Lab 03: 2D Affine Transform OpenGL

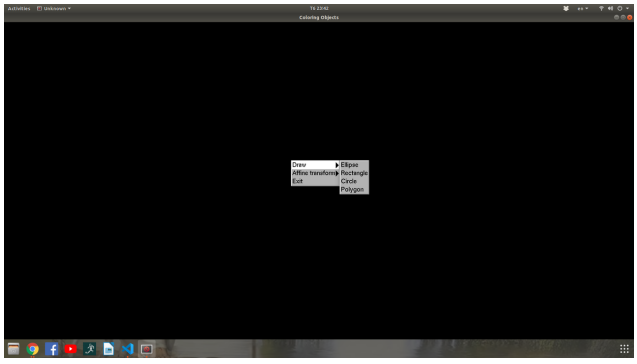
1. Execution

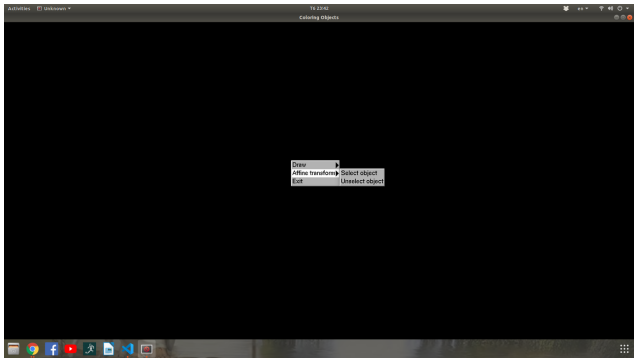
- The source is implemented on Ubuntu 18 with OpenGl 3.0 Mesa 19.2.8
- To compile all the source code. Run `make all`
- The target execution file is `main`
- To run the program `./main`
- Github: https://github.com/nnmhuy/Lab03-OpenGL-Affine-Transform.git

2. Algorithms

Menu and mouse operation:

- The main menu contains:
  - Submenu for drawing shapes
  - Submenu for affine transformation
  - 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 duplication 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 transformation menu, the following actions happen:
    - Left mouse down is attach to select an object
    - Attach keyboard to listen to corresponding actions and transformations:
        - Up, down, left, right: translation
        - l, r: rotating to the left/right
        - +, -: scaling
        - Esc: unselect the object and detach all mapping keyboard functions

Drawing shapes:

- The drawing algorithms is used implementation from Lab 02 with 4 different shapes:
    - Rectangle
    - Circle
    - Ellipse
    - Polygon
- All the shapes are inherited from a abstract class `Object` with:
    - 4 main properties:
        - `center`: the center point of the object
        - `base_points`: set of vertexes forming the object
        - `angle`: the accumulate angle by rotating transformation comparing to original position
        - `sScale`: the accumulate scale by scale transformation comparing to original size
    - 3 main methods:
        - `getParameters`: inferring center of a shape base on user's mouse operations defining rectangle boundary of the shape
        - `draw`: calculating vertexes of corresponding shapes and store in `base_points`
        - `drawScreen`: calling appropriate OpenGL functions to draw the shape to the window

    - `rotate`: rotate transformation
    - `translate`: translate transformation
    - `scale`: scale transformation
    - `isPointInside`: determining if the selected point is inside the object

Affine transformation

- This is consisting two main parts:
  - Selecting object: base on clicked point to determined which object to focus
  - Transformation: executing affine transformations

Selecting object

- I divided into two kinds of objects:
  - Polygon type (store by vertexes) - default implementation `isPointInside` of class Object: including custom polygons and rectangle
    - I use the ray casting algorithm: by drawing a line through the selected point and counting number of intersections between that line and the objects to determined if it is inside or not. If the number of intersections with x-coordinate smaller or equal than the selected point x-coordinate is odd → inside and if even → outside
  - Other shape with can expressed in equation form: including circle and ellipse
    - I substitute the point into the shape formula and base on the sign of the result to determine inside or outside
- Special kinds of shapes override the implementation of virtual pure function check `isPointInside` of class Object
- When selecting a point, simply run through the objects from the lasted object added to check if the point is inside that object and return the first found

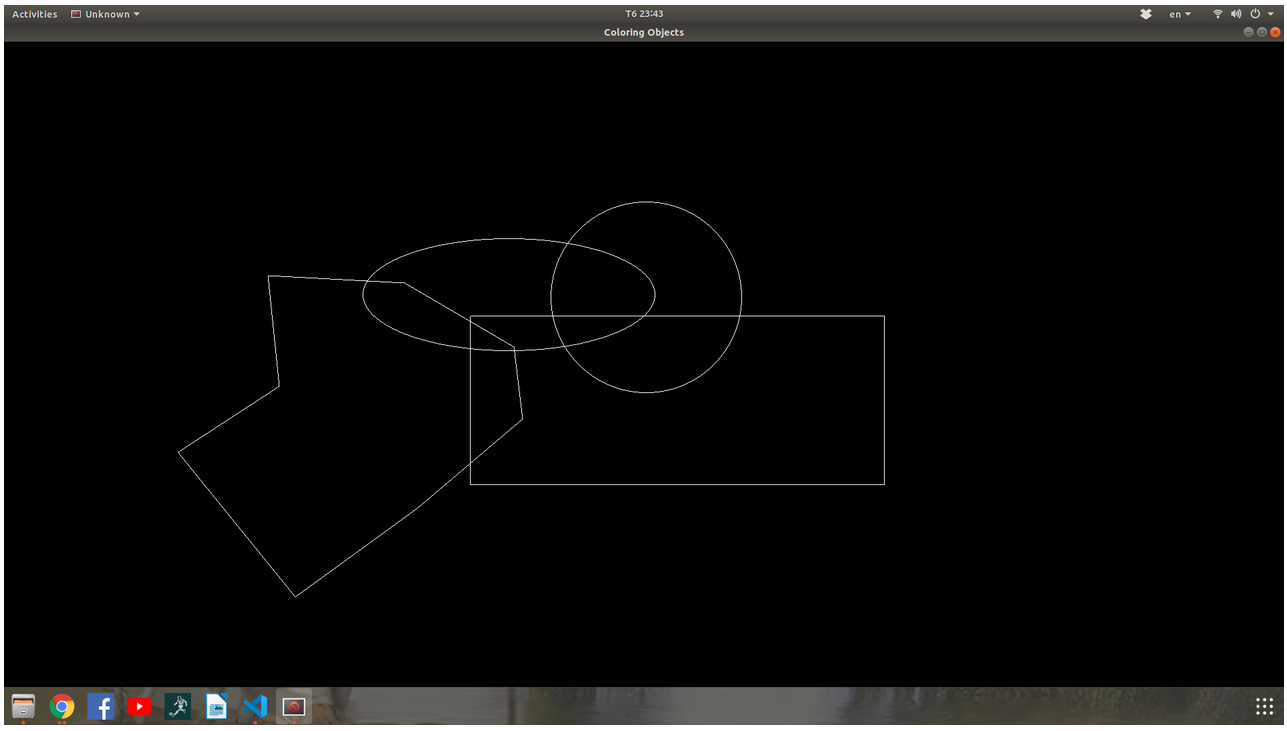
Transformation

- The transformation of each shape is by calling appropriate transformation of that shape's center point and base points methods explained as below
- **Translation:**
  - This is simply by adding `dx` and `dy` to the center and all points in base points of the object
- **Rotate:**
  - First, I implemented this by transform all base points of the object at each steps. But this approach has the problem with rounding float number and the error is accumulated to deform the shape

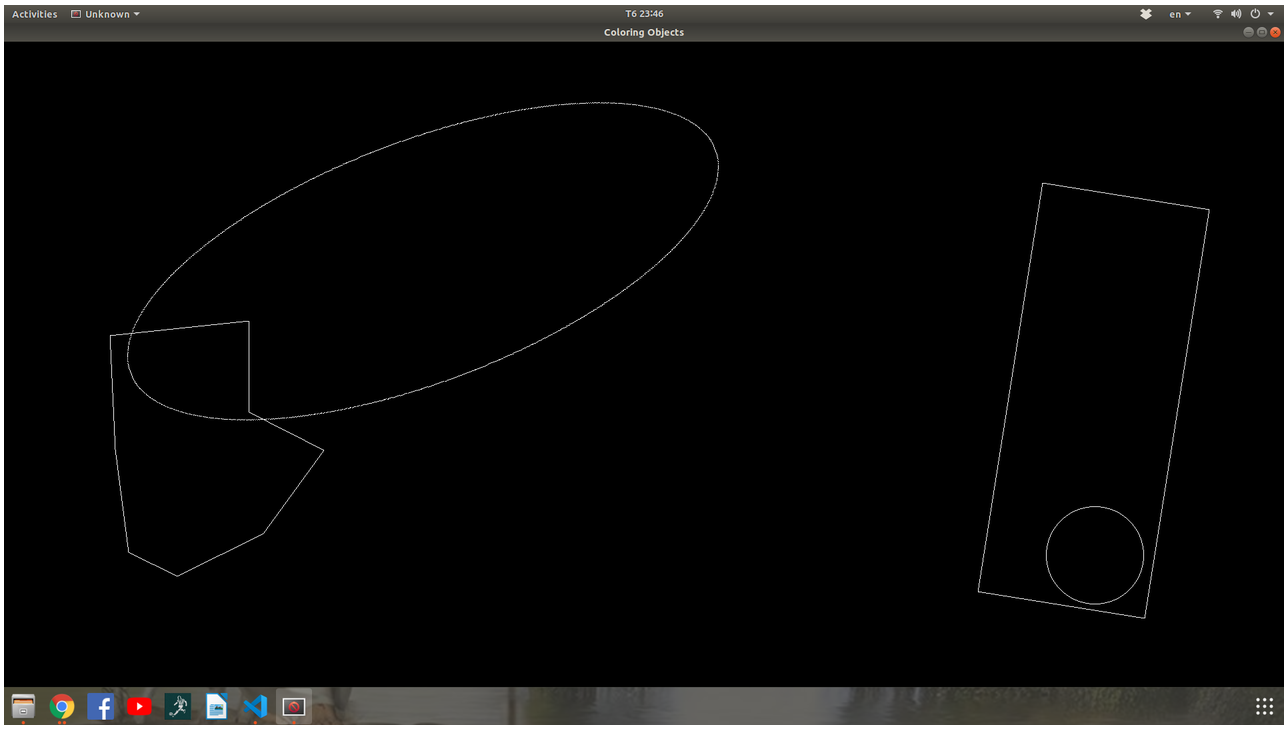
    - So I come with new approach by adding the rotate angle into an accumulating variable `angle` for each object. And I only apply the rotation for each of base points when drawing the shape to screen. This approach minimize the error and then not deforming the shapes.
- **Scale:**
    - This use similar approach as rotating
    - The scale is accumulated to variable `sScale` and only apply to points when executing `drawScreen`
    - However, with circle and ellipse, scaling by this method made the shape points make great distance between continuous points since these shapes are drawn by putting individual points. So I override the implementation of scaling for circle and ellipse by just scaling the radius or axises of these shape and re-calculate all the base points recalling `draw` method.

3. Demo

- Before applying transformation



- After transformation


Activities
Unknown
T6 23:46
en
Coloring Objects