Contents

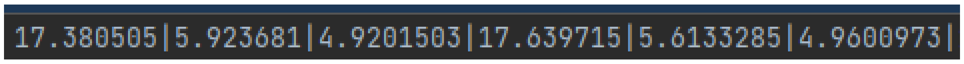
What is a point-cloud?

Every point has position within space. If there are many points that are included together to logically form one unit, then they become a point cloud. Since a point by itself cannot be seen as it is infinitely small, it can be represented by a geometrical representation such as a cube. If this is done for every point, then a visual representation of the cloud can be created.

How to use the Point Cloud Visualizer

Format of a Point Cloud File

A point is composed of a x, y and z component. These are separated by the "|" character all on one line. This is repeated for every point. For example:



```
17.380505|5.923681|4.9201503|17.639715|5.6133285|4.9600973|
```

The above shows two points. Note that the z component indicates height, as is common in a mathematical coordinate system. The y and z coordinates are swapped internally in the visualizer as the y component represents height in a graphical setting.

Loading a Point Cloud

Static Point Cloud

These types of clouds do not change. They are loaded once and then never change. To load a static point cloud , specify a location using the "-i" command to a point cloud file when launching the program. For example:

```
./PointCloudVisualizer -i someLocation\pointCloudFile.txt
```

Dynamic Point Cloud

These types of clouds are updated after a certain period of time. These are updated using file IPC. There is an external program that generates new point cloud data and writes it to a file. Then the visualizer reads that file to get the new cloud data and renders it. This data file used for communication is specified with the "-d" option. To synchronize the communication (such that the data file is not being updated while the visualizer is reading it), a mutex file is used. A mutex file is specified using the "-m" option. When the external cloud program is generating new data, it will only do so if the mutex file contains specific data; when this is the case, the visualizer will not attempt to read the data file. When the external program is done generating new data, it writes different content to the mutex file. At this point the visualizer will start reading the data file, and the external program cannot update the data file until the visualizer has updated the mutex file with its own content.

When the external program is done generating new cloud data, it must write “taken” to the mutex file (only that- the file should be deleted of all previous content before writing). When the visualizer is done reading the data file, it will write “clear” (and only that- the mutex file will be cleared of anything else before writing). Only at this point, when the mutex file contains “clear”, can the external data start populating the data file. These mutex files can be checked in a simple loop; opening the file, check if its contents are exactly as expected. If not, then close the file and wait some time before checking again.

To speed things up, several data file and mutex file pairs can be provided. This allows the external program to write updated data to a different data file while the visualizer is reading a different data file. This is specified using several “-m” and “-d”; there must be an equal number of mutex and data files.

An example dynamic point cloud:

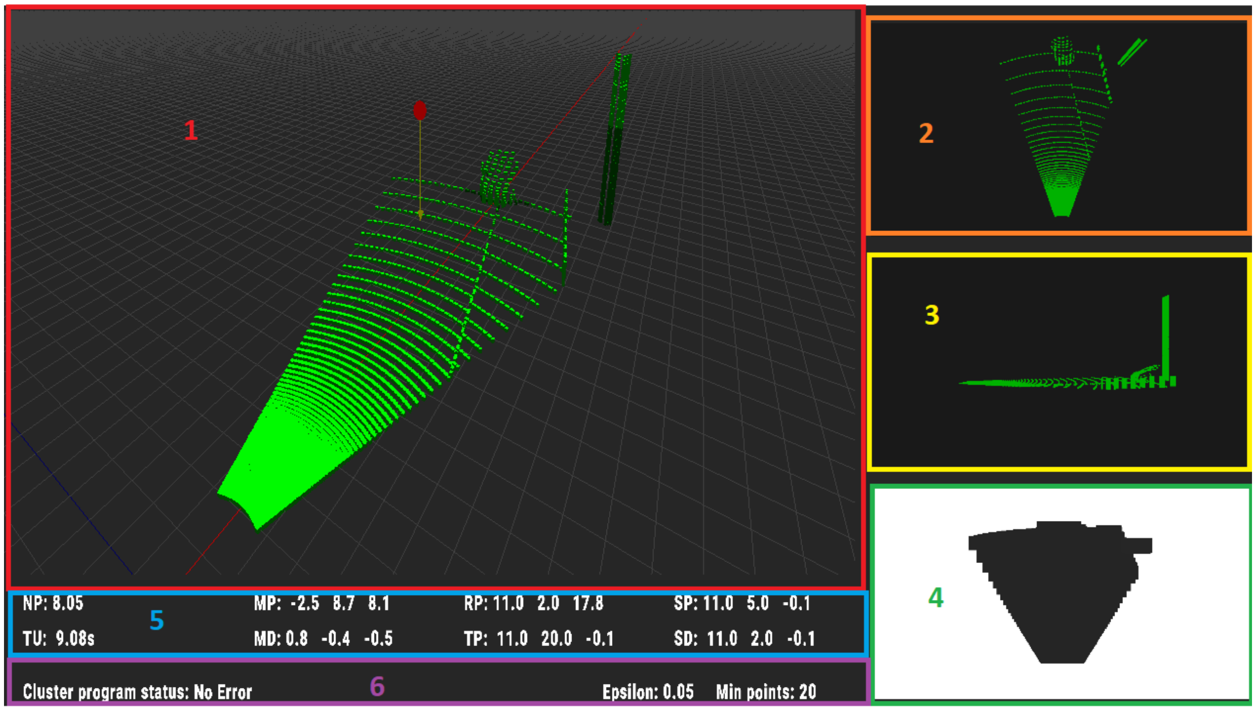
```
./PointCloudVisualizer -d someLocation\data1.txt -d someLocation\data2.txt
-m someLocation\mutex1.txt -m someLocation/mutex2.txt
```

The data and mutex file pairs are specified by the order given to the program; for example, since data1.txt and mutex1.txt are the first data and mutex file given to the program, they form a data-mutex file pair. The pairs to be read from are chosen using round-robin.

When the point cloud is initially loaded, the main camera, sun and side views will be centred around the loaded point cloud (or the first point cloud loaded if a dynamic point cloud is loaded). These objects can be moved using the instructions noted later in this guide.

The rate at which the visualizer checks the currently chosen mutex file can be specified using the “-s” option. This is the amount of time to wait if the mutex file indicates the data is still write-ready in milliseconds before checking again.

Parts of the Visualizer



```
NP: 8.05          MP:  -2.5   8.7   8.1    RP: 11.0   2.0   17.8    SP: 11.0   5.0   -0.1
TU:  9.08s        MD: 0.8   -0.4   -0.5    TP:  11.0   20.0   -0.1   SD:  11.0   2.0   -0.1
Cluster program status: No Error                          Epsilon: 0.05    Min points: 20
```

The parts of the UI are as follows:

1. The window into the scene. This is where the point cloud is rendered. The red ball indicates the sun and the arrow extending from it is the position it is looking at
2. The top view of the point cloud
3. The right view of the point cloud
4. The shadow map of the point cloud
5. Point cloud information and camera information
6. Cluster information

Main Scene

The standard WASD keys are used to move around the scene. To rotate the camera, clock and hold the middle button (scroll wheel) and move the mouse around.

Top View and Right View

Click on either of these views to make them active. This is indicated by a green border around the selected view. Then to move the camera, use the WASD keys. Additionally, to move in the camera's "up" direction, use the Q key to move in that negative direction and "E" to move in the positive direction. Click the view again or anywhere else to exit the selected view state. The right camera always points in the negative x-direction; the top-view in the negative y-direction.

Shadow Map View

Clicking on the shadow view once will result in a green border of the view. In this state the position of the sun can be moved using the same controls and the top and right view. Clicking the view again will result in a blue border around the view. In this state the position that the sun is looking at can be changed using the same keys as the top and right view (WASDQE keys). Click the view again or anywhere else to exit the selected view state.

Point Cloud and Camera Information

The data being represented by the text is as follows:

- NP: Number of points in the point cloud in thousands
- TU: Time since the cloud was lasted updated. Has no meaning for static point clouds
- MP: Position of the main camera (that of the scene)
- MD: The direction the main camera is looking in
- RP: The position of the right view camera. Direction is not given as it is always looking in one direction
- TP: The position of the top view camera
- SP: The position of the sun
- SD: The position the sun is looking at

Cluster Information

Contains any error messages describing errors while running the cluster program. Also displays the epsilon and the minimum number of points required to form a cluster

Running Cluster Detection

This functionality is provided through an external program, included with the visualizer. It runs the DBSCAN algorithm. To execute it on the current point cloud data, click the “c” button. To change the epsilon value, use the “Z” and “X” keys. To change the minimum point cloud number, use the “V” and “B” keys.

The cluster detection can only be run on static or paused dynamic point cloud.

Pausing Dynamic Point Clouds

To run a cluster detection on a dynamic point cloud, it must be paused from updating. This can be achieved using the “P” key. Unpausing the cloud so that it can be updated again can be done by clicking the “P” key again.

Moving the Point Cloud

The entire point cloud can be moved by:

- X-dimension: F2 for negative, F3 for positive
- Z-dimension: F3 for negative, F4 for positive
- Y-dimension: F5 for negative, F6 for positive

Flipping the Point Cloud Vertically

Depending on the positions of the cloud points, the point cloud may appear to be flipped upside down. This can be fixed by clicking the “F7” button, which will flip the cloud across the x-z plane (or x-y plane in non-graphical setting) axis. When this happens, the keys for moving the point cloud vertically (F5 and F6) are reversed.

Input Summary

- WASDQE keys:
    - If in main scene: Moves the main scene camera
    - If in top or right side view (click view to enter and exit): Moves selected view camera
    - Shadow map view:
        - Sun mode (view is clicked once): Move the position of the sun
        - Sun look at mode (view is clicked twice): Move where sun is looking at

- Mouse movement + Middle / Scroll wheel down
    - If in the main scene, will rotate the camera

- ZX keys:
    - Changes the epsilon value for the DBSCAN algorithm

- VB keys:
    - Changes the minimum number of points required for a cluster using the DBSCAN algorithm

- C key:
    - Runs the DBSCAN clustering algorithm using the provided epsilon and minimum number of points for cluster parameters

- P:
    - Pauses updates to the point cloud if a dynamic point cloud is loaded

- F1, F2
    - Moves the entire point cloud around x-axis

- F3, F4
    - Moves the entire point cloud around z-axis

- F5, F6
    - Moves the entire point cloud around y-axis
- F7
    - Flips the point cloud vertically (depending on the coordinates of the points in the cloud the cloud may appear upside-down)
- Escape
    - Closes the program

Reason for File IPC (for cloud updates and cluster detection)

At the time of writing the program, there were no widely used implementations of DBSCAN in the Rust ecosystem. Implementing a simple version of the algorithm proved to be too slow. It was then decided to use a library of a different language, in this case C++. Creating a shared or static library that contained the detection algorithm resulted in linking errors. Due to time limitations, it was decided to have the algorithm run as a separate program.

Ideally sockets would have been used for the IPC that is required when running two programs. However, when using sockets with C++ and having them interact with a Rust program, communication was not consistent- message content appeared slightly different than expected. This is likely due to errors in the way the sockets were used. This could be figured out, but as mentioned before, time limitations meant that the simplest option had to be used. In this case that was file IPC.

The same reason is why File IPC is used to updating the point cloud. A sample Python program that sent data over sockets to the visualizer stalled unexpectedly. This could have been solved given enough time, but it was known File IPC worked so that was chosen instead.

It should be noted that since cluster detection is not run often, performance hits of file IPC is not noticeable. Same with cloud updates- if a cloud is to be analyzed, then it will be paused so that any points of interest are not lost through updates.

Notes for Running Program From the Project Source

If compiling the project, indicating that the program will be launched from the project root, then the environment flag “DevelopmentFlag” needs to be set. This ensures that the folders containing required files can be found (relative to the project root).

Citation

The implementation of the DBSCAN clustering algorithm is provided by Open3D:

```
@article{Zhou2018,
    author    = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
    title     = {{Open3D}: {A} Modern Library for {3D} Data Processing},
    journal   = {arXiv:1801.09847},
    year      = {2018},
}
```