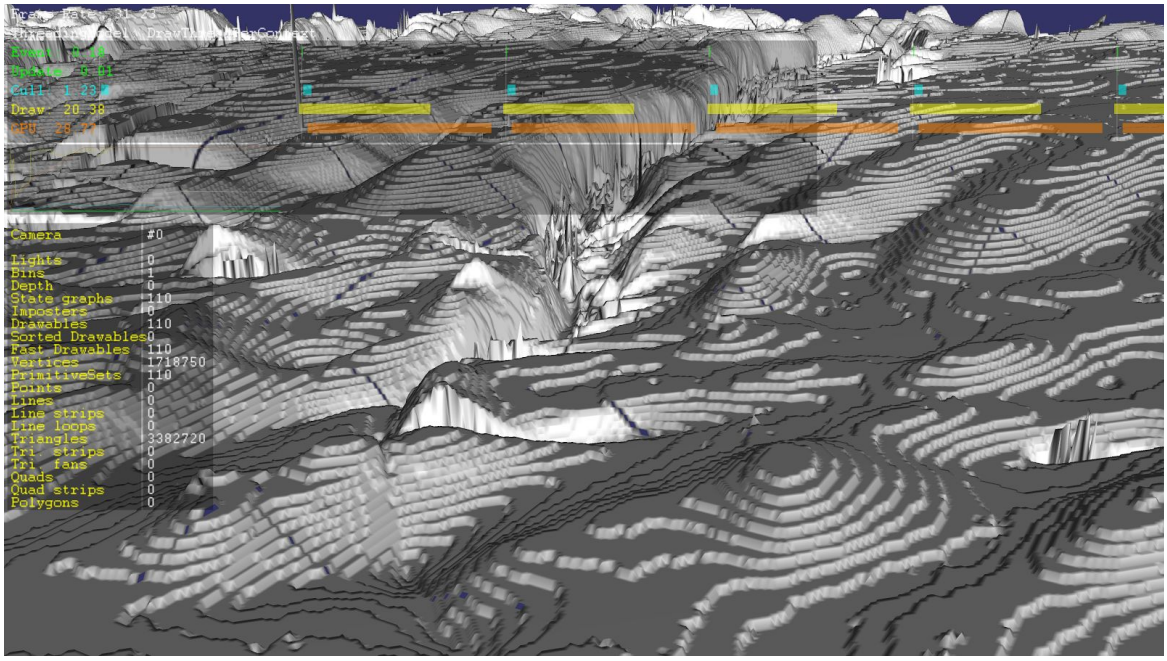
Since we did not have access to any data until a long way into the project, initial testing of the renderer was done by generating simplex noise using the GLM (OpenGL Maths Library) GTC (extensions) simplex noise function. This allowed smooth height maps to be generated to test the rendering capabilities. During this limbo period before we had access to useful data, we attempted to test the renderer by generating test data. One way we did this was by simply drawing out an image in GIMP, where each pixel of white, black or blue represented whether the vertex at that point was ice, crack or water respectively. The below images show the image map used on the left, which was then exported to a C file and loaded by the renderer as seen on the right. This highlighted a problem with height maps which is discussed later in failures.



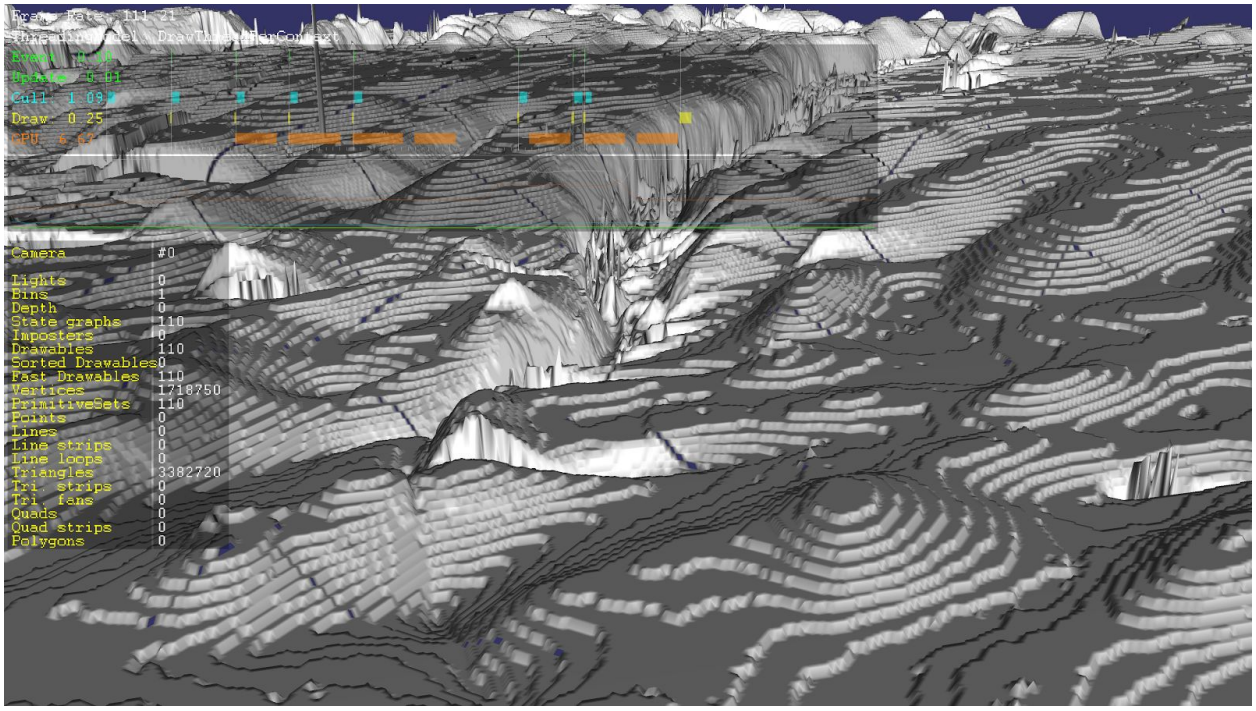
The renderer was able to render the heightmap data provided:



It also performed fairly well, which is the primary benefit of using a fairly bare metal nature of OpenSceneGraph. Run on my local machine, with Intel's integrated HD 4400 graphics, with a render distance of 10 tiles and a reduction factor of 4 on the tiles, the renderer performed as follows (at about 30 frames per second). Note that the heights here are exaggerated in a different way to the above picture, we are currently not sure what the actual heights should be:



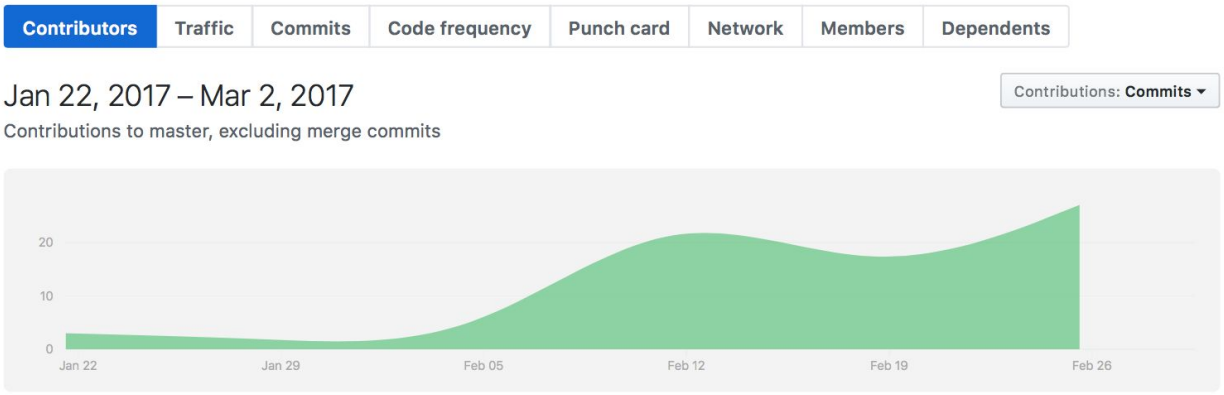
Using instead the onboard NVIDIA GTX 970m, it performed quite a bit better, at about 110 frames per second:



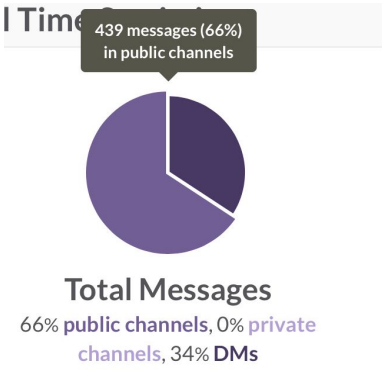
It’s worth noting that the performance indicated here is not entirely accurate as frame draw times spike when I run a screenshot.

1.5 GitHub and Slack

Using GitHub as a tool for team management and synchronisation proved to be extremely beneficial for the development of the project. All members contributed to different stages in the pipeline, as shown in the Contributors graph that can be found on https://github.com/nandck/Bravo/graphs/contributors, and is also displayed below:



Using Git and GitHub as a tool to store, synchronize and display our work makes it very easy for us to know who worked on what part. It also improves accountability, as it is easy to see who contributed to which piece of code so that we can at any point know who is responsible for what. Using GitHub also improves the transparency of the project as it tells the story of how the project went. In our case, the contributions graph shows that the project was quite back-loaded, as most lines of code were contributed in the later stages of the project.



We also used Slack as a tool for communication. This improved accountability of communication within the team: as Slack's design encourages visibility, it was really easy for us to track down who is responsible for doing what.

2 Project Failures

The team had to assemble a short risk assessment in the beginning of the Group Project as our first assignment. We identified risks related to data gathering, management and processing; communication inside the team and with experts; and management of the team overall.

In this section we discuss the shortcomings of the project. Although the challenges Group Bravo has faced in the duration of the group project are closely related to each other, we are exploring them one by one in order to achieve clarity on the issues that we have experienced.

2.1 Scope

The scope of the functionalities that we anticipated to have was quite broad, especially given the time constraints we had as well as the difficulties in communication with experts from the British Antarctic Survey. Extensive work was done to get the basics of the visualisation down — we have a steady pipeline that is easy to modify and maintain, which takes us from GeoTIFF and other data all the way to a convincing 3D visualisation of the Brunt Ice Shelf and the Chasm One in particular. But despite that, we had trouble fulfilling all the stretch goals that we outlined in the Functional Specification/Project Plan in the beginning of our work on the project.

A deliverable that Group Bravo could not ship is the introduction of audio in the visualization. Although this was a stretch goal (a “should” rather than a “must”), it was not implemented due to lack of resources.

In the initial meeting with our client, Matt Polaine of the BAS, the audio capability was established to be helpful for imagining the experience of being near the chasm: although being on the surface of the Brunt Ice Shelf is very noisy as one hears constant ice cracking, descending into the chasm would mean that all ice cracking noises disappear, leaving the only thing that one can hear be the noise of ice that has broken off.

2.2 Data

As we had observed in the risk assessment and the Functional Specification, we faced a number of different challenges regarding access to data. There were some aspects of data collection and application that we thought of would be a challenge for us and that we might be unsuccessful in using.

2.2.1 Images

As it became clear in the first client meeting, drone footage covered a very small area around the crack. Moreover, satellite images are difficult to use as the Ice Shelf moves over time and changes its shape as it is ice and not rock. This means satellite images from some coordinates at one timestamp wouldn’t overlap exactly with satellite images from the same coordinates but

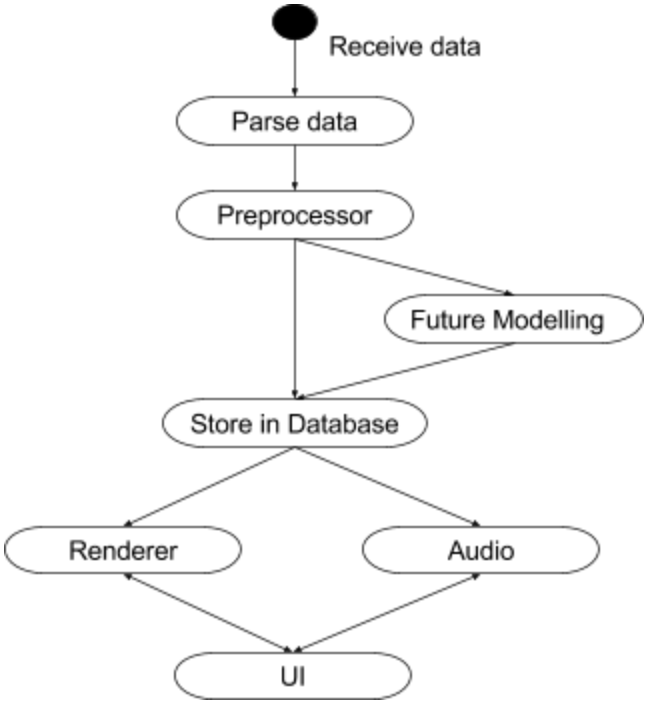
from another timestamp. These factors made drone footage and satellite imagery very difficult to include in the final visualization.

2.2.2 Access

There was also the question of access to data: probably the main challenge the project faced. We never received any drone footages, and it turned out that a lot of the imagery that the British Antarctic Survey can work with is licensed to them through other organizations so that the BAS couldn't give us access to that data. We got access to some LandSat (satellite) images; however, the problems arising from the unusual movements of the ice mass prevented us from adding satellite data to the body of the visualization.

Access to data turned out to be a crucial element of the project, and its lack was the thing that most deeply affected the outcomes. Our pipeline (image below) starts with receiving data, and everything else -- parsing and processing, storing and visualization, comes after. This means that our project is extremely reliant on the data provided to us.

However, accessing data was very slow and difficult. Unfortunately, up until the second client meeting we only had access to an ESRI grid with 1000x1000m resolution from the BedMap2 project [11].



Although the data was of good quality and set the project up for a well-designed pipeline, it was quite old (collected between 2007-2009) and so Chasm One was not visible in the data -- and even if it was, a one-kilometre grid is way too big for the crack to be visualized properly. At the second client meeting we received a big, recent GeoTIFF file with a much better resolution that gave us the data to visualize the crack. We also received guidance about access to Satellite images but could not incorporate those into the final product.

Only in the last few days we got access to some GPR (Ground-Penetrating Radar) data, but no LIDAR or other types of data that we anticipated.

2.3 Renderer (Visualiser)

2.3.1 Programming language and library choice

There was some uncertainty initially about what libraries and tools should be used for creating the visualisation. The two main suggestions initially were to use C++, or to use Unity. Since Unity is difficult to use in collaboration, and none of the team had experience with it, it seemed like it'd be preferable to try and make progress with C++. In hindsight, the use of C++ and OpenSceneGraph was a poor decision, as although it was easy to handle version control, and the end product was efficient, it suffered some major pitfalls: collaboration to the C++ renderer was severely impeded by the issue of building the project on different machines. While the dependencies were cross platform compatible, and the build did work on OSX and Linux machines that were run by the members of the group, it was cumbersome to set up, and wasted a lot of valuable development time. Another pitfall of C++ was that it made many tasks overly confusing due to the necessity to implement memory and locking management. And in fact even after taking care (i.e. wasting time) writing memory safe code, and using OpenSceneGraphs built in reference pointers, an initial execution of valgrind memory analyser reveals a number of issues:

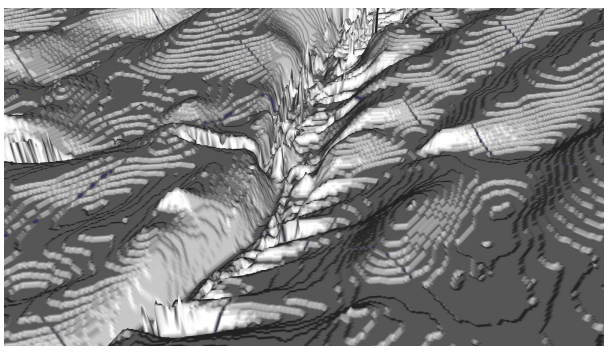
Memcheck — Memory Analyzer Tool finished, 1406 issues were found.

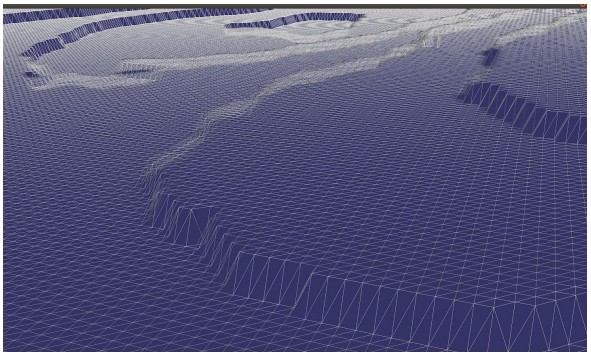
Issue	Location
Invalid read of size 8	AsyncTerrainUpdater.cpp:41
Invalid read of size 8	AsyncTerrainUpdater.cpp:41
Address 0xf75f0d0 is 0 bytes inside a block of size 40 free'd	AsyncTerrainUpdater.cpp:41
Block was alloc'd at	AsyncTerrainUpdater.cpp:41
136 bytes in 1 blocks are possibly lost in loss record 2,324 of 3,367	TerrainTile.h:36

Given more time, these issues could be debugged, but it is arguable a considerable failure of the project, given such short time to undertake it, if we spend time on memory analysis. In hindsight it would likely have been wiser to spend significantly more time researching other visualisation libraries, possibly on a platform such as the Java JVM, which would be easier to maintain and distribute.

2.3.2 Unanticipated issues

The biggest unanticipated issue with the renderer was that using height maps for rendering the surface of the ice resulting in a rather unaesthetically pleasing blocky appearance.





As can be seen in the images above, the “stepping” effect that the height map results in due to the discrete values of heights. This can be partly fixed by smoothing the surface (interpolating points), which is currently a work in progress. This is less a failure of the project as it is a lack of time, as this problem could be addressed given more time. Although it could be considered a failure that we did not pick up on this earlier on in the project, which is down to us not effectively anticipating the data before it arrived.

2.4 Communication

The issue of data access is closely related to the issue of communication with the experts from the British Antarctic Survey. Although the internal communication of the team was more than reasonable, it was really difficult to establish connection with the experts that were responsible for different data that the BAS works with.

In the beginning of the project development, the most promising part of the project seemed to be related to the newly-developed Halloween Crack, also known as Chasm Two. However, this crack had been growing really fast during the last few months. While exciting, this meant that there was little to no data available for us to work with.

The unpredictable growth of both Chasm One and Two meant that the time of our Group Project was really exciting for scientists to be doing fieldwork and collecting data at the Brunt Ice Shelf. However, while in the Antarctic, scientists have generally very limited opportunity for communication with the outside world. This meant that scientists would be generally quite unresponsive to our emails, or respond slower than we initially anticipated in our Risk Assessment.

Moreover, at the first meeting with our client it became clear that the crack growth predictions for Chasm One that we would have visualized would not be based purely on data but instead we would have to be in touch with some of the leading experts from the British Antarctic Survey. Unfortunately, due to the incompatibility of our schedule with BAS’ own scientists’, it was quite difficult to receive guidance on the prediction side of the project. However, we designed our pipeline with a time-based prediction in mind so that it would be quite easy to provide suitable inputs so that we visualize the state of the Chasm One crack as well as the Halloween Crack at different time points.

3 Team members' activity

3.1 Nand

I primarily worked on the C++ interface to Datastore, which handles all requests for connecting, sending, receiving, and querying data from the database. This was challenging given that we had large amounts of data in different formats, and needed a way to make spatio-temporal queries to retrieve the data to be used by the renderer. My first task was to research a connector that allowed us to interface with the database through C++, and I found libpqxx to be the ideal solution for such a task. I then designed and implemented the Datastore api, which allowed for connecting and querying from the Datastore in a simple and intuitive manner. This involved implementing a number of major methods including connect(), run_query(), and processDbRow(). I also created a DbData class that would have all the fields representing data from the database, and wrote the parsing functions to convert the raw data that libpqxx returned to a DbData object.

My api design seemed to work for a while, until we received other forms of data and had to have multiple different tables with different schema's in the database. I realized that a refactoring was necessary, and changed the design of Datastore to take advantage of object-oriented paradigms. Basically, both the Datastore class and DbData class would now be superclasses that contained general methods for all database operations/data. Next, for each table we would now have a subclass of DbData to encapsulate data specifically from that table. Similarly, for each table we would also have a subclass of Datastore that contained useful methods specifically for querying and processing that particular table. Examples of such subclasses were IceData/IceStore, which allowed for querying the ice surface, thickness, etc. and SquareTile/TileStore, which allowed for querying for tiles of height-map data. These changes allowed us to add and process additional data-types while keeping the codebase clean and easily extendable.

Finally, I spent a bit of time on the side writing an interpolation method to create a parametric spline in order to interpolate a set of spatio-temporal points. This would be useful for creating a smooth time-based animation. However, since we did not have enough time-based data, the method is not currently in use.

I thought my work was challenging since it involved essentially being the middle-man between two different parts of the pipeline (the data and the renderer), forcing me to be up-to-date for any demands or changes on either side. I learnt many new technologies and concepts through the project including PostgreSql, libpqxx, handling GIS data, writing spatio-temporal queries, building an api, and improving my C++ ability.

3.2 Andreea

The two main things I worked on were putting the data resulted from the preprocessing of a ESRI grid file from the BedMap2 project into a database and smoothing the edges for the visualizer based on data preprocessed from a satellite image.

My first task involved using PostgreSQL with the PostGIS extension. We chose them because they provide spatial and geographic tools which are useful for the nature of our project. I created a database and a table that was populated with data from the .csv file which was the output from preprocessing the bedmap. In addition to this, I created a column that kept records of type geometry made of (x,y) coordinates and a primary key id which made it faster for the renderer to search for records. I added tests, a sample query that returned the points in a given rectangle and documentation in which I wrote about the steps I followed and a list of problems I encountered and what solutions I found.

For the smoothing, I wrote two scripts in python: one that decompressed all the .snappy files that resulted from the preprocessing of the satellite image and one that iterated over all consecutive values in the decompressed files. For each iteration, it checked the difference between the heights - if it was too small, it meant that the data is too dense for a flat area and points can be deleted, while if it is too big, it meant that data is too sparse and points need to be added to turn a slope in stairs into a more realistic figure.

I believe I learnt a lot during this project. Apart from PostgreSQL and Python, which are the tools I used the most and learned the most about, I also improved my knowledge of git, Snappy and Matlab.

3.3 Julia

Although my primary and secondary tasks were specified as User Interface and Integration testing, I found myself involved in a wider range of activities in the project. From the beginning I was helping the team with certain organisational challenges. This involved creating Slack channels and monitoring activity, as the deadlines for deliverables were approaching; creating Google Docs and Sheets, minuting group and client meetings.

On the technical side of things, one of my responsibilities was examining the prospects of using MATLAB's mesh and meshgrid commands with the Bedmap2 data. Unfortunately, even though it proved possible to create a 2D visualisation as a heat-map, turning it into a 3D model did not work out. This was largely due to the resolution of the data (points 1 km apart) and the breadth of coverage (all of the Antarctica and adjacent seas). Instead, I resolved to looking into the C++ code, as I had no previous experience in programming in this language. OpenSceneGraph provided a lot of useful libraries and it was interesting use some of the shading techniques that we learnt in Computer Graphics course.

Another job was using Prewitt, Sobel and Roberts filters to detect edges of the chasm on the satellite image. This was done before the higher resolution data was acquired and so was not used afterwards. Coming back to the UI and testing tasks, I decided to look further into

documentation of the project where Doxygen tool proved very useful. This is an ongoing process and we hope to have a well-documented project before submission.

UI features were not a pressing issue at the start, because the visualiser provides a camera view that can be controlled with a keyboard. However, the plan now is to integrate the CameraManipulator class to have a drone-like control with momentum and ease of movement instead of mouse clicks for zooming in and arrows for moving around. The biggest problem that I have been facing is the OS features of my machine that sometimes prevent me from using certain tools or require a lot of modification.

Overall, I learned something new in every aspect of the project and reinforced my knowledge of graphics algorithms, C++ structures and Software Engineering practices. I explored how databases and servers before the start of the corresponding course, reminded myself of Python and educated myself on Git.

3.4 Matt

My initial aim was to work on preprocessing the large amounts of heterogeneous data about the chasm that we were expecting from BAS. Delays in receiving data mean that I had to be more proactive in searching for tasks to work on. As a result, I ended up working on an experimental random scene generator, and later on handling GeoTIFF data.

I was primarily working to ensure that the data necessary for the visualisation was in a form that could reliably be loaded into the visualiser. This meant investigating ways of reducing memory usage by dividing heightmap data into tiles, and reducing disk space using compression, while aiming to maintaining reasonable load times.

The project has given me the opportunity to gain experience working with OpenGL in C++, dealing with concurrency in C++, as well as working with geospatial data in Python and PostgreSQL.

3.5 James

I assumed the role of writing the Renderer/Visualiser, while there was much uncertainty around what platform and libraries we would use for visualisation initially, I began by prototyping the visualisation using C++ and OSG (OpenSceneGraph). This ended up being used for the duration of the project, during which I worked on improving the visualisation, extending it to work with the data that we were eventually provided.

Primarily the problems I solved were, turning data in the form of height maps provided from the datastore into Geometry which is then rendered with OSG. The renderer also had to handle dynamically loading this data, and dynamically loading a suitable level of detail for the surface.

During the course of the project, I’ve gained more experience in writing and maintaining maintainable modern C++ code. While in the past my experience with C++ has been trying to write highly efficient C++, I have used this project as an opportunity to write C++ making

better use of standard templates, in more readable and efficient to write ways, and also implementing my own templates to facilitate such as parallelising operations.

Due to the very late arrival of useable data, much of the visualiser work involved generating test data, such as the test crack data shown in the renderer section above.

3.6 Val

I had two main roles during the development of this Group Project. I was the Project Manager of the team, and also worked primarily on the parsing and pre-processing of data.

As a project manager, my primary responsibility was to serve as the connection between Group Bravo, the Project Organisers, and the British Antarctic Survey. In particular, I was in charge of email communication: sending emails to the Project Organizers and the client, as well as some of the scientists from the BAS; forwarding important emails, especially those coming from the BAS as responses to our inquiries for data. Moreover, I was involved with formatting and editing the Functional Specification, Progress Report and the last Project Report: all documents that Group Bravo is submitting. Lastly, I was responsible for writing out the sections that relate to team management in these documents.

In terms of development of the actual project, my work was concentrated in the beginning of the pipeline. In particular, along with Matthew I was responsible for parsing and pre-processing of the data that came into the pipeline. I wrote the scripts that transform the ESRI grid data from BedMap2 into elevation data that can be directly imported into the datastore for the visualization to use.

However, my role in the team was also quite involved around research of the public data that the scientists from the British Antarctic Survey pointed us to. An example is the public data from EarthExplorer [12], a web-service ran by the USGS, the U.S. Geological Survey. Our team was pointed to this resource by the BAS scientists, who thought we could use some of the LandSat images in the EarthExplorer datasets. However, Andreea, Julia and I had to look through the variety of datasets available in order to figure out which datasets are useful to the project.

Overall, working on this group project has been fun. I am very glad that I could learn about project management in the context of such a motivated team that had extremely diverse expertise. Moreover, I was able to apply concepts from the industry setting in this project.

References

1. http://www.openscenegraph.org/
2. https://www.qt.io/ide/
3. https://www.postgresql.org/
4. https://github.com/google/snappy
5. http://pqxx.org/development/libpqxx
6. https://github.com/nandck/Bravo/blob/master/Preprocessing/compressionbenchmarks.py
7. https://github.com/python/cpython/blob/master/Lib/lzma.py
8. https://github.com/python/cpython/blob/master/Lib/bz2.py
9. https://github.com/python/cpython/blob/master/Lib/gzip.py
10. https://github.com/andrix/python-snappy
11. Fretwell, P., et al. "Bedmap2: improved ice bed, surface and thickness datasets for Antarctica." The Cryosphere 7.1 (2013).
12. https://earthexplorer.usgs.gov