Lab 02: 2D Object coloring with OpenGL

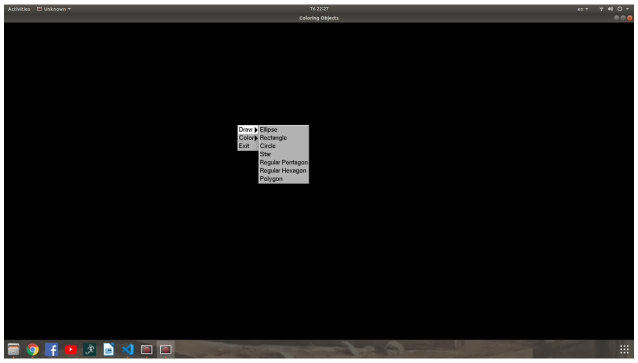
1. Execution

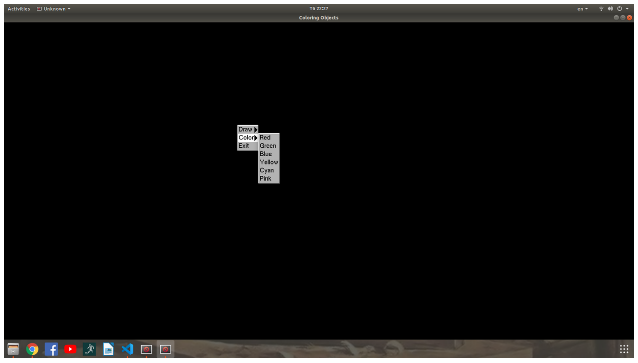
- The source is implemented on Ubuntu 18 with OpenGl 3.0 Mesa 19.2.8
- To compile all the source code. Run `make all`
- The target execution file is `main`
- To run the program `./main`

2. Algorithms

Menu and mouse operation:

- The main menu contains:
  - Submenu for drawing shapes
  - Submenu for coloring
  - Exit: exiting the program



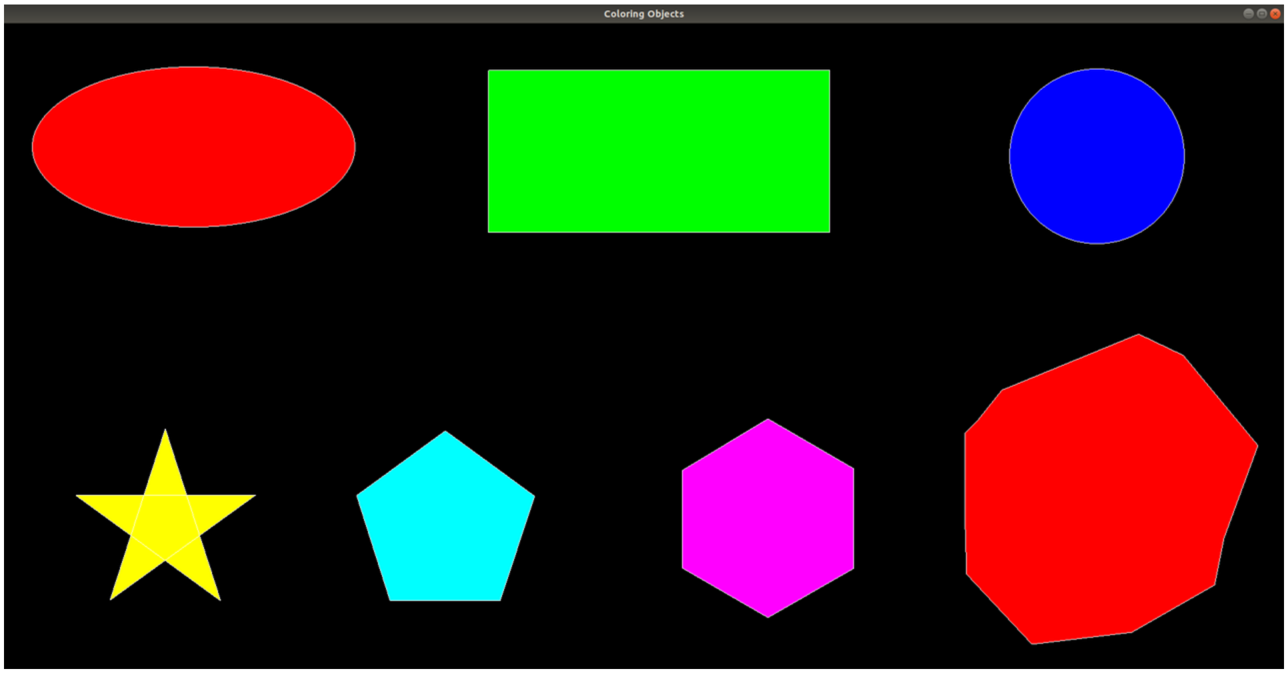


- The menu is attached to right mouse click
- If choosing to draw a shape, the following actions happen:
  - Detach right mouse click from menu (to prevent dupplication with ending polygon)
  - Left mouse down is attach to determining start point of rectangle boundary of the shape
  - Detach left mouse down

    - Left mouse up is attach to determining end point of rectangle boundary of the shape
    - Detach left mouse up
    - Draw shape base on rectangle boundary
    - Re-attach menu to right click
        - Any wrong mouse action in the sequence will cancel the drawing
- If choosing to color, the following actions happen:
    - Left mouse down is attach to determining start point for coloring
    - Color from clicked seed point
        - Any wrong mouse action in the sequence will cancel the coloring

Drawing shapes:

- I implemented algorithms for drawing 7 different shapes, including:
    - Based on midpoint algorithm:
        - Ellipse
        - Circle
    - Base on vertex drawing using built in GL_LINE_LOOP:
        - Rectangle
        - Star
        - Regular Pentagon
        - Regular Hexgon
        - Polygon

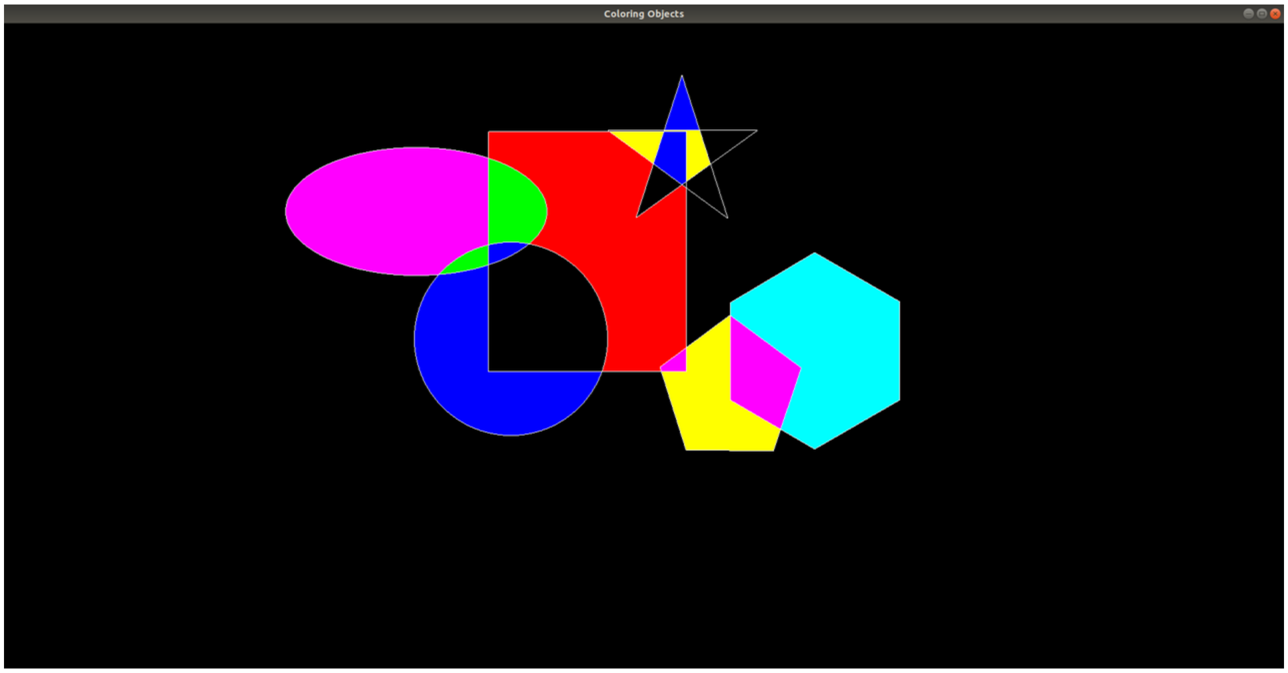


- All the shapes are inherited from a abstract class `Object` with 2 main methods:
  - `getParameters`: infering base points of a shape base on user's mouse operations defining rectangle boundary of the shape
  - `draw`: calculating vertexs of corresponding shapes and calling OpenGL functions to draw

Coloring

- For coloring, I use **Flood Fill** algorithm
- From the seed point, I will start to spread the color to 4-neighbor points with same old color with the seed point until reaching boundary of window or boundary of shapes (white color)
- However, the recursive call and get color, set color, and flush for every pixel is very slow.
- So I change the implementation into using **BFS** and queue
- First I get all pixels value on window and store back in an 2D array
- Then I start the BFS from the seeds points
- For each point pop from the queue:
  - Color that point using `glDrawPixels` (without flush)
  - Update that point's color in the stored colors array
  - Pushing 4-neighbor to queue if inside window and have the same old color
- Finally `glFlush` only at the end
- This approach run significantly faster and don't have to worry about the stack size

3. Demo



4. Limitations and proposing solutions

Placeholder

- When dragging mouse to draw image, there are not showing placeholder image like a preview for user easy to imagine and draw exactly
- Possible solution:
    - Using double buffer
    - Back buffer store the previously drawn shapes and colors
    - For each mouse position change tracked by `glutPassiveMotionFunc`, we will copy the back buffer and draw the preview shape on top of the buffer and swap to front for displaying
    - While still always keeping the back buffer as official drawn shapes and colors saving

Resizing

- When resizing window, the shapes' size is not changing as same ratio as window's width and height but keeping the original size
- Possible solution:
    - In handler of `glutReshapeFunc` for window changing size event, we will update the base points of the shapes base on the ratio between new height

and old height, new width and old width