Lab Report 4

Title: To use Midpoints algorithm to draw a ellipse.

Theory: The Midpoint Ellipse Drawing Algorithm is crucial in computer graphics due to its ability to efficiently render ellipses using integer arithmetic, in contrast to methods like the Digital Differential Analyzer (DDA) which require floating-point calculations. This makes it ideal for devices with limited computing power, ensuring rapid execution, essential for realtime applications.

This algorithm operates by plotting pixels around the ellipse's perimeter simultaneously in all four quadrants, ensuring smooth and precise elliptical shapes. It employs decision-making to determine optimal pixel placements, adjusting them incrementally as it traverses the ellipse. This approach minimizes computational complexity while maintaining high accuracy, establishing it as a preferred choice for software and systems requiring fast and precise ellipse rendering capabilities.

Midpoint Algorithm (Ellipse):

Step 1: Start

Step 2: Initialize the midpoint ellipse algorithm with parameters: semi-major axis a, semiminor axis b, center coordinates (x_c, y_c)

Step 3: Set initial values, x = 0, y = b

Step 4: Initialize the decision parameter, p1 = b^2 - a^2*b + 0.25*a^2

Step 5: Draw an octant using a loop to generate points symmetrically around all octants of the ellipse

```
while (b^2*(x+1) < a^2*(y-0.5))
{     if (p1 < 0) {         p1 +=
b^2*(2*x + 3);
       x++;     } else {         p1 += b^2*(2*x
+ 3) + a^2*(-2*y + 2);         x++;         y--;
   }


   putpixel(x_c + x, y_c + y, WHITE); // Octant 1
putpixel(x_c - x, y_c + y, WHITE); // Octant 4
putpixel(x_c - x, y_c - y, WHITE); // Octant 5
putpixel(x_c + x, y_c - y, WHITE); // Octant 8 }
```

Step 6: Initialize another decision parameter, p2 = b^2*(x + 0.5)^2 + a^2*(y - 1)^2 - a^2*b^2 while (y > 0) {   if (p2 > 0) {   p2 += a^2*(-2*y + 3);

```
        y--;
    } else {
        p2 += b^2*(2*x + 2) + a^2*(-2*y +
    3);      x++;      y--;
      }

      putpixel(x_c + x, y_c + y, WHITE); // Octant 2
    putpixel(x_c - x, y_c + y, WHITE); // Octant 3
    putpixel(x_c - x, y_c - y, WHITE); // Octant 6
    putpixel(x_c + x, y_c - y, WHITE); // Octant 7
    }
```

Step 7: Stop

The code:

Using the application of Midpoint Algorithm, ellipse is drawn using the following code:

```
#include <iostream>
#include <graphics.h>
#include <conio.h>
#include <math.h>

class midpoint_algorithm { private:
   int a, b, xc, yc;

public:
   midpoint_algorithm(int a, int b, int xc, int yc) {
this->a = a;      this->b = b;      this->xc = xc;
this->yc = yc;
   }

   void generate() {
int x0 = 0;      int
y0 = b;
```

```
        float
p;

      // Region 1         p = pow(b, 2) - pow(a, 2) * b +
0.25 * pow(a, 2);

      int x = 0, y = b;         while (2 * pow(b, 2) * x
< 2 * pow(a, 2) * y) {            if (p < 0) {
x++;                p += 2 * pow(b, 2) * x + pow(b, 2);
          } else {
x++;                y--;
             p += 2 * pow(b, 2) * x - 2 * pow(a, 2) * y + pow(b, 2);
          }
          putpixel(x+xc, y+yc, WHITE);
putpixel(-x+xc, y+yc, WHITE);
putpixel(x+xc, -y+yc, WHITE);              putpixel(-
x+xc, -y+yc, WHITE);
      }

      // Region 2         p = pow(b, 2) * pow((x + 0.5), 2) + pow(a, 2) * pow((y - 1), 2) -
pow(a, 2) * pow(b, 2);

      while (y > 0) {
if (p > 0) {
y--;
             p += -2 * pow(a, 2) * y + pow(a, 2);
          } else {
x++;                y--;
             p += 2 * pow(b, 2) * x - 2 * pow(a, 2) * y + pow(a, 2);
          }
```

```
        putpixel(x+xc, y+yc, WHITE);
putpixel(-x+xc, y+yc, WHITE);
putpixel(x+xc, -y+yc, WHITE);            putpixel(-
x+xc, -y+yc, WHITE);
      }
   }
};

int main() {     int gd = DETECT, gm;
initwindow(800, 600, "Ellipse Drawing");
outtextxy(170,50, "Kshitiz Lab-4");
midpoint_algorithm ellipse(90, 60, 220,70);
ellipse.generate();

   getch();
closegraph();

   return 0;
}
```

The output:

The output of the asked program is given below:

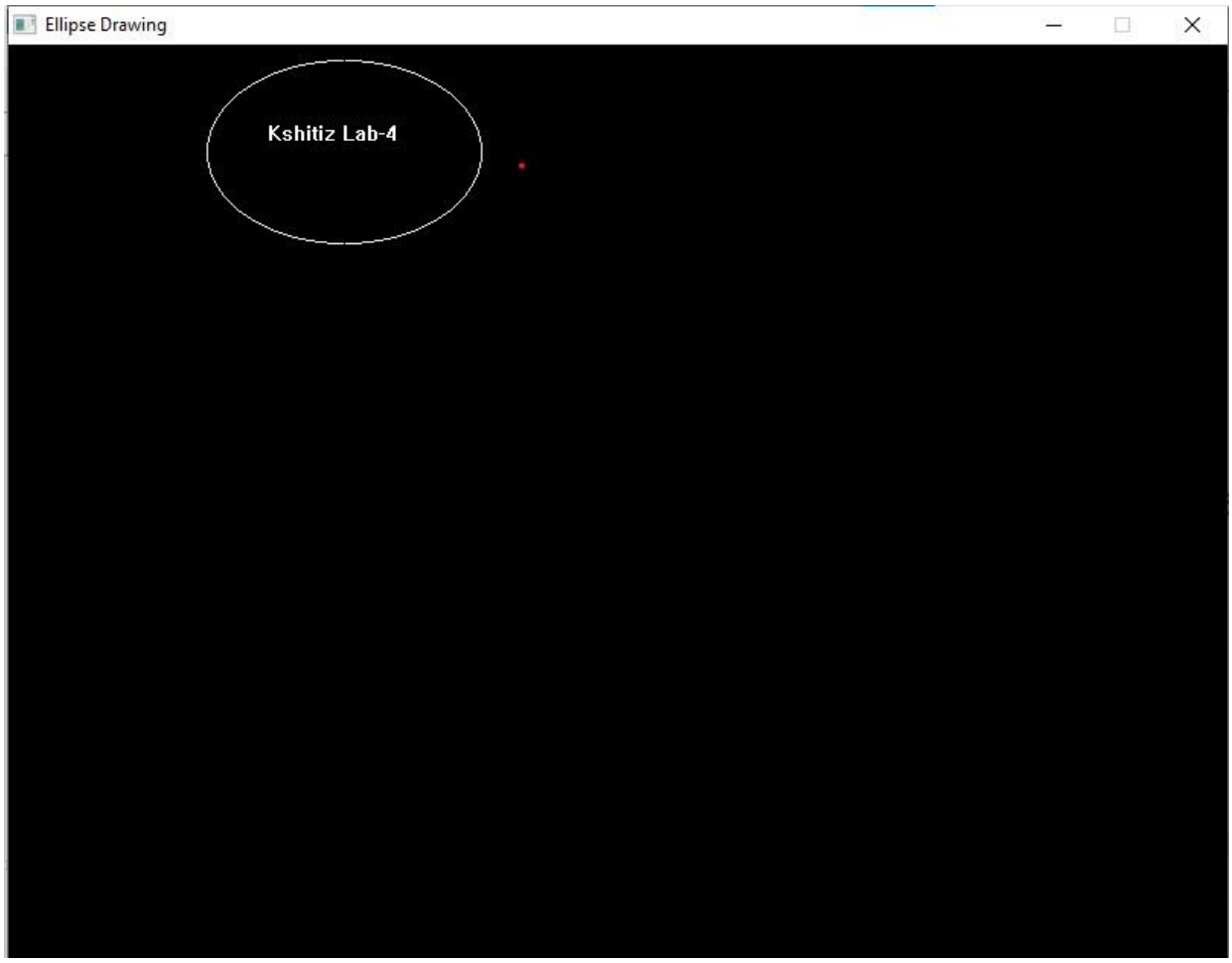

Figure : Application of Midpoint Algorithm in Ellipse Drawing

Conclusion:

The Midpoint Ellipse Drawing Algorithm efficiently renders ellipses using integer arithmetic, ensuring smooth and precise shapes across all octants. This method is ideal for real-time graphics applications due to its speed and accuracy compared to methods relying on floatingpoint operations.