ICS 2311: Group 15 - Ellipse Drawing, Transformation and Filling

Introduction

This document explains how we approached the ellipse drawing and transformation assignment for Group 15 in ICS 2311. We've implemented solutions using both C++ and Python with OpenGL and GLUT to handle:

- Drawing an ellipse from its mathematical equation
- Applying flood-fill algorithm with cyan color
- Implementing shear transformations
- Boundary filling the transformed ellipse with green color
- Anti-aliasing techniques for smooth elliptical boundaries

Code repository: https://github.com/canonallan/OpenGL.git

Understanding the Problem

The assignment asks us to:

1. Draw an ellipse with center at (-2,2) given by an equation
2. Apply flood-fill algorithm to fill the ellipse with cyan color
3. Apply shear parameters (2 on X-axis, 2 on Y-axis) and find new coordinates
4. Boundary fill the transformed ellipse with green color
5. Develop anti-aliasing techniques for elliptical boundaries

Mathematical Foundation

Ellipse Equation

The standard form of an ellipse equation is:

$(x-h)^2/a^2 + (y-k)^2/b^2 = 1$

Where:

- (h,k) is the center of the ellipse

- a is the semi-major axis length
- b is the semi-minor axis length

For our problem, the center is at (-2,2), with a=3 and b=2.

Parametric Form

To draw the ellipse, we use the parametric equation:

- $x = h + a * \cos(\theta)$
- $y = k + b * \sin(\theta)$

This allows us to generate points along the ellipse by varying θ from 0 to 2π.

Implementation Approach

1. Drawing the Original Ellipse

We begin by setting up the coordinate system and drawing the ellipse using the parametric equations:

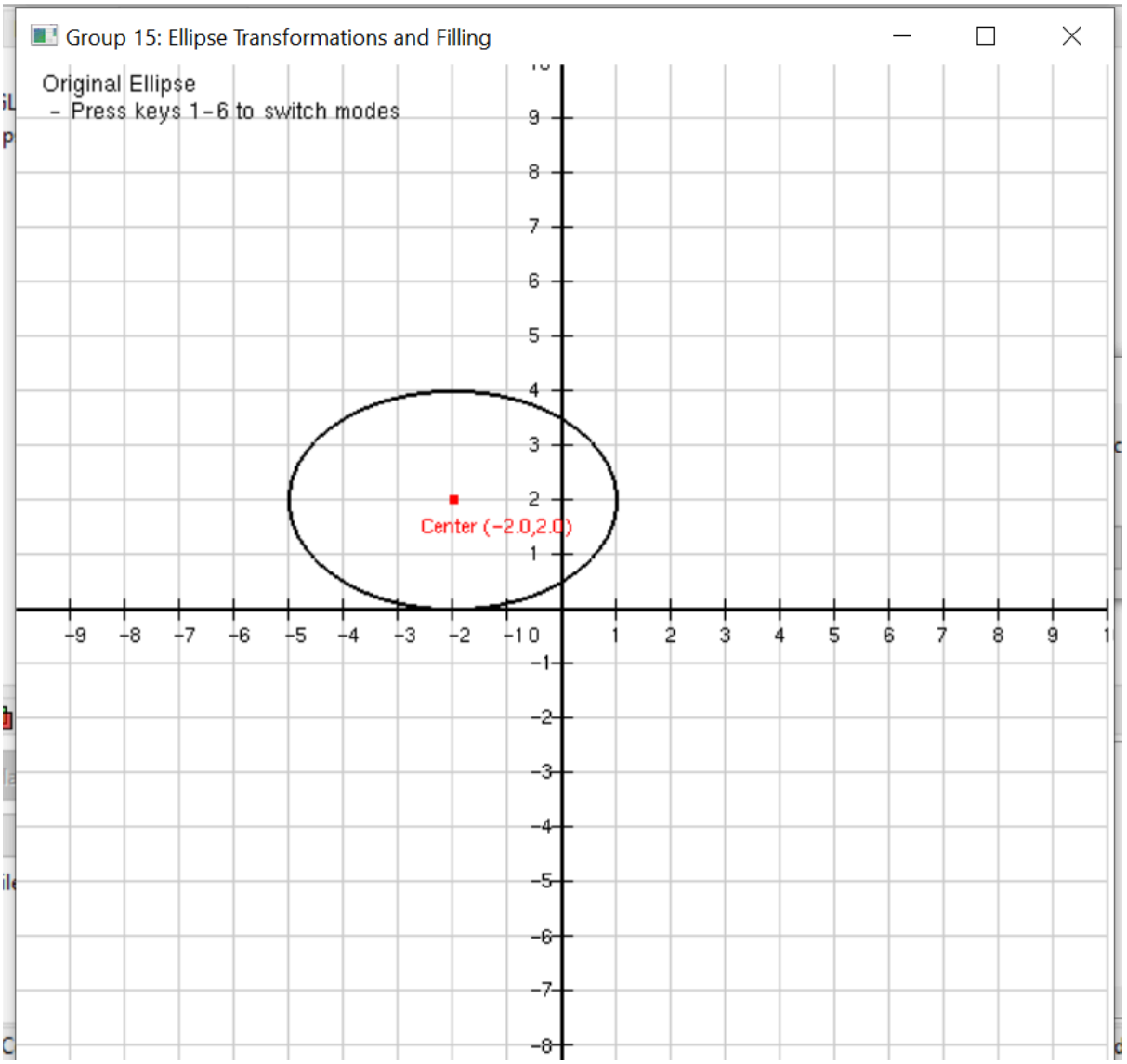
In C++:

```cpp
// Constants for the ellipse equation
const float a = 3.0f; // Semi-major axis
const float b = 2.0f; // Semi-minor axis
const float centerX = -2.0f;
const float centerY = 2.0f;

void generateEllipsePoints() {
    originalEllipsePoints.clear();

    // Create points for the ellipse with higher resolution for smoother curve
    const int numSegments = 100;
    for (int i = 0; i <= numSegments; i++) {
        float t = 2.0f * M_PI * i / numSegments;
        float x = centerX + a * cos(t);
        float y = centerY + b * sin(t);
        originalEllipsePoints.push_back(Point(x, y));
    }
}

void drawOriginalEllipse() {
    glColor3f(0.0f, 0.0f, 0.0f); // Black
    glLineWidth(2.0f);

    glBegin(GL_LINE_LOOP);
    for (size_t i = 0; i < originalEllipsePoints.size(); i++) {
        glVertex2f(originalEllipsePoints[i].x, originalEllipsePoints[i].y);
    }
    glEnd();

    // Mark center
    glPointSize(5.0f);
    glColor3f(1.0f, 0.0f, 0.0f); // Red
    glBegin(GL_POINTS);
    glVertex2f(centerX, centerY);
    glEnd();
}
```

In Python:

python

```python
def draw_original_ellipse():

      glColor3f(0, 0, 1)  # Blue
      glLineWidth(2.0)

    # Draw ellipse using parametric equation
      glBegin(GL_LINE_LOOP)
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)
        glVertex2f(x, y)
    glEnd()

    # Mark the center
    glColor3f(1, 0, 0)  # Red
    glPointSize(5.0)
    glBegin(GL_POINTS)
    glVertex2f(center_x, center_y)
    glEnd()
```



2. Flood-Fill Algorithm (Cyan)

The flood-fill algorithm traditionally fills a bounded area by starting from a seed point and expanding outward until hitting a boundary. However, for this implementation, we use a more efficient approach for OpenGL by filling the ellipse using GL_POLYGON:

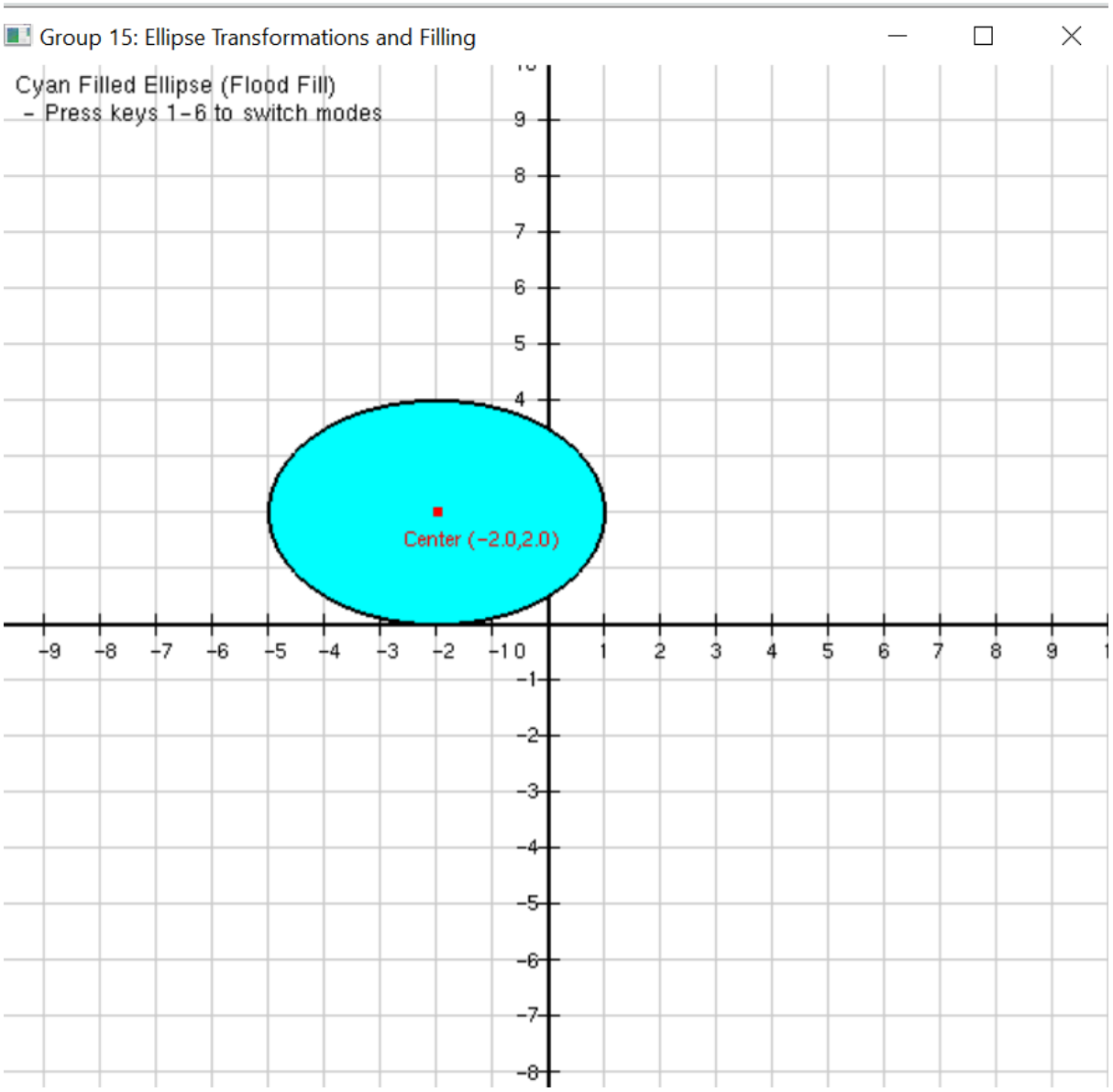
In C++:

```cpp
void floodFillOriginalEllipse() {
    // Fill the ellipse with cyan color
    float fillColor[3] = {0.0f, 1.0f, 1.0f}; // Cyan (RGB)

    // Create a simple filled approximation of the ellipse
    glColor3f(fillColor[0], fillColor[1], fillColor[2]);
    glBegin(GL_POLYGON);
    for (size_t i = 0; i < originalEllipsePoints.size(); i++) {
        glVertex2f(originalEllipsePoints[i].x, originalEllipsePoints[i].y);
    }
    glEnd();
}
```

In Python:

```python
def draw_filled_ellipse():
    """Draw the original ellipse filled with cyan color"""
    glColor3f(0, 1, 1)  # Cyan

    # Draw filled ellipse using parametric equation
    glBegin(GL_POLYGON)
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)
        glVertex2f(x, y)
    glEnd()

    # Draw the border
    glColor3f(0, 0, 1)  # Blue
    glLineWidth(2.0)
    glBegin(GL_LINE_LOOP)
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)
        glVertex2f(x, y)
    glEnd()
```

This approach uses the parametric form of the ellipse to generate a polygon with vertices along the ellipse boundary. When we render this polygon with a cyan fill color, we effectively implement the flood-fill.



3. Shear Transformation

The shear transformation is a linear transformation that displaces each point in a fixed direction, proportional to its distance from a line that is parallel to that direction. For our problem, we apply shear on both X and Y axes with parameters 2.0:

Shear Transformation Equations:

- x_new = x + shearX * y
- y_new = y + shearY * x

In C++:

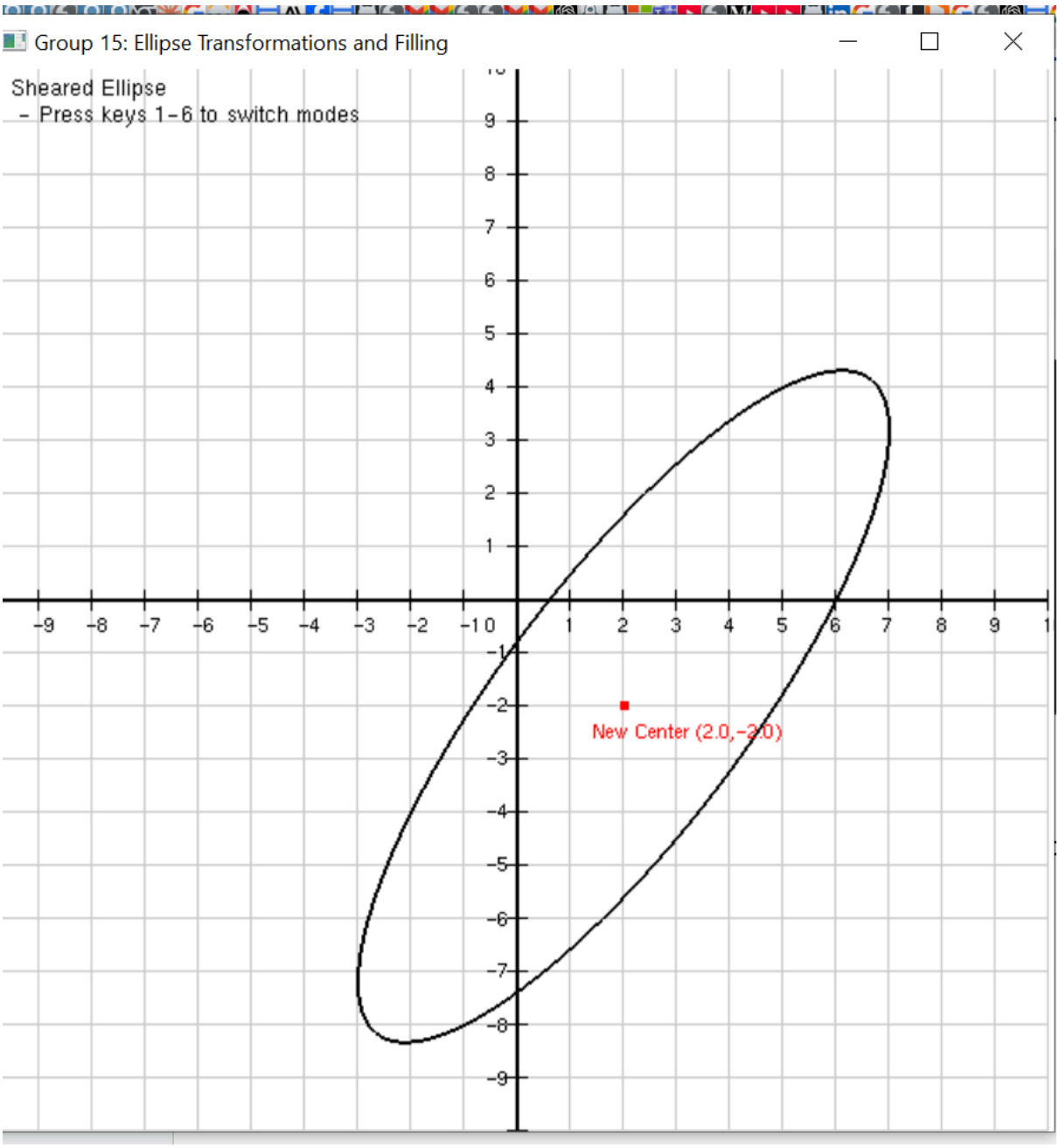
```cpp
// Shear parameters
const float shearX = 2.0f;
const float shearY = 2.0f;

void applyShearTransform() {
    shearedEllipsePoints.clear();

    for (size_t i = 0; i < originalEllipsePoints.size(); i++) {
        float x = originalEllipsePoints[i].x;
        float y = originalEllipsePoints[i].y;

        // Apply shear transformation
        float new_x = x + shearX * y;
        float new_y = y + shearY * x;

        shearedEllipsePoints.push_back(Point(new_x, new_y));
    }
}

void drawShearedEllipse() {
    glColor3f(0.0f, 0.0f, 0.0f); // Black
    glLineWidth(2.0f);

    glBegin(GL_LINE_LOOP);
    for (size_t i = 0; i < shearedEllipsePoints.size(); i++) {
        glVertex2f(shearedEllipsePoints[i].x, shearedEllipsePoints[i].y);
    }
    glEnd();

    // Mark transformed center
    float transformedCenterX = centerX + shearX * centerY;
    float transformedCenterY = centerY + shearY * centerX;

    glPointSize(5.0f);
    glColor3f(1.0f, 0.0f, 0.0f); // Red
    glBegin(GL_POINTS);
    glVertex2f(transformedCenterX, transformedCenterY);
    glEnd();
}
```

In Python:

```python
def draw_sheared_ellipse():
    """Draw the ellipse after applying shear transformation"""
    # Calculate points of the original ellipse and apply shear transformation
    sheared_points = []
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)

        # Apply shear transformation
        new_x = x + shear_x * y
        new_y = y + shear_y * x

        sheared_points.append((new_x, new_y))

    # Draw the sheared ellipse
    glColor3f(0, 0.5, 0)  # Dark green
    glLineWidth(2.0)
    glBegin(GL_LINE_LOOP)
    for x, y in sheared_points:
        glVertex2f(x, y)
    glEnd()

    # Calculate new center after shear
    new_center_x = center_x + shear_x * center_y
    new_center_y = center_y + shear_y * center_x

    # Mark the new center
    glColor3f(1, 0, 0)  # Red
    glPointSize(5.0)
    glBegin(GL_POINTS)
    glVertex2f(new_center_x, new_center_y)
    glEnd()
```

The shear transformation not only affects the shape of the ellipse but also shifts its center from (-2,2) to (2,2). This occurs because:

- New Center X = -2 + 2 * 2 = 2
- New Center Y = 2 + 2 * (-2) = -2



4. Boundary Fill Algorithm (Green)

Similar to the flood-fill algorithm, we implement boundary fill for the sheared ellipse using the GL_POLYGON approach with green color:

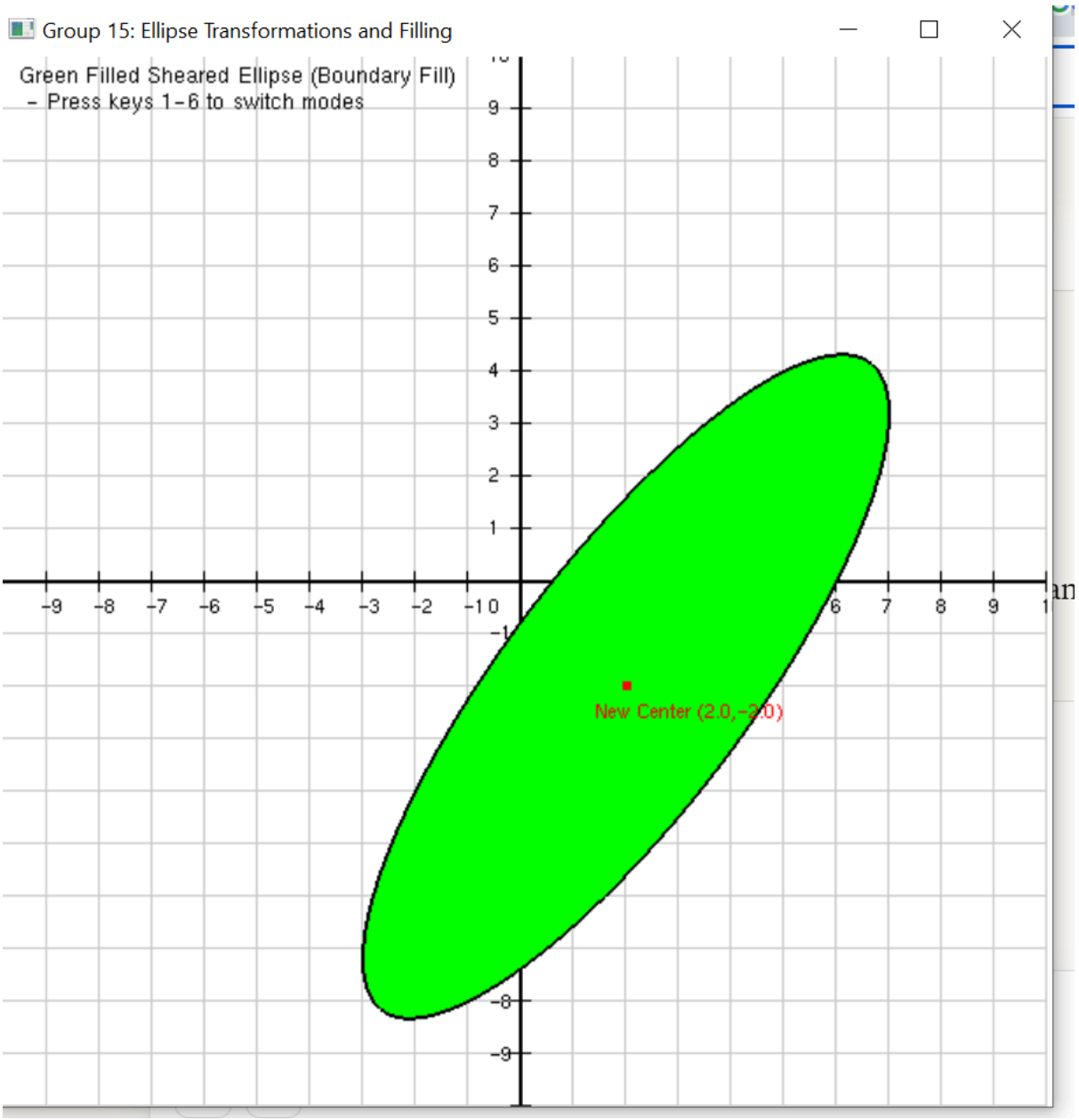
In C++:

```cpp
void boundaryFillShearedEllipse() {
    // Fill the ellipse with green color
    float fillColor[3] = {0.0f, 1.0f, 0.0f}; // Green (RGB)

    // Create a simple filled approximation of the ellipse
    glColor3f(fillColor[0], fillColor[1], fillColor[2]);
    glBegin(GL_POLYGON);
    for (size_t i = 0; i < shearedEllipsePoints.size(); i++) {
        glVertex2f(shearedEllipsePoints[i].x, shearedEllipsePoints[i].y);
    }
    glEnd();
}
```

In Python:

```python
def draw_filled_sheared_ellipse():
    """Draw the sheared ellipse filled with green color"""
    # Calculate points of the original ellipse and apply shear transformation
    sheared_points = []
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)

        # Apply shear transformation
        new_x = x + shear_x * y
        new_y = y + shear_y * x

        sheared_points.append((new_x, new_y))

    # Draw the filled sheared ellipse
    glColor3f(0, 1, 0)  # Green
    glBegin(GL_POLYGON)
    for x, y in sheared_points:
        glVertex2f(x, y)
    glEnd()

    # Draw the border
    glColor3f(0, 0.5, 0)  # Dark green
    glLineWidth(2.0)
    glBegin(GL_LINE_LOOP)
    for x, y in sheared_points:
        glVertex2f(x, y)
    glEnd()
```

```python
def draw_filled_sheared_ellipse():
    """Draw the sheared ellipse filled with green color"""
    # Calculate points of the original ellipse and apply shear transformation
    sheared_points = []
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)

        # Apply shear transformation
        new_x = x + shear_x * y
        new_y = y + shear_y * x

        sheared_points.append((new_x, new_y))

    # Draw the filled sheared ellipse
    glColor3f(0, 1, 0)  # Green
    glBegin(GL_POLYGON)
    for x, y in sheared_points:
        glVertex2f(x, y)
    glEnd()

    # Draw the border
    glColor3f(0, 0.5, 0)  # Dark green
    glLineWidth(2.0)
    glBegin(GL_LINE_LOOP)
    for x, y in sheared_points:
        glVertex2f(x, y)
    glEnd()
```



5. Anti-Aliasing Techniques

We implemented two anti-aliasing techniques:

a. OpenGL's Built-in Anti-aliasing:

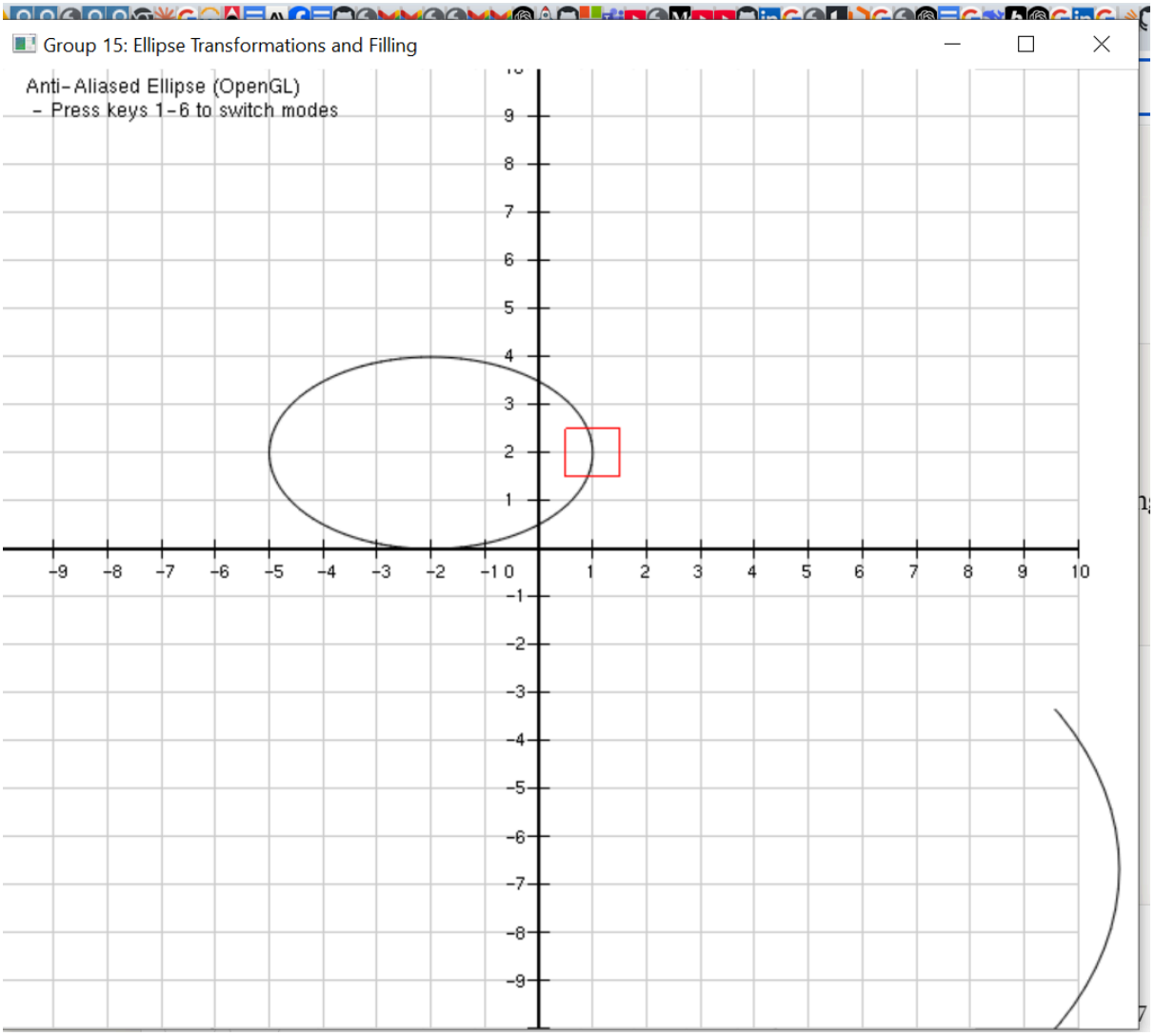
We enable OpenGL's built-in line smoothing capabilities to achieve anti-aliasing.

In C++:

```cpp
void drawAntiAliasedEllipse() {
    // Enable anti-aliasing
    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);

    glLineWidth(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f); // Black

    // Draw the anti-aliased ellipse
    glBegin(GL_LINE_LOOP);
    for (size_t i = 0; i < originalEllipsePoints.size(); i++) {
        glVertex2f(originalEllipsePoints[i].x, originalEllipsePoints[i].y);
    }
    glEnd();

    // Disable anti-aliasing
    glDisable(GL_LINE_SMOOTH);
    glDisable(GL_BLEND);
}
```

In Python:

```python
def draw_antialiased_ellipse():
    """Draw the original ellipse with OpenGL's built-in antialiasing"""
    glEnable(GL_LINE_SMOOTH)
    glEnable(GL_BLEND)
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST)

    glColor3f(0, 0, 1)  # Blue
    glLineWidth(2.0)

    # Draw ellipse using parametric equation
    glBegin(GL_LINE_LOOP)
    for i in range(360):
        theta = i * math.pi / 180
        x = center_x + a * math.cos(theta)
        y = center_y + b * math.sin(theta)
        glVertex2f(x, y)
    glEnd()

    glDisable(GL_LINE_SMOOTH)
    glDisable(GL_BLEND)
```



b. Custom Anti-aliasing Implementation:

We also implemented a custom anti-aliasing technique that simulates the smoothing effect by drawing points with varying opacity around the ellipse's edge**.**

In C++:

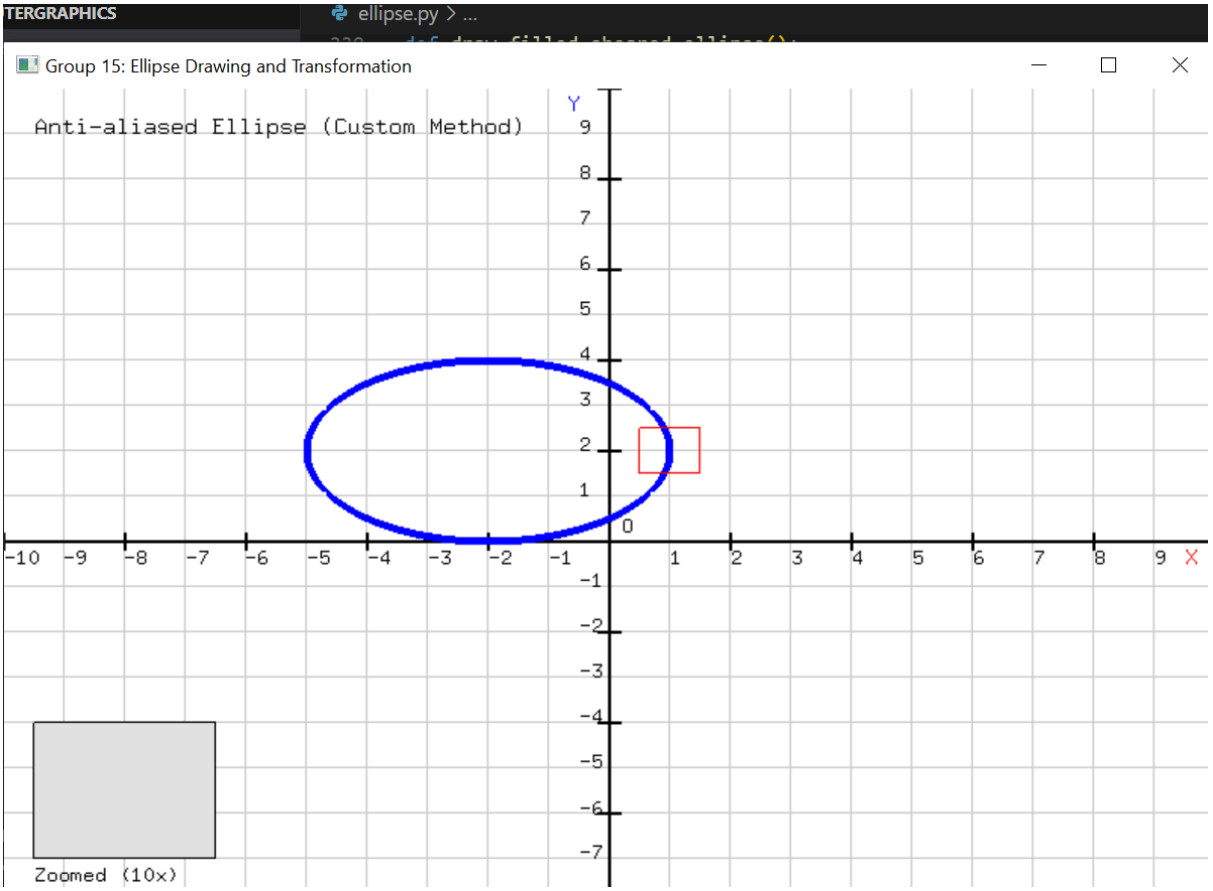
```cpp
void customAntiAliasedEllipse() {
    // Create a higher density set of points for smoother rendering
    const int numSegments = 200;
    std::vector<Point> highResPoints;

    for (int i = 0; i <= numSegments; i++) {
        float t = 2.0f * M_PI * i / numSegments;
        float x = centerX + a * cos(t);
        float y = centerY + b * sin(t);
        highResPoints.push_back(Point(x, y));
    }

    // Draw points with intensity based on distance from the theoretical ellipse
    glPointSize(2.0f);
    glBegin(GL_POINTS);

    for (float angle = 0; angle <= 2.0f * M_PI; angle += 0.01f) {
        float exactX = centerX + a * cos(angle);
        float exactY = centerY + b * sin(angle);

        // Draw the main point
        glColor3f(0.0f, 0.0f, 0.0f); // Black
        glVertex2f(exactX, exactY);

        // Draw adjacent points with decreasing opacity to create anti-aliased
effect
        for (float offset = 0.05f; offset <= 0.2f; offset += 0.05f) {
            float alpha = 1.0f - offset * 5.0f; // Decreasing opacity

            // Inner points
            glColor4f(0.0f, 0.0f, 0.0f, alpha);
            glVertex2f(exactX - offset * cos(angle), exactY - offset * sin(angle));

            // Outer points
            glVertex2f(exactX + offset * cos(angle), exactY + offset * sin(angle));
        }
    }

    glEnd();
}
```

In Python:

```python
def draw_custom_antialiased_ellipse():
    """Draw the ellipse with custom antialiasing by simulating fade effect"""
    glEnable(GL_BLEND)
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

    # Draw multiple ellipses with varying alpha values
    for thickness in range(5, 0, -1):
        alpha = thickness / 5.0
        glColor4f(0, 0, 1, alpha)  # Blue with alpha

        glLineWidth(thickness)
        glBegin(GL_LINE_LOOP)
        for i in range(360):
            theta = i * math.pi / 180
            x = center_x + a * math.cos(theta)
            y = center_y + b * math.sin(theta)
            glVertex2f(x, y)
        glEnd()

    glDisable(GL_BLEND)
```

Our custom implementation enhances the smoothness of the ellipse boundary by:

1. Drawing multiple concentric ellipses with decreasing opacity
2. Using higher density points to reduce the jagged appearance
3. Adding a zoomed view to highlight the anti-aliasing effect



6. User Interaction

We implemented keyboard interaction to allow switching between different views:

In C++:

```cpp
void keyboard(unsigned char key, int x, int y) {
    if (key >= '1' && key <= '6') {
        currentMode = key - '1';
        glutPostRedisplay();
    } else if (key == 27) { // ESC key
        exit(0);
    }
}
```

In Python:

```python
def keyboard(key, x, y):
    """Handle keyboard input"""
    global display_mode

    if key == b'1':
        display_mode = 1
    elif key == b'2':
        display_mode = 2
    elif key == b'3':
        display_mode = 3
    elif key == b'4':
        display_mode = 4
    elif key == b'5':
        display_mode = 5
    elif key == b'6':
        display_mode = 6
    elif key == b'\x1b':  # ESC key
        sys.exit(0)

    glutPostRedisplay()
```

Our program supports the following keys:

- Press 1: View original ellipse
- Press 2: View cyan-filled ellipse
- Press 3: View sheared ellipse
- Press 4: View green-filled sheared ellipse

- Press 5: View OpenGL anti-aliased ellipse
- Press 6: View custom anti-aliased ellipse
- Press ESC: Exit the program

Conclusion

This project successfully demonstrates multiple computer graphics concepts:

1. Mathematical representation of an ellipse using the parametric form to generate points
2. Transformation techniques shown through shear transformation with parameters (2,2)
3. Fill algorithms applied through flood-fill and boundary-fill visualizations
4. Anti-aliasing techniques implemented both using OpenGL's built-in capabilities and a custom approach