UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

PROJECT REPORT

LAB 1 - 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.

CHAPTER 2

DRAWING TIME COMPARISON

2.1 Comparation of execute time

Table 2.1: Drawing Time Comparison (microseconds)

Drawing Method	**OpenGL Function**	**DDA Algo**	**Bresenham Algo**	**Midpoint Algo**
Line	34	6406	640	-
Circle	499	-	-	599
Ellipse	336	-	-	975
Parabola	1708	-	-	349
Hyperbola	336	-	-	411

- Comments:

- For line drawing, the Digital Differential Analyzer (DDA) algorithm consistently exhibits the highest execution time, followed by the Bresenham algorithm. In contrast, OpenGL functions demonstrate significantly lower execution times, indicating their efficiency in handling line drawing operations.
- In circle drawing, both the midpoint algorithm and OpenGL functions offer competitive execution times, with OpenGL functions slightly outperforming the midpoint algorithm.
- The execution times for ellipse drawing tasks highlight the superior efficiency of OpenGL functions compared to the midpoint algorithm, with OpenGL functions demonstrating notably lower execution times.
- For parabola drawing, the midpoint algorithm displays relatively shorter execution times compared to OpenGL functions, indicating its efficiency in handling such tasks. However, OpenGL functions in here is coded by myself to draw a specific equations. This suggests that the comparison of execution times is inaccurate.

2.2 Accuracy and Precision Assessment

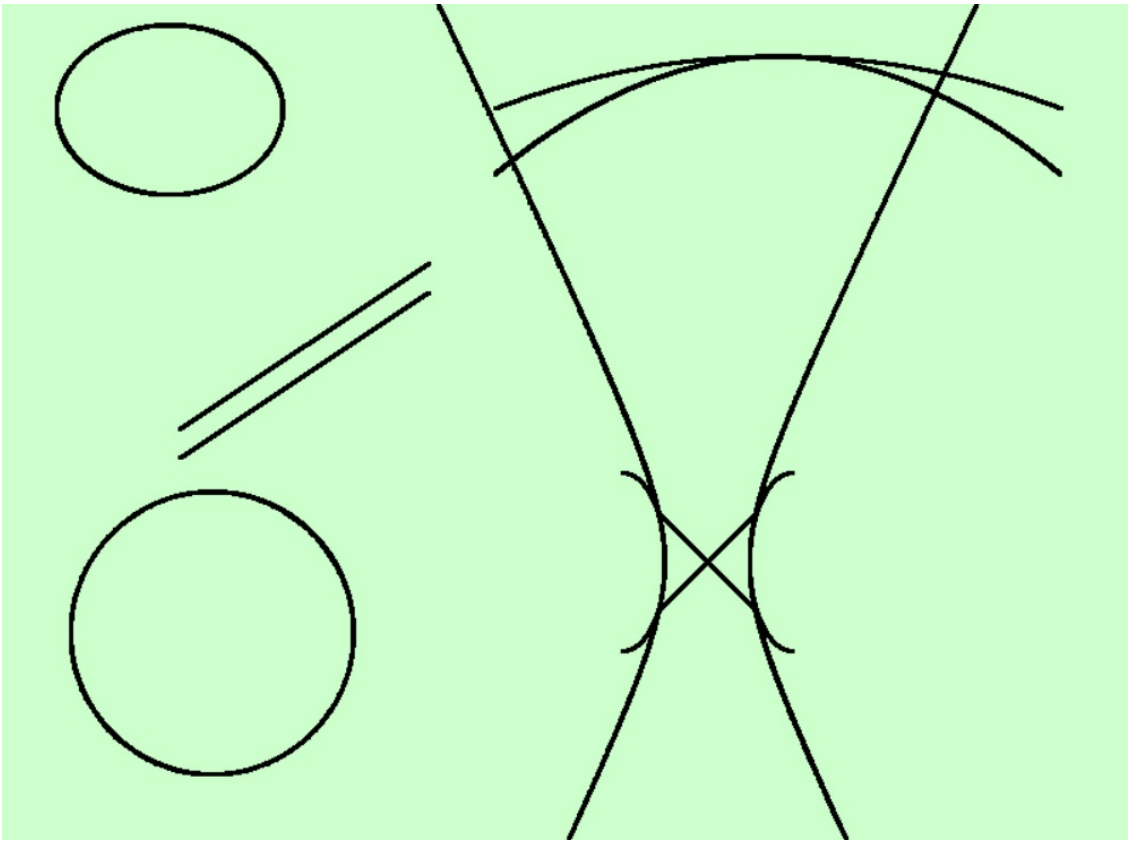

Figure 2.1: Drawing result.

- Comments:

- In the context of drawing straight lines, circles, and ellipses, the rendered images produced by both OpenGL functions and the implemented algorithms appear virtually identical to the naked eye. From this observation, it can be inferred that the accuracy of the implemented algorithms for drawing lines, circles, and ellipses is remarkably high.
- However, when it comes to rendering parabolas and hyperbolas, OpenGL functions lack predefined functions for drawing these curves. Consequently, I devised my own formulas for rendering them. Upon examination, noticeable differences emerged in the rendered images among the various algorithms. Although initially appearing congruent, discrepancies became apparent as the curves extended further. This suggests that while the initial stages of the curves may appear similar, discrepancies become more prominent as they progress.

- Conclusion:

- The assessment of the accuracy of the implemented algorithms for drawing geometric shapes reveals notable findings. Overall, the algorithms demonstrate a high level of precision, particularly evident in the rendering of straight lines, circles, and ellipses. When compared visually, the images produced by both OpenGL functions and the implemented algorithms appear virtually identical.
- However, deviations in accuracy become apparent when dealing with parabolic and hyperbolic curves. Due to the absence of predefined functions for drawing these curves in OpenGL, custom formulas were employed. While the initial segments of these curves may appear similar across algorithms, discrepancies emerge as the curves progress, indicating potential limitations in accuracy.