Point Categorization

Program Specification

Inputs:

- Input .pcd filename
- Output .pcd filename
- Runtime keypresses: change brush size, undo, redo, save point cloud, set current point category, categorize points (PCL Point Picking)

Processes:

- Read input .pcd file
- Initialize colour data for input .pcd file (if a point cloud has no colour data associated with it, the rgb values will automatically be set to 0,0,0 when the file is read. The program will therefore automatically initialize any r,g,b=0,0,0 point to the colour corresponding to the ‘Unassigned’ category.
- Set brush size (this must take into account minimum and maximum brush sizes)
- Get points near target (this will generally be used with the current brush size to get all of the points near a picked point)
- Categorize Points
- Undo/Redo point categorization action

Outputs:

- Output .pcd file (using output filename from **Inputs**)
- Point cloud state with RGB data (PCLVisualizer)
- Program state and logging

Design Choices and Rationale

OOP

The point categorization program must manage different variables, such as brush size and interaction state. Additionally, the PCLVisualizer subsytem must be created, updated and managed. It is therefore much easier and cleaner to implement the point categorization program with an OOP design as the internal variables of the program can be encapsulated. The callback functions registered with the PCLVisualizer can also be encapsulated and all internal management is handled by the object.

Point Categories Header

Since different modules follow the same definitions for different point categories and their corresponding colours, defining these elements under the common source folder and namespace allows other modules to access the same definitions without rewriting them. Additionally, any changes to point categories or their colours can be made quickly across all modules just by changing the definitions in “point_categories.hpp” and “point_categories.cpp”.

DebugOut

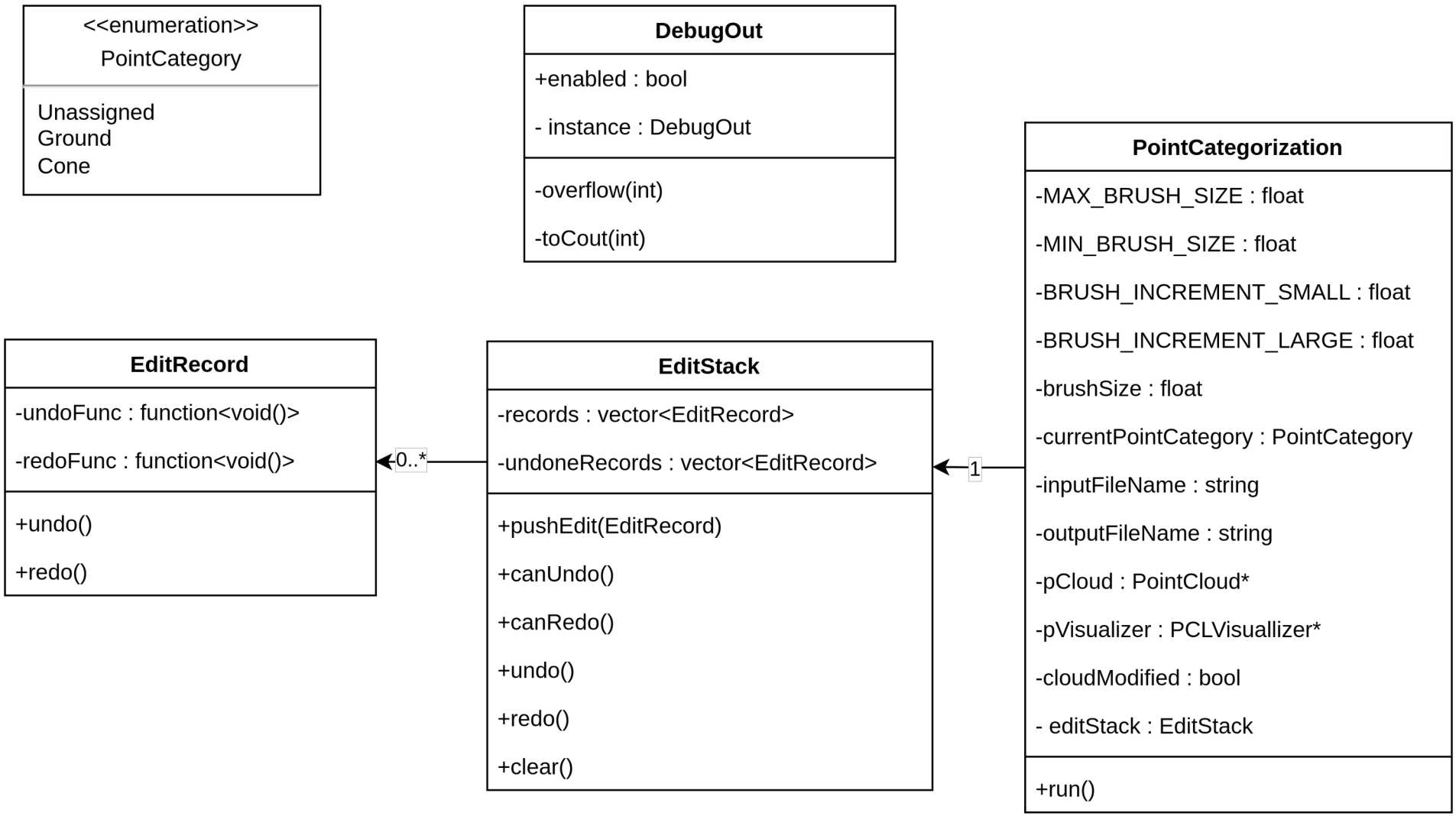
The DebugOut class provides an output stream that can be easily enabled/disabled. This was done in order to provide a more flexible debug logging during development. While standard output is an option, it can become cumbersome over time and removing debug logging would require digging through the code and removing and lines in which we write to stdout. With the debug stream, we can very easily control the verbosity of individual modules just by changing a single variable at the start of the program. Additionally, the debug output stream is easy to modify. The stream can be changed so that it outputs its data to a file, or even across a socket.

Point Clustering – Naive Implementation

In order to categorize points, all points near where the user clicked (using the brush size as the maximum distance) are selected. This could be done quite efficiently with the correct data structures and algorithms. However, for the purposes of this program, a naive approach in which the entire point cloud is traversed is suitable. The naive algorithm does not create a performance issue as it runs only when the user selects a point, which can only be done so frequently. Overall, it is the simplest implementation method and meets the requirements of the program.

Class Overview

Header	Namespace	Name	Description
point_categorization.hpp	point_categorization	PointCategorization	The program class. This class encapsulates all of the core functions of the program.
edit_record.hpp	point_categorization	EditRecord	Represents an undoable and redoable action. All edit record instances must define specific undo/redo functionality.
edit_stack.hpp	point_categorization	EditStack	A stack-like container that manages EditRecord objects. An EditStack provides functions to undo/redo edit records using the most common undo/redo convention. (See **Rationale**)
debug_out.hpp	common	DebugOut	A singleton output stream that can be enabled/disabled. When enabled, it directs its input to std::cout.


<<enumeration>>
PointCategory
Unassigned
Ground
Cone
DebugOut
+enabled : bool
- instance : DebugOut
-overflow(int)
-toCout(int)
PointCategorization
-MAX_BRUSH_SIZE : float
-MIN_BRUSH_SIZE : float
-BRUSH_INCREMENT_SMALL : float
-BRUSH_INCREMENT_LARGE : float
-brushSize : float
-currentPointCategory : PointCategory
-inputFileName : string
-outputFileName : string
-pCloud : PointCloud*
-pVisualizer : PCLVisuallizer*
-cloudModified : bool
- editStack : EditStack
+run()
EditRecord
-undoFunc : function<void()>
-redoFunc : function<void()>
+undo()
+redo()
0..*
EditStack
-records : vector<EditRecord>
-undoneRecords : vector<EditRecord>
+pushEdit(EditRecord)
+canUndo()
+canRedo()
+undo()
+redo()
+clear()
1