Student ID: 1751036
Student Name: Tran Thi Anh Thu
CS411 – Lab03

Lab03 Report
Affine 2D Object Transformation

Program Description

The program lets users draw 4 kinds of shapes: Rectangle, Circle, Ellipse and Polygon and transform them as translation, rotation and scaling.

Usage íntruction:

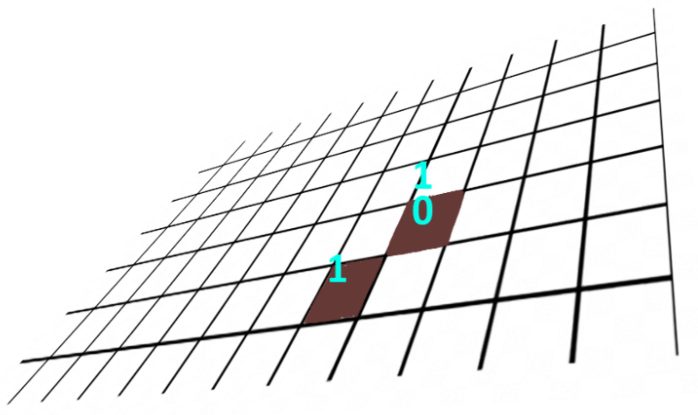
- First time opening, the default shape is Rectangle
- To draw shapes:
    - Rectangle, Circle, Ellipse: click a start point and drag mouse to an end point, then release mouse.
    - Polygon: click a start point and keep clicking next point as polygon's next vertices, to close the shape or finish a polygon, press Esc.
    - Right click mouse to open a menu to choose other shapes.
- To transform shape:
    - After drawing a shape, user can transform it immediately
    - To choose other shape for transforming, hold Ctrl key and Left mouse click on other shape's boundary.
    - Transform controls are:
        - Translate: Left / Right / Up / Down arrows.
        - Rotate: L/l/R/r.
        - Scale: + / -.

Implementation of manually transformation:

1. *Clear()*
   To achieve the effect that the shape appears at other position, we need to *delete* its pixel's old position. In the clear() function, I simply overdraw the pixel with color WHITE.
2. *Update_variables()*
   Based on the required type of transformation, update the shape' anchor points.
3. *Redraw()*
   Redraw shape at new position by simple calling shape's draw() function.

Implementation of overlapped shapes:

Keep a (HEIGHT+1) x (WIDTH+1) 2D array of std::set, then when a shape is drawn, all of its point at (x_i,y_i) will be record at the (y,x) element of the 2D array by pushing the shape ID to that std::set, illustrated as figure on the right.



I use std::set because shapeID must be unique, and avoid pushing the same ID too many times when the windows refreshes as well as the corners of rectangle where the edges intersect.

When a pixel is drawn, its corresponding set is inserted with that shape ID. When a pixel is cleared, its corresponding set erases that shape ID.

Translation

Keyboard:

- Left: dx = -1, dy = 0
- Right: dx = +1, dy = 0
- Up: dx = 0, dy = 1
- Down: dx = 0, dy = -1

Rectangle:

4 anchor points += dx, dy;
Redraw by calling draw line with each pair of such new values (in a correct order for a rectangle).

Circle:

cX, cY += dx, dy, radius unchanged.
Redraw with such new values.

Ellipse:

I use a different way to draw an ellipse by using this formula:

```
for (float theta = 0; theta < 2 * PI; theta += 0.005) {
       x = cX + a * cos(theta) * cos(alpha) - b * sin(theta) * sin(alpha);
       y = cY + a * cos(theta) * sin(alpha) + b * sin(theta) * cos(alpha);

       myMap[y][x].insert(shapeID);

       glVertex2i(x, y);
}
```

Where alpha is the shape's rotation degree, when first draw, alpha = 0;
Therefore, cX, cY += dx, dy
Redraw with such new values.

Polygon:

I keep a vector of polygon's vertices that is: x0, y0, x1, y1, x2, y2, xN, yN, x0, y0
Then iterate over the vector to add dx and dy correspondingly to each x-coordinate and y-coordinate.
Redraw that polygon with such new values by call draw line with each pair of points sequentially.

Rotation

Keyboard:

- R or r: d = 1
- L or l: d = -1

Rectangle:

Convert d to radian
For each anchor point (4 anchors):

```
x_i = cos(alpha) * (x_i - cX) - sin(alpha) * (y_i - cY) + cX;

y_i = sin(alpha) * (x_i - cX) + cos(alpha) * (y_i - cY) + cY;
```

cX and cY is center point of current rectangle
Redraw by calling draw line with each pair of such new values (in a correct order for a rectangle).

Circle:

```
void rotate(int d) {
       return;
}
```

Ellipse:

For the above way to draw an ellipse, I simply set alpha to the inputted degree.
Redraw with such new values.

Polygon:

Then iterate over the vector to change each x_i and y_i the same manner as the functions used for Rectangle's rotation above.
Redraw that polygon with such new values.

Scale

Keyboard:

- + : d = +1
- - : d = -1

Disclaimer: The shape is not scaled by a factor of 0.9 or 1.1 in area unit. Rather than that, I scale its edge by the factors, which makes its overall shape will appear to be bigger or smaller by a small amount.

Rectangle:

```
              float cX = (x1 + x2) / 2;
              float cY = (y1 + y2) / 2;

              x1 = x1 * (1 + d * 0.1);
              x2 = x2 * (1 + d * 0.1);
              xA = xA * (1 + d * 0.1);
              xB = xB * (1 + d * 0.1);

              y1 = y1 * (1 + d * 0.1);
              y2 = y2 * (1 + d * 0.1);
              yA = yA * (1 + d * 0.1);
              yB = yB * (1 + d * 0.1);

              float ccX = (x1 + x2) / 2;
              float ccY = (y1 + y2) / 2;
```

```
            translateNoRender(- ccX + cX, - ccY + cY);

            redraw();
```

The idea is to increase all vertices, then translate the rectangle back to its original position.

Circle:

Scale the radius by 0.9 or 1.1 factor based on signed of d.
Then redraw.

Ellipse:

Scale a and b by 0.9 or 1.1 factor based on signed of d.
Redraw with such new a and b and current degree of rotation.

Polygon:

Then iterate over the vector to change each x_i and y_i the same manner as the functions used for Rectangle's scale above, then translate all the vertices back to its original position.
Redraw that polygon with such new values.

Windows Size Limit

The windows origin O(0, 0) is placed at top left corner to benefit the indexing of the (HEIGHT+1) x (WIDTH+1) 2D array of std::set.

In the three sequentially function call of *clear(), update_variable()* and *redraw(),* the *clear()* and *redraw()* (which internally calls shape's *draw()* function) return immediately if the coordinates passed to them are out of windows sizes. However, *update_variable()* is still running, this helps me keep track of out-of-screen shape. For that, whenever the shape's coordinates are back within the windows size, the shape is rendered as normal.

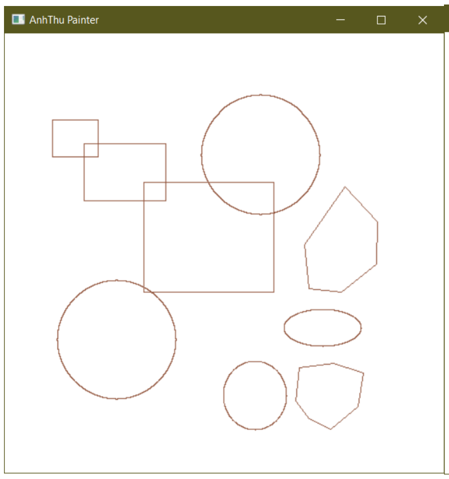
Determine Object to Transform

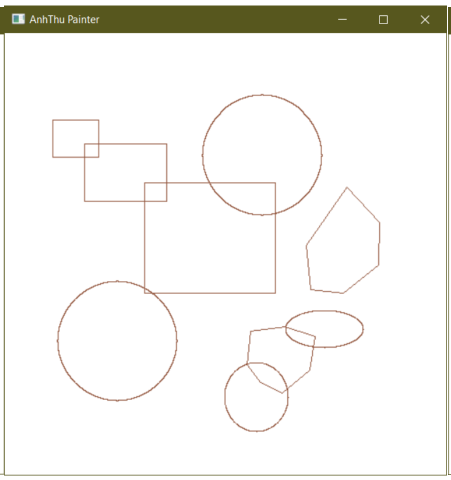
A shape creation is as followed:

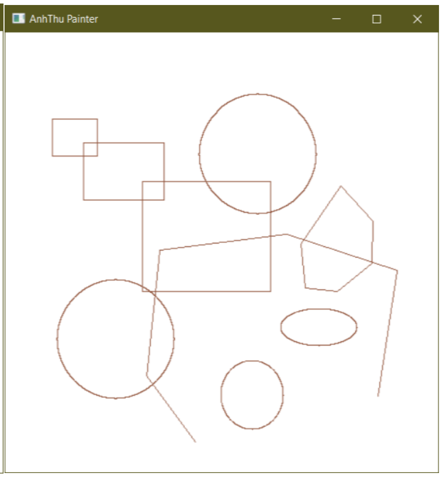
- Initially, set draw mode to Rectangle, waiting to get input from user mouse click
- When left mouse is down, allocate a new shape according to the current chosen mode. If users are drawing a polygon, don't allocate new shape but continue update the polygon's vertices.
- When left mouse is up, if a polygon is still drawing when mouse up then do nothing. Otherwise, render the shape to windows and record its x_i, y_i to the 2D array.
- At this point, the just-finished-drawing shape is the current shape that can be transform immediately without the need to click on it.
- For polygon, its shape is render immediately when new points are clicked along with updating the 2D array. The Esc key connects the current point with the polygon's start point (drawing last stroke).
- Hold Ctrl and click on any shape's boundary. The latest shape ID retrieved at the [y][x] std::set of the 2D array is returned to be the index into the array of shapes. This gives back which shape is clicked on (chosen). Such shape's pointer is assigned to the current shape pointer so that user can transform it. The latest shape ID is the largest in the std::set.

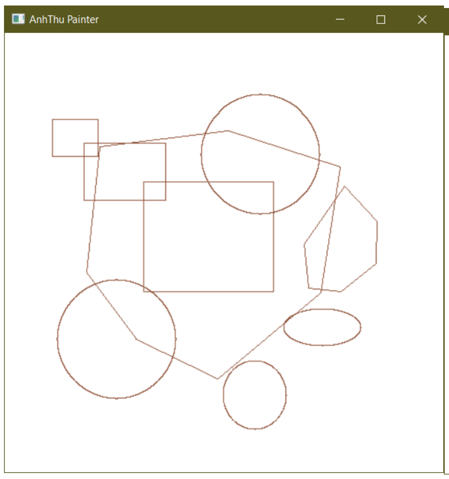
Demo

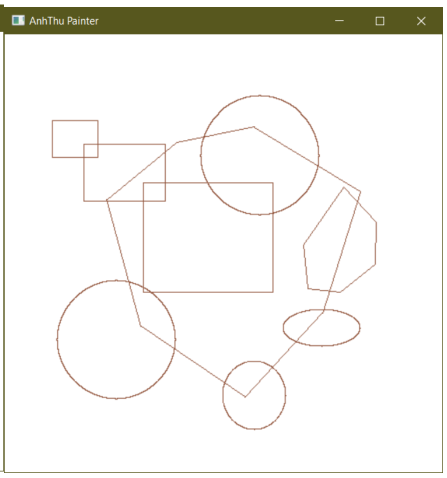
First, draw shapes -> choose and move the small polygon at the bottom right -> scale it up (note it doesn't render edges that cross the windows boundary) -> move it to the center -> rotate it



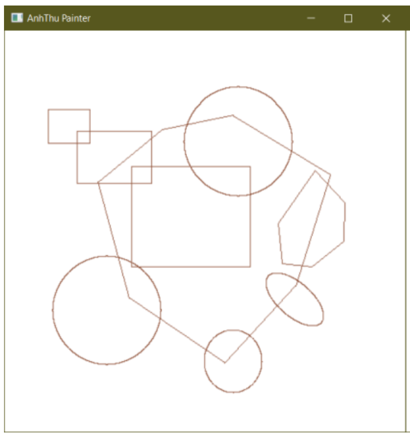


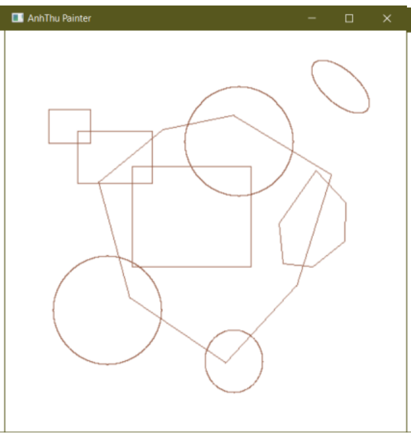


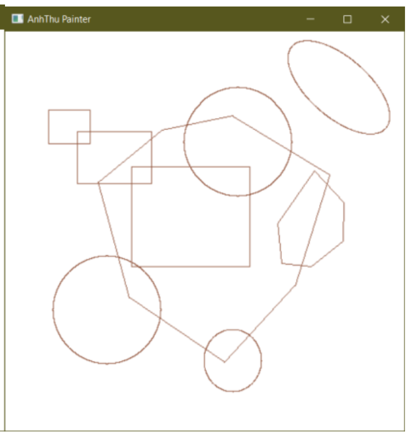


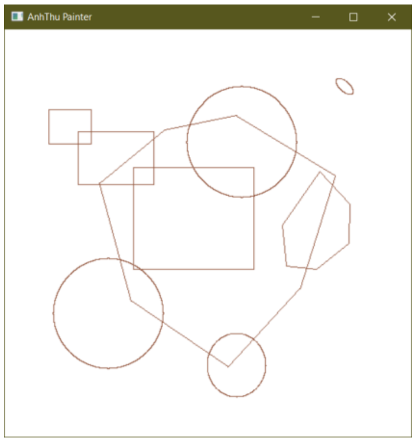


Now rotate the higher ellip on the right-> move it to top right -> scale it up -> scale it down

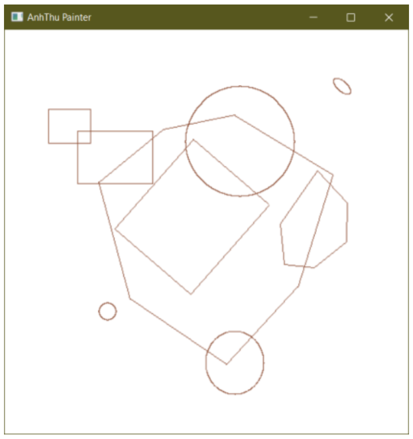




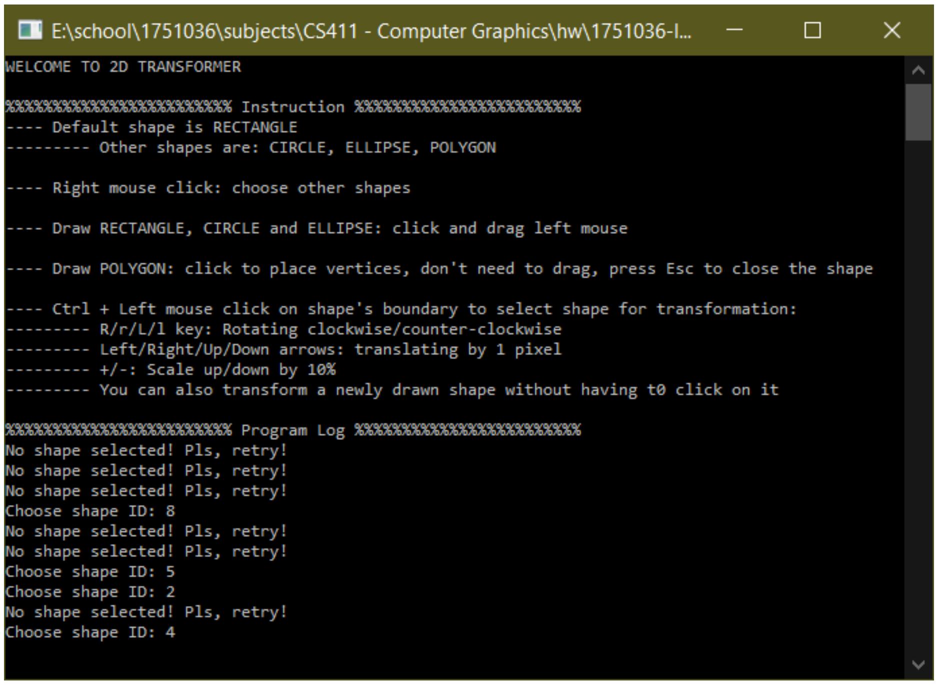




Now, rotate the middle rectangle and scale the left circle down



The console output for this demo is:



E:\school\1751036\subjects\CS411 - Computer Graphics\hw\1751036-l...

```
WELCOME TO 2D TRANSFORMER

%%%%%%%%%%%%%%%%%%%%%%%% Instruction %%%%%%%%%%%%%%%%%%%%%%%%
---- Default shape is RECTANGLE
--------- Other shapes are: CIRCLE, ELLIPSE, POLYGON

---- Right mouse click: choose other shapes

---- Draw RECTANGLE, CIRCLE and ELLIPSE: click and drag left mouse

---- Draw POLYGON: click to place vertices, don't need to drag, press Esc to close the shape

---- Ctrl + Left mouse click on shape's boundary to select shape for transformation:
--------- R/r/L/l key: Rotating clockwise/counter-clockwise
--------- Left/Right/Up/Down arrows: translating by 1 pixel
--------- +/-: Scale up/down by 10%
--------- You can also transform a newly drawn shape without having t0 click on it

%%%%%%%%%%%%%%%%%%%%%%%% Program Log %%%%%%%%%%%%%%%%%%%%%%%%
No shape selected! Pls, retry!
No shape selected! Pls, retry!
No shape selected! Pls, retry!
Choose shape ID: 8
No shape selected! Pls, retry!
No shape selected! Pls, retry!
Choose shape ID: 5
Choose shape ID: 2
No shape selected! Pls, retry!
Choose shape ID: 4
```

The end.