CS475: COMPUTER GRAPHICS
ASSIGNMENT 1: MYDRAW

Kumar Saurav [160050057], Yash Shah [160050002]

Abstract

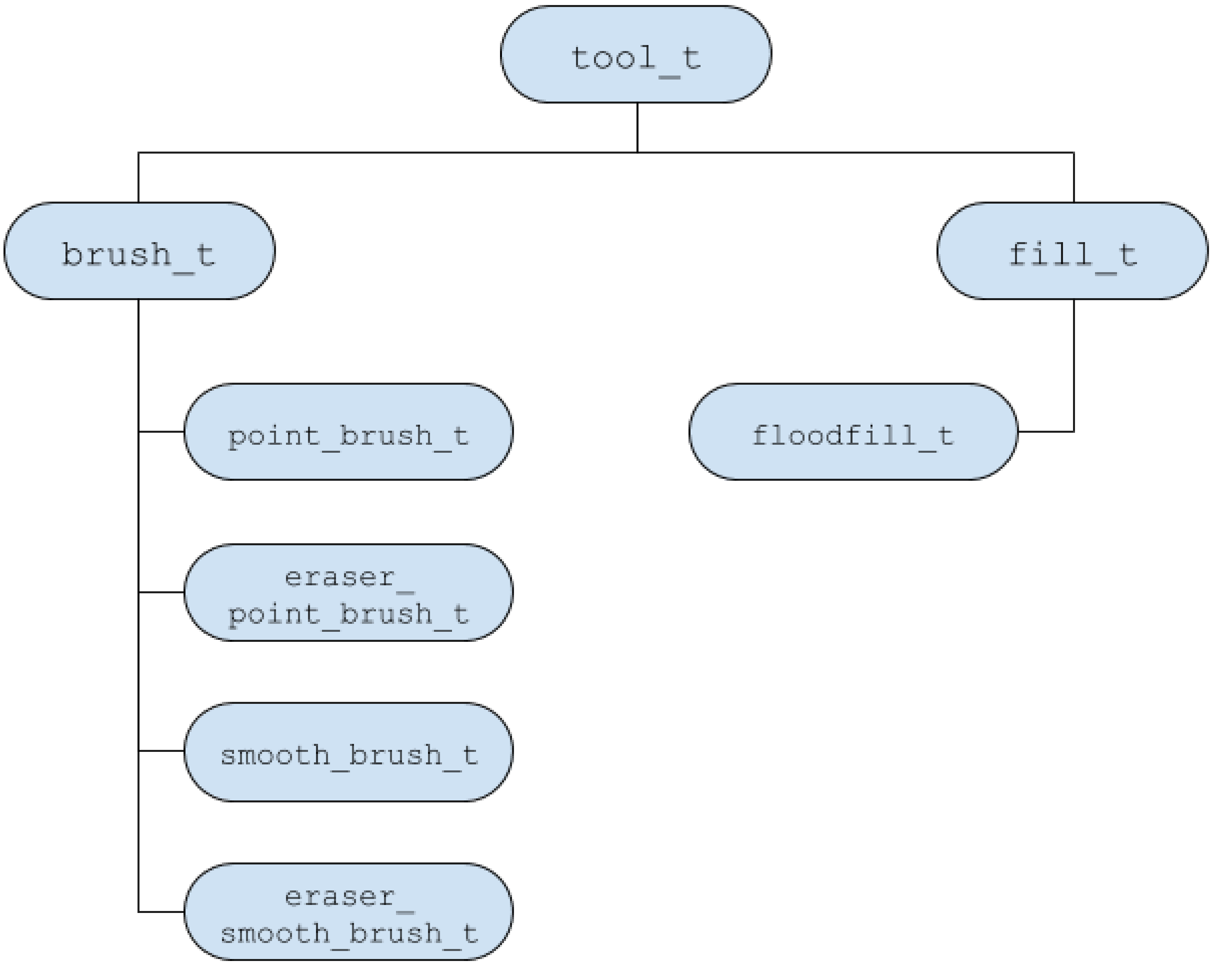
In this assignment we implemented a simple drawing utility, aptly named 'mydraw', in C++ using OpenGL by coding up different rasterization modules. In this report, we describe in sufficient detail what changes we brought about in the starter code, the class hierarchical model we followed and the functionalities we successfully implemented as a part of this assignment.

Contents

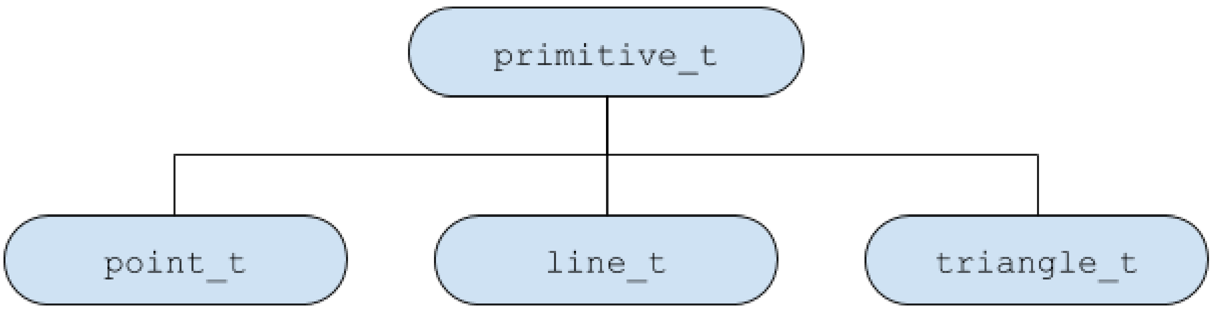
I. Keyboard Shortcuts

Key	Action
F/f	Select the fill tool (floodfill_t)
B/b	Select the brush tool (on repeated pressing in `point` primitive mode, it toggles between point brush and smooth brush)
E/e	Select the eraser tool (on repeated pressing in `point` primitive mode, it toggles between point eraser and smooth eraser)
C/c	Select the foreground color (used by fill and brush tools) by entering the RGBA values from terminal (range 0-1)
D/d	Select the background color (used by eraser tool) by entering the RGBA values from terminal (range 0-1)
+	Increase the size of the active tool (brush or eraser)
-	Decrease the size of the active tool (minimum size is 1)
S/s	Save the current drawing as "default.tga"
P/p	Cycle through the point, line and triangle primitive modes
Esc	Close the drawing window

II. Class Hierarchy



(a)



(b)

Fig:
(a) Tool class hierarchy
(b) Primitive class hierarchy

III. Class methods and variables

Each tool is derived from the abstract `tool_t` class, which defines the basic properties of each tool which include the following:

a. `size` - protected member which stores the size of the tool
b. `tool_type` - protected member which describes the type of the tool. This is useful for differentiating between its child classes

The member functions of each tool are:

a. `action()` - defines the action of the tool on a callback event such as mouse click or drag
b. `release()` - defines the action of the tool on a callback event such as mouse release
c. `get_tool_type()` - returns the type of child tool class

Other than these functions, there are `size` functions which manipulate the size of the tools.

The primitives on the other hand are derived from the abstract `primitive_t` class which has a concrete `get_primitive_mode()` member function, which returns the primitive mode. There are two other virtual functions in the abstract base class - `set_point()` and `get_point()`. Each of the derived classes are supposed to give their own implementations of these functions. The aim of this function is to provide and store the points required for drawing in that particular primitive mode. For example, `set_point()` in the triangle primitive mode stores 3 points in a queue fashion, while the line primitive mode stores 2 points.

The final major modification was to add an `active_tool` of the class `tool_t` in the `draw_context_t` class. This allows us to set the active tool to any of the `brush_t` or `fill_t` type and exploiting polymorphism, use the function `action()` in a callback event to perform the required actions.

IV. Integration with `mydraw`

The program initialises with the point primitive mode and the brush tool as the active tool. A callback is set to the mouse click, which calls `action()` for the active tool on mouse press. When the user changes to the smooth brush or eraser mode, a callback is set on the cursor position in which the `action()` function is called for the tool. On changing back to the point brush or eraser, that particular callback is removed. On changing to another primitive mode, the brush and erasers are reset to their respective point type tools.
Inside the `action()` function for the point brushes (i.e., brush and eraser), the brush checks the current primitive mode and draws the corresponding figure. The smooth brush and eraser work only in the point primitive mode. When the primitive mode is

changed, the current brush and eraser in the canvas are reset to their point counterparts.

V. Functionalities implemented

A. *Variable size brush and eraser* - We extended the `action()` function (which was the `stroke()` function in the starter code) for each tool so that it colored all pixels within a circle of radius r (where r = tool size). This was done by iterating over all pixels lying in a square-neighborhood of size `2*r` around the click position (x_0, y_0), and setting all pixels which satisfied the equation $x^2 + y^2 <= r^2$, where x and y are pixel offsets relative to (x_0, y_0).

B. *Smooth brush implementation with variable size* - We implemented a click-and-drag responsive smooth brush/eraser by using a clever combination of boolean flags and attaching/detaching of a `CursorPosCallback` function. Whenever the user clicked the left mouse button with the smooth brush/eraser as the active tool, we set a `click_and_drag_conscious` flag to true and attached a cursor position callback to the window. This callback invoked the tool's `action()` function whenever the mouse position changed - this `action()` function for smooth tools was implemented in such a manner that it drew a 'line of circles' interpolating between the last and current callback points. Whenever the user released the pressed mouse button, we would set the `click_and_drag_conscious` flag to false and detach the cursor position callback function. The interpolated lines were drawn using the current tool size, using the extension introduced in functionality A.

C. *Primitive modes implementation* - The point, line and triangle primitive modes were all implemented by first designing an abstract base class `primitive_t`, and then inheriting its features to construct `point_t`, `line_t` and `triangle_t` classes. Each of the derived classes followed a common API -
   a. `glm::vec2 pts` - an array of (the latest) previously encountered points, with sizes 1, 2 and 3 in point, line and triangle primitive modes respectively, and initialized with all points as (-1, -1)
   b. `get_point(pos)` - member function that returned the point stored at position `pos` in the `pts` array
   c. `set_point(pt)` - member function that updated the `pts` array by discarding the oldest point, shifting the remaining two backwards and then inserting `pt` as the latest point (position 0)
   d. `get_primitive_mode()` - member function that returns the primitive mode of a tool

   Every time the mouse was clicked, the `set_point()` function was invoked with the mouse coordinates as arguments, for updating the sequence of encountered points. The `action()` function of the active tool was then called, which used `get_point()` to retrieve the appropriate number of points ((-1, -1) points were ignored) and draw a point, line or triangle using them according to the primitive mode type.

D. *Floodfill* - We implemented the non-recursive, four neighbor version of the floodfill algorithm using queues. In order to make the algorithm efficient, we used two queues, one each for `x` and `y` coordinate, instead of using a single one to store `point_t` (this saved us additional cost of creating objects). We also used a boolean matrix of size `height * width` to indicate the pixels which were already colored/visited. The queue(s) was(were) initialized with the source pixel. Atmost four neighbors of every popped pixel were pushed back into the queue(s) after updating their color and the boolean matrix, provided that they had not been visited already and their color matched that of the source pixel. The process was repeated till the queue(s) were empty.

E. *Key-press callbacks* - All callbacks were handled in the `key_callback()` function in `gl_framework.cpp`. `B/b`, `E/e` and `F/f` key-press events were handled by setting the `active_tool` to the corresponding brush/eraser/fill tool, along with handling repeated presses (for toggling between point and smooth mode) when in the point primitive mode. `+/–` key events were handled by calling the `increase_size()/decrease_size()` functions of the active tool. Cycling through the primitive modes on pressing `P/p` was accomplished by changing the `current_primitive` member of `canvas`. For changing foreground or background color, we took RGBA input from the console (range 0-1) and changed the `brush_color` or `bg_color` member of `draw_context_t` accordingly.