UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

PROJECT REPORT

LAB 3 - COMPUTER GRAPHICS

Nguyen Dinh Ngoc Tri - 21125065

CLASS: 21APCS2

CONTENTS

CHAPTER 1

WRITE A PROGRAM BY GLUT

1.1 Initialization

1.1.1 Initialize the GLUT library

```
glutInit(&argc, argv);
```

The `glutInit(&argc, argv)` function initializes the GLUT library, which is used for creating graphical user interfaces (GUIs) in OpenGL programs. It initializes GLUT with the command line arguments `argc` and `argv`, which are typically passed to the program's `main` function.

1.1.2 Set the initial window size

```
glutInitWindowSize(800, 600);
```

The line `glutInitWindowSize(800, 600);` sets the initial size of the window created by GLUT to 800 pixels in width and 600 pixels in height.

1.1.3 Creating an OpenGL Window

```
glutCreateWindow("Computer Graphics");
```

The line `glutCreateWindow("Computer Graphics");` creates a window with the specified title "Computer Graphics" using GLUT.
After initializing the window screen, if the program runs, it will display a black screen with the specified size.

1.1.4 set up the projection matrix in OpenGL

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, 800, 0, 600);
```

The code first sets the matrix mode to `GL_PROJECTION`, which specifies that subsequent matrix operations will affect the projection matrix. Then, `glLoadIdentity()` loads the identity matrix onto the current matrix stack, effectively resetting it. Finally, `gluOrtho2D()` sets up a 2D orthographic projection with the specified parameters, defining the mapping from normalized device coordinates to window coordinates. This setup is essential for correctly rendering 2D graphics within the specified window dimensions.

1.1.5 Register a display function

```
glutDisplayFunc(display);
```

The line `glutDisplayFunc(display);` registers the `display()` function as the callback function to be called whenever the display needs to be redrawn. This function is crucial for rendering graphics within the OpenGL window. Typically, this function contains OpenGL rendering commands to draw graphics or scenes onto the window. By registering the display function, OpenGL ensures that the graphics are updated appropriately when needed.

1.1.6 Start the main event processing loop

```
glutMainLoop();
```

The line `glutMainLoop();` is a call to GLUT's main event processing loop. This function enters an infinite loop where it waits for events such as user input, window resizing, and timer ticks. It continuously processes these events and invokes the corresponding callback functions that were registered using GLUT's registration functions (e.g., `glutDisplayFunc()`, `glutReshapeFunc()`, etc.). This loop ensures that your OpenGL application remains interactive and responsive to user actions.

1.2 Drawing 2D Objects

OpenGL provides support for drawing fundamental shapes such as points, lines, and polygons.

To draw a set of pixels at positions $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, the following OpenGL commands are used:

```
glBegin(GL_POINTS);
glVertex2i(x1, y1);
glVertex2i(x2, y2);
...
glVertex2i(xn, yn);
glEnd();
glFlush();
```

The `glBegin()` function specifies the type of shape to be drawn, with `GL_POINTS` representing a set of points. Each call to `glVertex()` defines a vertex of the shape.

The `glEnd()` function marks the end of the shape definition block.

Upon calling `glFlush()`, the shape is rendered on the screen.

Additionally, to color the vertices, the `glColor3f(r, g, b)` function precedes the `glVertex()` calls, specifying the RGB color values (r, g, b).

The color of each vertex is determined by the nearest preceding `glColor()` call. If none exists, the default color is white.

1.3 Translate

```
void Poly::translate(float dx, float dy) {
    for (auto& vertex : vertices) {
        int newx = (int) ( (float) vertex.getX() + dx);
        int newy = (int) ( (float) vertex.getY() + dy);
        vertex.setX(newx);
        vertex.setY(newy);
    }
};
```

Listing 1.1: RGBColor struct

1.4 Rotate function

```
void Poly::rotate(float angleDegrees) {
    float angleRadians = angleDegrees * M_PI / 180.0f;
    float cosTheta = cos(angleRadians);
    float sinTheta = sin(angleRadians);

    for (auto& vertex : vertices) {
        float newX = (float) vertex.getX() * cosTheta - (float) vertex.getY() * sinTheta;
        float newY = (float) vertex.getX() * sinTheta + (float) vertex.getY() * cosTheta;
        vertex.setX((int) newX);
        vertex.setY((int) newY);
    }
}
```

Listing 1.2: BoundaryFill algorithm

1.5 Scale function

```
void Poly::scale(float scaleFactor) {
    for (auto& vertex : vertices) {
        float newX = (float) vertex.getX() * scaleFactor;
        float newY = (float) vertex.getY() * scaleFactor;
        vertex.setX((int) newX);
        vertex.setY((int) newY);
    }
}
```

Listing 1.3: BoundaryFill algorithm

CHAPTER 2

RESULTS

2.1 How to run

2.1.1 Selecting a Polygon

- Right-click anywhere inside the window to open the context menu. - To open Menu, click the mid cursor on your mouse and then select available shapes in submenu.

2.1.2 Transforming Polygons

After selecting a polygon type, you can perform transformations using hotkeys:

- Rotation: Use the <L> key to rotate the polygon 1 degree clockwise per press, and the <R> key to rotate it counterclockwise.
- Translation: Use the arrow keys (up, down, left, right) to move the polygon by 1 pixel in the corresponding direction per press.
- Scaling: Use the <+> key to scale up the polygon by 10% per press, and the <-> key to scale it down.

2.2 Menu

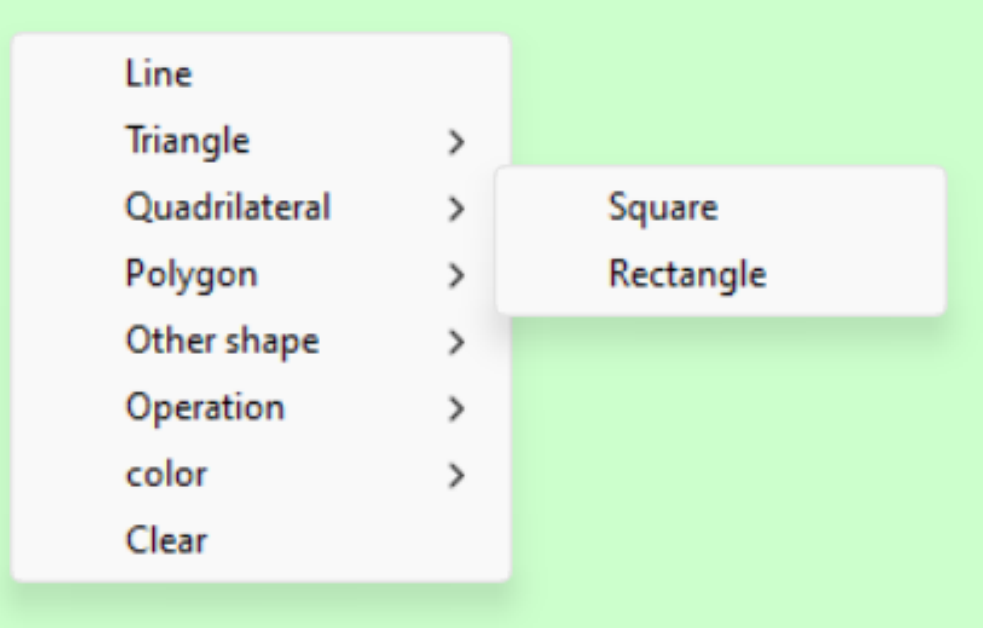

Figure 2.1: Menu

2.3 Transforming Shape

- I am unable to demonstrate the process of transforming a specific shape in a moving picture. However, I can provide a visual representation by merging two images: one showing the shape before transformation and another showing it after the transformation to help you understand how this works.

- Rotate around point(0,0)

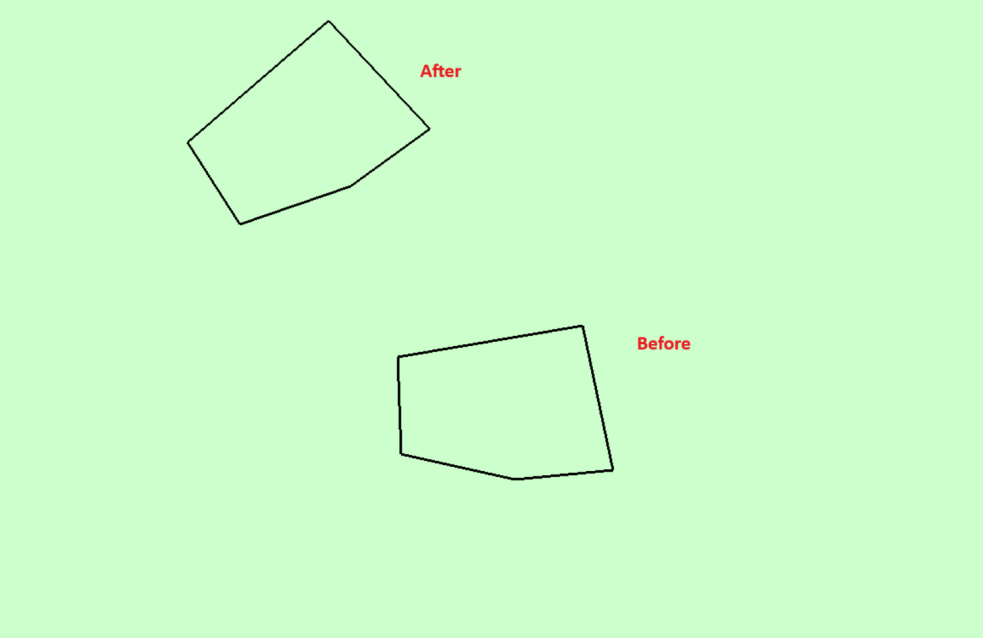

Figure 2.2: Rotate around point(0,0)

-Scale up:

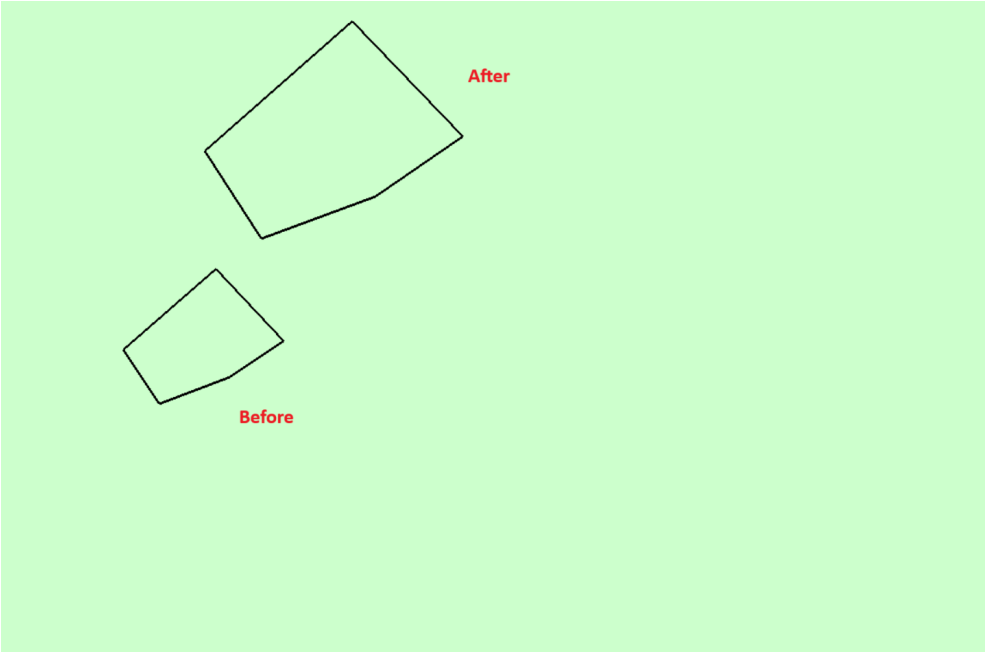

Figure 2.3: Scale up

- Translate to right:

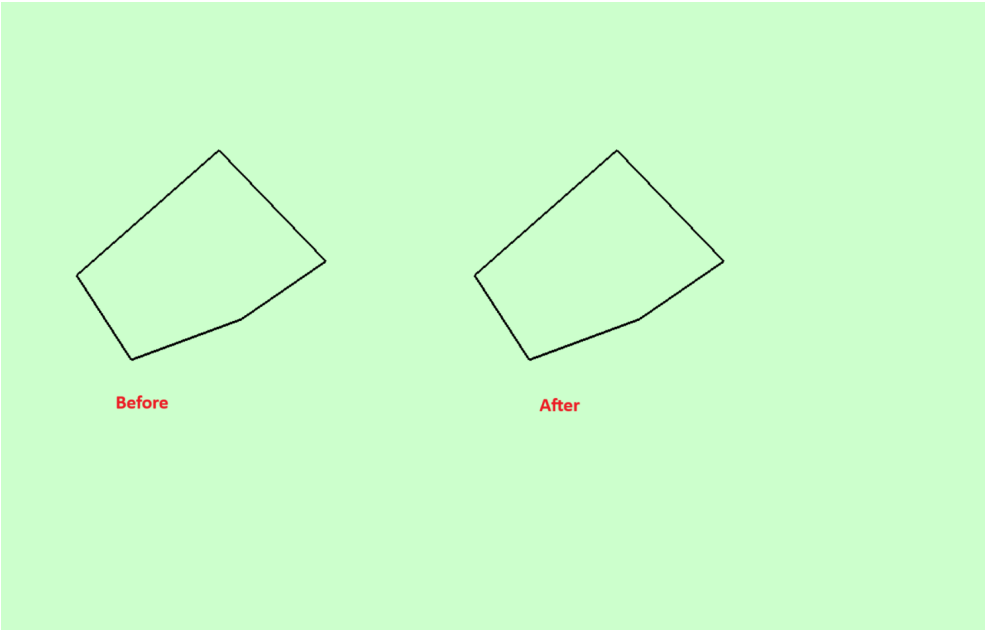

Figure 2.4: Scale