Computer Graphics

Lab File

Name – Tushya Gupta
Roll number – UE233106
Group – 6

Index

Lab 1

Q1. Explain the graphics.h and the various functions inside of it

graphics.h is a legacy C header from Borland/Turbo C used with the Borland Graphics Interface (BGI). It provides simple 2-D drawing on DOS or DOS-style environments

Initialization

- initgraph(&driver, &mode, "path") - Loads a BGI driver and sets the graphics mode. Requires matching BGI driver files.

Drawing primitives

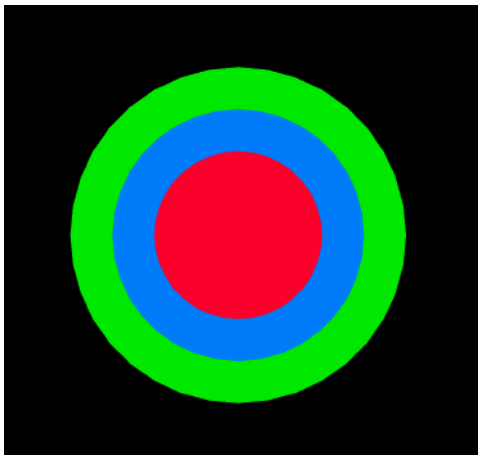
- line(x1, y1, x2, y2) – Draws a line from P(x1, y1) to Q(x2, y2)
- rectangle(left, top, right, bottom) – draws a rectangle from corner to corner
- circle(x, y, radius) – Draws a circle or radius at center O(x, y)
- ellipse(x, y, startAngle, endAngle, xRadius, yRadius) – Draws an ellipse from startAngle to endAngle having r_x and r_y with center O(x, y)
- arc(x, y, startAngle, endAngle, radius) – Draws a arc from startAngle to endAngle of radius r with midpoint(O(x, y)
- bar(left, top, right, bottom) – Draws a filled in rectangle
- bar3d(left, top, right, bottom, depth, topFlag) – Draws a 3D shaped bar with depth in the z axis and topFlag for showing the top face of the bar or not

Pixel operations

- putpixel(x, y, color) – put pixel of color at P(x, y)
- getpixel(x, y) – get the color of pixel at P(x, y)

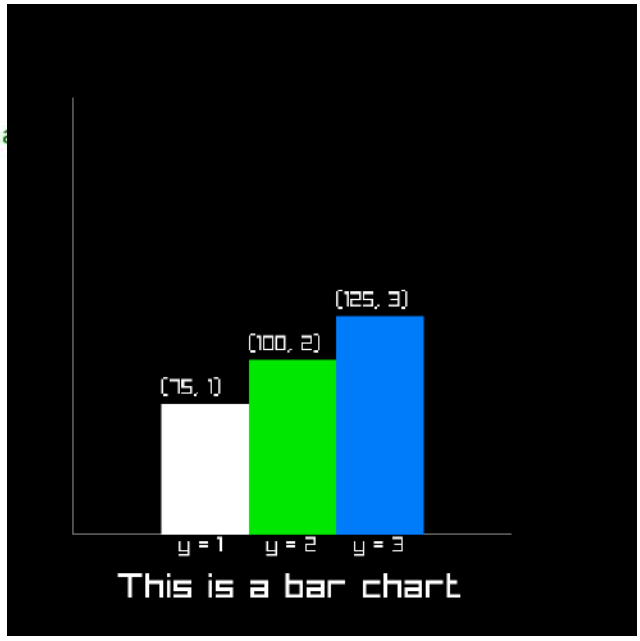
Q2. Draw concentric circles with different color of each of them
Code –

```c
#include "raylib.h"
#include "apps.h"

int concentric_circles() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "Hello Raylib");

  while (!WindowShouldClose()) {
    BeginDrawing();
    ClearBackground(BLACK);


    DrawCircle(400, 400, 200, GREEN);
    DrawCircle(400, 400, 150, BLUE);
    DrawCircle(400, 400, 100, RED);

    EndDrawing();
  }

  CloseWindow();
  return 0;
}
```



Q3. Draw a bar graph with text underneath
Code –

```c
#include "raylib.h"
#include "apps.h"

int bar_graph() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "Hello world");

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(BLACK);

    DrawLine(50, 400, 50, 150, WHITE);
    DrawLine(50, 400, 300, 400, WHITE);
    DrawRectangle(100, 325, 50, 75, WHITE);
    DrawText("(75, 1)", 100, 310, 12, WHITE);
    DrawText("y = 1", 110, 400, 12, WHITE);
    DrawRectangle(150, 300, 50, 100, GREEN);
    DrawText("(100, 2)", 150, 285, 12, WHITE);
    DrawText("y = 2", 160, 400, 12, WHITE);
    DrawRectangle(200, 275, 50, 125, BLUE);
    DrawText("(125, 3)", 200, 260, 12, WHITE);
    DrawText("y = 3", 210, 400, 12, WHITE);
    DrawText("This is a bar chart", 75, 420, 20, WHITE);

    EndDrawing();
  }

  // destroy the window and clea
  CloseWindow();
  return 0;
}
```



Lab 2

Q4. To draw a straight line between two points (x1, y1) and (x2, y2) using Simple Digit Differential Line Drawing Algorithm and Bresenham's Line Drawing Algorithm and display it on the screen.

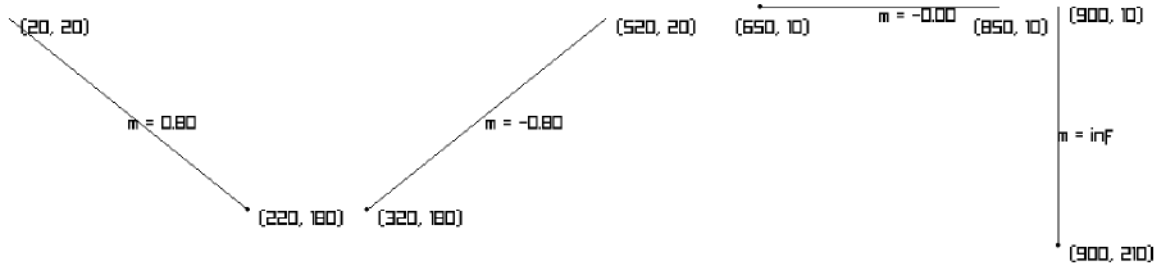
Code –

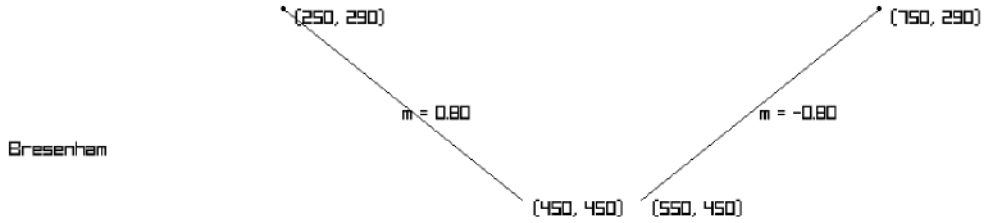
```
#include "apps.h"
#include "raylib.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

bool print_lda = true;

void incDDA(Vector2 init, Vector2 final, Color color) {
  int dy = final.y - init.y;
  int dx = final.x - init.x;
  float m = (float)dy / dx;
  float b = (float)init.y - m * init.x;

  float x = init.x, y = init.y;
  float step, xinc, yinc;
  step = fmax(abs(dy), abs(dx));
  xinc = dx / step;
  yinc = dy / step;
  if (print_lda) {
    printf("xinc=%f yinc=%f\n", xinc, yinc);
    printf("dx=%d dy=%d\n", dx, dy);
    printf("init.x=%f init.y=%f\n", init.x, init.y);
    printf("final.x=%f final.y=%f\n", final.x, final.y);
  }
  for (int i = 0; i <= step; i++) {
    if (print_lda) {
      printf("k = %d, x = %f, y = %f\n", i, x, y);
    }
    DrawPixel(round(x), round(y), color);
    y += yinc;
    x += xinc;
  }
  // print = false;
}

void bresenhamLDA(Vector2 init, Vector2 final, Color color) {
  int dy = final.y - init.y;
  int dx = final.x - init.x;
  int sx = dx < 0 ? -1 : 1;
  int sy = dy < 0 ? -1 : 1;
  float m = (float)dy / dx;
  dx = abs(dx);
  dy = abs(dy);
  int p;
  if (dx >= dy) {
    p = 2 * dy - dx;
  } else {
    p = 2 * dx - dy;
  }
```

```
  if (print_lda) {
    printf("m=%f\n", m);
    printf("dx=%d dy=%d\n", dx, dy);
    printf("p0=%d\n", p);
    printf("init.x=%f init.y=%f\n", init.x, init.y);
    printf("final.x=%f final.y=%f\n", final.x, final.y);
  }
  int x = init.x, y = init.y;
  if (m < 1 && m > -1) {
    for (int i = 0; i <= dx; i++) {
      if (print_lda) {
        printf("k = %d, x = %d, y = %d, p = %d\n", i, x, y, p);
      }
      DrawPixel(x, y, color);
      if (p < 0) {
        x += sx;
        p += 2 * dy;
      } else {
        x += sx;
        y += sy;
        p += 2 * dy - 2 * dx;
      }
    }
  } else {
    for (int i = 0; i <= dy; i++) {
      if (print_lda) {
        printf("k = %d, x = %d, y = %d, p = %d\n", i, x, y, p);
      }
      DrawPixel(x, y, color);
      if (p < 0) {
        y += sy;
        p += 2 * dx;
      } else {
        x += sx;
        y += sy;
        p += 2 * dx - 2 * dy;
      }
    }
  }
  // print = false;
}

void draw(Vector2 initial_point, Vector2 final_point, Color color) {
  // incDDA(initial_point, final_point, color);
  bresenhamLDA(initial_point, final_point, color);

  DrawCircleV(final_point, 2, color);
  DrawText(TextFormat("m = %.2f", (final_point.y - initial_point.y) /
                                      (final_point.x - initial_point.x)),
           (final_point.x + initial_point.x) / 2,
           (final_point.y + initial_point.y) / 2, 16, color);
}

int LDA() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "LDA");
```

```
// game loop
while (!WindowShouldClose()) {
  // drawing
  BeginDrawing();

  // Setup the back buffer for drawing (clear color and depth buffers)
  ClearBackground(WHITE);

  Vector2 initial_point = {
      .x = 20,
      .y = 20,
  };
  Vector2 final_point = {
      .x = 220,
      .y = 180,
  };
  if (print_lda) {
    printf("=== Line 1 ===\n");
  }
  draw(initial_point, final_point, BLACK);
  DrawText(TextFormat("(%.0f, %.0f)", initial_point.x, initial_point.y),
           initial_point.x+10, initial_point.y, 16, BLACK);
  DrawText(TextFormat("(%.0f, %.0f)", final_point.x, final_point.y),
           final_point.x+10, final_point.y, 16, BLACK);

  initial_point.x = 520;
  initial_point.y = 20;
  final_point.x = 320;
  final_point.y = 180;
  if (print_lda) {
    printf("=== Line 2 ===\n");
  }
  draw(initial_point, final_point, BLACK);
  DrawText(TextFormat("(%.0f, %.0f)", initial_point.x, initial_point.y),
           initial_point.x+10, initial_point.y, 16, BLACK);
  DrawText(TextFormat("(%.0f, %.0f)", final_point.x, final_point.y),
           final_point.x+10, final_point.y, 16, BLACK);

  initial_point.x = 850;
  initial_point.y = 10;
  final_point.x = 650;
  final_point.y = 10;
  if (print_lda) {
    printf("=== Line 3 ===\n");
  }
  draw(initial_point, final_point, BLACK);
  DrawText(TextFormat("(%.0f, %.0f)", initial_point.x, initial_point.y),
           initial_point.x-20, initial_point.y+10, 16, BLACK);
  DrawText(TextFormat("(%.0f, %.0f)", final_point.x, final_point.y),
           final_point.x-20, final_point.y+10, 16, BLACK);

  initial_point.x = 900;
  initial_point.y = 10;
  final_point.x = 900;
  final_point.y = 210;
  if (print_lda) {
    printf("=== Line 4 ===\n");
  }
  draw(initial_point, final_point, BLACK);
```

```
    DrawText(TextFormat("(%.0f, %.0f)", initial_point.x, initial_point.y),
             initial_point.x+10, initial_point.y, 16, BLACK);
    DrawText(TextFormat("(%.0f, %.0f)", final_point.x, final_point.y),
             final_point.x+10, final_point.y, 16, BLACK);

    initial_point.x = 450;
    initial_point.y = 450;
    final_point.x = 250;
    final_point.y = 290;
    if (print_lda) {
      printf("=== Line 5 ===\n");
    }
    draw(initial_point, final_point, BLACK);
    DrawText(TextFormat("(%.0f, %.0f)", initial_point.x, initial_point.y),
             initial_point.x+10, initial_point.y, 16, BLACK);
    DrawText(TextFormat("(%.0f, %.0f)", final_point.x, final_point.y),
             final_point.x+10, final_point.y, 16, BLACK);

    initial_point.x = 550;
    initial_point.y = 450;
    final_point.x = 750;
    final_point.y = 290;
    if (print_lda) {
      printf("=== Line 6 ===\n");
    }
    draw(initial_point, final_point, BLACK);
    DrawText(TextFormat("(%.0f, %.0f)", initial_point.x, initial_point.y),
             initial_point.x+10, initial_point.y, 16, BLACK);
    DrawText(TextFormat("(%.0f, %.0f)", final_point.x, final_point.y),
             final_point.x+10, final_point.y, 16, BLACK);

    DrawText("Bresenham", 20, 400, 16, BLACK);
    // DrawText("DDA", 20, 400, 16, BLACK);
    print_lda = false;

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```






```
=== Line 1 ===
m=0.800000
dx=200 dy=160
p0=120
init.x=20.000000 init.y=20.000000
final.x=220.000000 final.y=180.000000
k = 0, x = 20, y = 20, p = 120
k = 1, x = 21, y = 21, p = 40
k = 2, x = 22, y = 22, p = -40
k = 3, x = 23, y = 22, p = 280
k = 4, x = 24, y = 23, p = 200
k = 5, x = 25, y = 24, p = 120
k = 6, x = 26, y = 25, p = 40
k = 7, x = 27, y = 26, p = -40
k = 8, x = 28, y = 26, p = 280
k = 9, x = 29, y = 27, p = 200
k = 10, x = 30, y = 28, p = 120
k = 11, x = 31, y = 29, p = 40
k = 12, x = 32, y = 30, p = -40
k = 13, x = 33, y = 30, p = 280
k = 14, x = 34, y = 31, p = 200
```

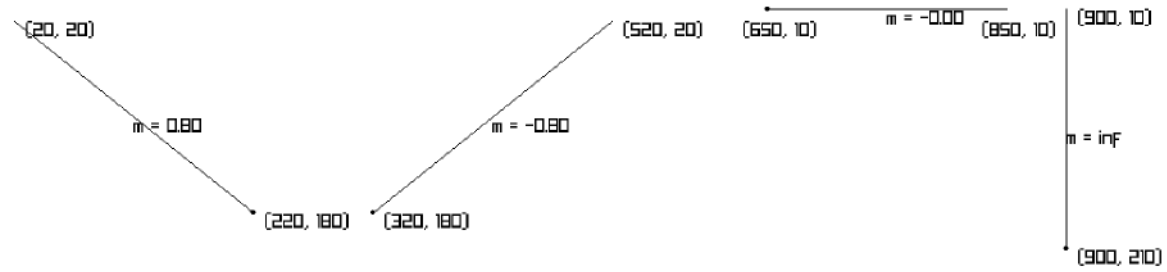
```
m=-0.800000
dx=200 dy=160
p0=120
init.x=520.000000 init.y=20.000000
final.x=320.000000 final.y=180.000000
k = 0, x = 520, y = 20, p = 120
k = 1, x = 519, y = 21, p = 40
k = 2, x = 518, y = 22, p = -40
k = 3, x = 517, y = 22, p = 280
k = 4, x = 516, y = 23, p = 200
k = 5, x = 515, y = 24, p = 120
k = 6, x = 514, y = 25, p = 40
k = 7, x = 513, y = 26, p = -40
k = 8, x = 512, y = 26, p = 280
k = 9, x = 511, y = 27, p = 200
k = 10, x = 510, y = 28, p = 120
k = 11, x = 509, y = 29, p = 40
k = 12, x = 508, y = 30, p = -40
k = 13, x = 507, y = 30, p = 280
k = 14, x = 506, y = 31, p = 200
```

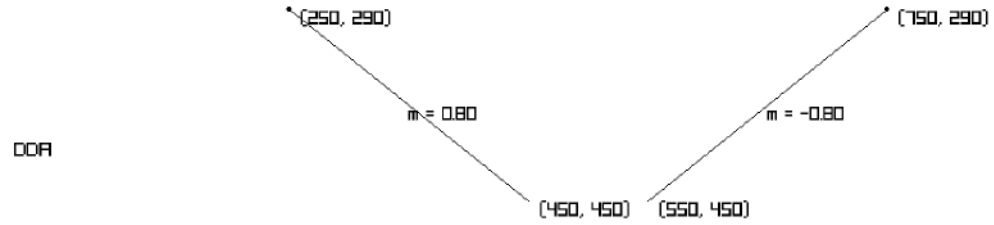
```
m=-0.000000
dx=200 dy=0
p0=-200
init.x=850.000000 init.y=10.000000
final.x=650.000000 final.y=10.000000
k = 0, x = 850, y = 10, p = -200
k = 1, x = 849, y = 10, p = -200
k = 2, x = 848, y = 10, p = -200
k = 3, x = 847, y = 10, p = -200
k = 4, x = 846, y = 10, p = -200
k = 5, x = 845, y = 10, p = -200
k = 6, x = 844, y = 10, p = -200
k = 7, x = 843, y = 10, p = -200
k = 8, x = 842, y = 10, p = -200
k = 9, x = 841, y = 10, p = -200
k = 10, x = 840, y = 10, p = -200
k = 11, x = 839, y = 10, p = -200
k = 12, x = 838, y = 10, p = -200
k = 13, x = 837, y = 10, p = -200
k = 14, x = 836, y = 10, p = -200
```

```
m=inf
dx=0 dy=200
p0=-200
init.x=900.000000 init.y=10.000000
final.x=900.000000 final.y=210.000000
k = 0, x = 900, y = 10, p = -200
k = 1, x = 900, y = 11, p = -200
k = 2, x = 900, y = 12, p = -200
k = 3, x = 900, y = 13, p = -200
k = 4, x = 900, y = 14, p = -200
k = 5, x = 900, y = 15, p = -200
k = 6, x = 900, y = 16, p = -200
k = 7, x = 900, y = 17, p = -200
k = 8, x = 900, y = 18, p = -200
k = 9, x = 900, y = 19, p = -200
k = 10, x = 900, y = 20, p = -200
k = 11, x = 900, y = 21, p = -200
k = 12, x = 900, y = 22, p = -200
k = 13, x = 900, y = 23, p = -200
k = 14, x = 900, y = 24, p = -200
```

```
m=0.800000
dx=200 dy=160
p0=120
init.x=450.000000 init.y=450.000000
final.x=250.000000 final.y=290.000000
k = 0, x = 450, y = 450, p = 120
k = 1, x = 449, y = 449, p = 40
k = 2, x = 448, y = 448, p = -40
k = 3, x = 447, y = 448, p = 280
k = 4, x = 446, y = 447, p = 200
k = 5, x = 445, y = 446, p = 120
k = 6, x = 444, y = 445, p = 40
k = 7, x = 443, y = 444, p = -40
k = 8, x = 442, y = 444, p = 280
k = 9, x = 441, y = 443, p = 200
k = 10, x = 440, y = 442, p = 120
k = 11, x = 439, y = 441, p = 40
k = 12, x = 438, y = 440, p = -40
k = 13, x = 437, y = 440, p = 280
k = 14, x = 436, y = 439, p = 200
```

```
m=-0.800000
dx=200 dy=160
p0=120
init.x=550.000000 init.y=450.000000
final.x=750.000000 final.y=290.000000
k = 0, x = 550, y = 450, p = 120
k = 1, x = 551, y = 449, p = 40
k = 2, x = 552, y = 448, p = -40
k = 3, x = 553, y = 448, p = 280
k = 4, x = 554, y = 447, p = 200
k = 5, x = 555, y = 446, p = 120
k = 6, x = 556, y = 445, p = 40
k = 7, x = 557, y = 444, p = -40
k = 8, x = 558, y = 444, p = 280
k = 9, x = 559, y = 443, p = 200
k = 10, x = 560, y = 442, p = 120
k = 11, x = 561, y = 441, p = 40
k = 12, x = 562, y = 440, p = -40
k = 13, x = 563, y = 440, p = 280
k = 14, x = 564, y = 439, p = 200
```






```
xinc=1.000000 yinc=0.800000
dx=200 dy=160
init.x=20.000000 init.y=20.000000
final.x=220.000000 final.y=180.000000
k = 0, x = 20.000000, y = 20.000000
k = 1, x = 21.000000, y = 20.799999
k = 2, x = 22.000000, y = 21.599998
k = 3, x = 23.000000, y = 22.399998
k = 4, x = 24.000000, y = 23.199997
k = 5, x = 25.000000, y = 23.999996
k = 6, x = 26.000000, y = 24.799995
k = 7, x = 27.000000, y = 25.599995
k = 8, x = 28.000000, y = 26.399994
k = 9, x = 29.000000, y = 27.199993
k = 10, x = 30.000000, y = 27.999992
k = 11, x = 31.000000, y = 28.799992
k = 12, x = 32.000000, y = 29.599991
k = 13, x = 33.000000, y = 30.399990
k = 14, x = 34.000000, y = 31.199989
```

```
xinc=-1.000000 yinc=0.800000
dx=-200 dy=160
init.x=520.000000 init.y=20.000000
final.x=320.000000 final.y=180.000000
k = 0, x = 520.000000, y = 20.000000
k = 1, x = 519.000000, y = 20.799999
k = 2, x = 518.000000, y = 21.599998
k = 3, x = 517.000000, y = 22.399998
k = 4, x = 516.000000, y = 23.199997
k = 5, x = 515.000000, y = 23.999996
k = 6, x = 514.000000, y = 24.799995
k = 7, x = 513.000000, y = 25.599995
k = 8, x = 512.000000, y = 26.399994
k = 9, x = 511.000000, y = 27.199993
k = 10, x = 510.000000, y = 27.999992
k = 11, x = 509.000000, y = 28.799992
k = 12, x = 508.000000, y = 29.599991
k = 13, x = 507.000000, y = 30.399990
k = 14, x = 506.000000, y = 31.199989
```

```
xinc=-1.000000 yinc=0.000000
dx=-200 dy=0
init.x=850.000000 init.y=10.000000
final.x=650.000000 final.y=10.000000
k = 0, x = 850.000000, y = 10.000000
k = 1, x = 849.000000, y = 10.000000
k = 2, x = 848.000000, y = 10.000000
k = 3, x = 847.000000, y = 10.000000
k = 4, x = 846.000000, y = 10.000000
k = 5, x = 845.000000, y = 10.000000
k = 6, x = 844.000000, y = 10.000000
k = 7, x = 843.000000, y = 10.000000
k = 8, x = 842.000000, y = 10.000000
k = 9, x = 841.000000, y = 10.000000
k = 10, x = 840.000000, y = 10.000000
k = 11, x = 839.000000, y = 10.000000
k = 12, x = 838.000000, y = 10.000000
k = 13, x = 837.000000, y = 10.000000
k = 14, x = 836.000000, y = 10.000000
```

```
xinc=0.000000 yinc=1.000000
dx=0 dy=200
init.x=900.000000 init.y=10.000000
final.x=900.000000 final.y=210.000000
k = 0, x = 900.000000, y = 10.000000
k = 1, x = 900.000000, y = 11.000000
k = 2, x = 900.000000, y = 12.000000
k = 3, x = 900.000000, y = 13.000000
k = 4, x = 900.000000, y = 14.000000
k = 5, x = 900.000000, y = 15.000000
k = 6, x = 900.000000, y = 16.000000
k = 7, x = 900.000000, y = 17.000000
k = 8, x = 900.000000, y = 18.000000
k = 9, x = 900.000000, y = 19.000000
k = 10, x = 900.000000, y = 20.000000
k = 11, x = 900.000000, y = 21.000000
k = 12, x = 900.000000, y = 22.000000
k = 13, x = 900.000000, y = 23.000000
k = 14, x = 900.000000, y = 24.000000
```

```
xinc=-1.000000 yinc=-0.800000
dx=-200 dy=-160
init.x=450.000000 init.y=450.000000
final.x=250.000000 final.y=290.000000
k = 0, x = 450.000000, y = 450.000000
k = 1, x = 449.000000, y = 449.200012
k = 2, x = 448.000000, y = 448.400024
k = 3, x = 447.000000, y = 447.600037
k = 4, x = 446.000000, y = 446.800049
k = 5, x = 445.000000, y = 446.000061
k = 6, x = 444.000000, y = 445.200073
k = 7, x = 443.000000, y = 444.400085
k = 8, x = 442.000000, y = 443.600098
k = 9, x = 441.000000, y = 442.800110
k = 10, x = 440.000000, y = 442.000122
k = 11, x = 439.000000, y = 441.200134
k = 12, x = 438.000000, y = 440.400146
k = 13, x = 437.000000, y = 439.600159
k = 14, x = 436.000000, y = 438.800171
```

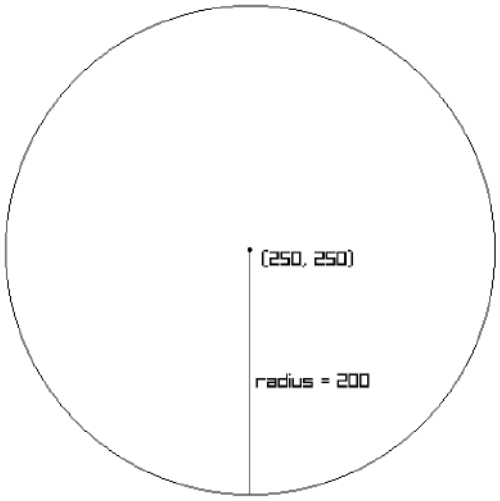
```
xinc=1.000000 yinc=-0.800000
dx=200 dy=-160
init.x=550.000000 init.y=450.000000
final.x=750.000000 final.y=290.000000
k = 0, x = 550.000000, y = 450.000000
k = 1, x = 551.000000, y = 449.200012
k = 2, x = 552.000000, y = 448.400024
k = 3, x = 553.000000, y = 447.600037
k = 4, x = 554.000000, y = 446.800049
k = 5, x = 555.000000, y = 446.000061
k = 6, x = 556.000000, y = 445.200073
k = 7, x = 557.000000, y = 444.400085
k = 8, x = 558.000000, y = 443.600098
k = 9, x = 559.000000, y = 442.800110
k = 10, x = 560.000000, y = 442.000122
k = 11, x = 561.000000, y = 441.200134
k = 12, x = 562.000000, y = 440.400146
k = 13, x = 563.000000, y = 439.600159
k = 14, x = 564.000000, y = 438.800171
```

Lab 3

Q5. Draw a circle using the midpoint circle algorithm
Code –

```c
#include "apps.h"
#include "raylib.h"
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

void draw_all_sides(int x, int y, Vector2 m) {
  DrawPixelV((Vector2){.x = x + m.x, .y = y + m.y}, BLACK);
  DrawPixelV((Vector2){.x = y + m.x, .y = x + m.y}, BLACK);
  DrawPixelV((Vector2){.x = -x + m.x, .y = y + m.y}, BLACK);
  DrawPixelV((Vector2){.x = -y + m.x, .y = x + m.y}, BLACK);
  DrawPixelV((Vector2){.x = -x + m.x, .y = -y + m.y}, BLACK);
  DrawPixelV((Vector2){.x = -y + m.x, .y = -x + m.y}, BLACK);
  DrawPixelV((Vector2){.x = x + m.x, .y = -y + m.y}, BLACK);
  DrawPixelV((Vector2){.x = y + m.x, .y = -x + m.y}, BLACK);
}

bool print_midpoint_circle = true;

void calc_circle(Vector2 midpoint, uint32_t r) {
  int x = 0, y = r;
  int di = 1 - r;
  draw_all_sides(x, y, midpoint);
  while (x < y) {
    if (print_midpoint_circle) {
      printf("x = %d, y = %d, d = %d\n", x, y, di);
    }
    if (di < 0) {
      di = di + 2 * x + 3;
      x += 1;
    } else {
      di = di + 2 * x - 2 * y + 5;
      x += 1;
      y -= 1;
    }
    draw_all_sides(x, y, midpoint);
  }
  print_midpoint_circle = false;
}

int midpointCircle() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "midpointCircle");

  // game loop
  while (!WindowShouldClose()) {
    // drawing
```

```
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(WHITE);

    Vector2 center = {.x = 250, .y = 250};
    int radius = 200;
    calc_circle(center, radius);
    DrawLineV(center, (Vector2){.x = center.x, .y = center.y + radius}, BLACK);
    DrawCircleV(center, 2, BLACK);
    DrawText(TextFormat("radius = %i", radius), center.x + 5,
             center.y + ((float)radius / 2), 16, BLACK);
    DrawText(TextFormat("(%.0f, %.0f)", center.x, center.y), center.x + 10,
             center.y, 16, BLACK);

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```




```
radius = 20
center = (250, 250)
d = -19
x = 0, y = 20, d = -19
x = 1, y = 20, d = -16
x = 2, y = 20, d = -11
x = 3, y = 20, d = -4
x = 4, y = 20, d = 5
x = 5, y = 19, d = -22
x = 6, y = 19, d = -9
x = 7, y = 19, d = 6
x = 8, y = 18, d = -13
x = 9, y = 18, d = 6
x = 10, y = 17, d = -7
x = 11, y = 17, d = 16
x = 12, y = 16, d = 9
x = 13, y = 15, d = 6
```

Lab 4

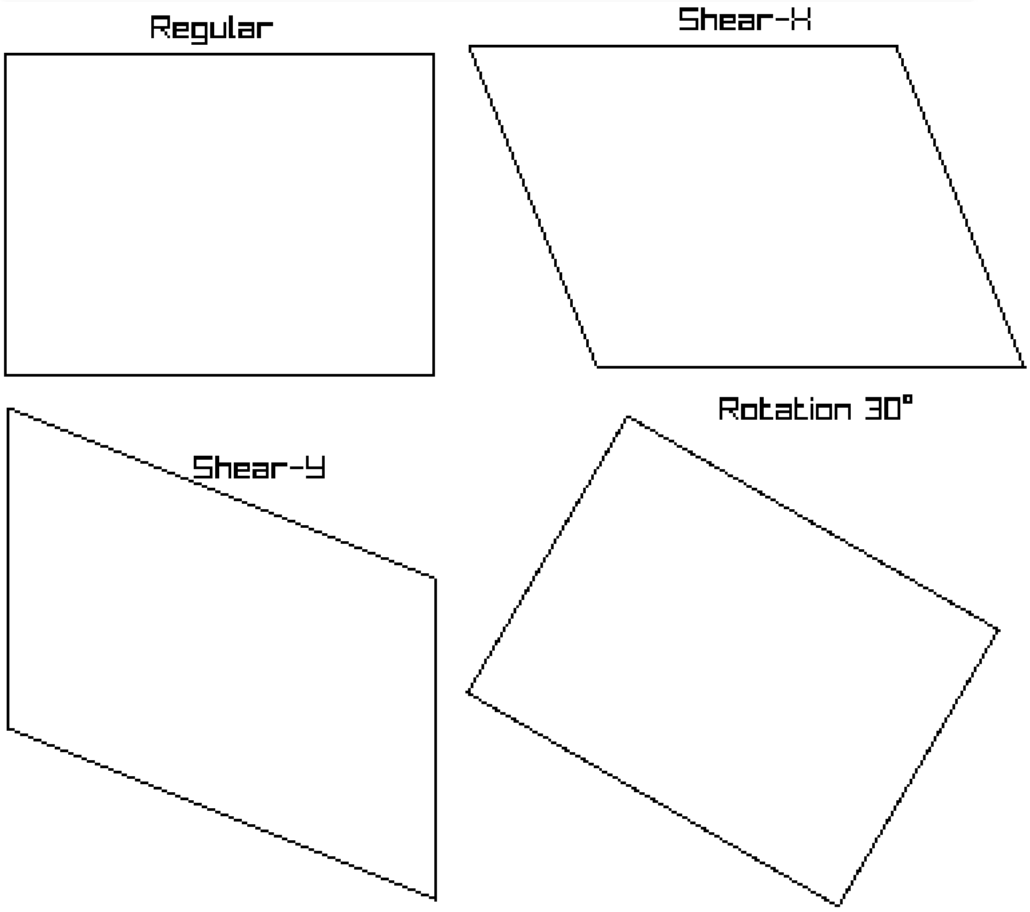
Q6. Apply all basic transformations on a rectangle.
Code –

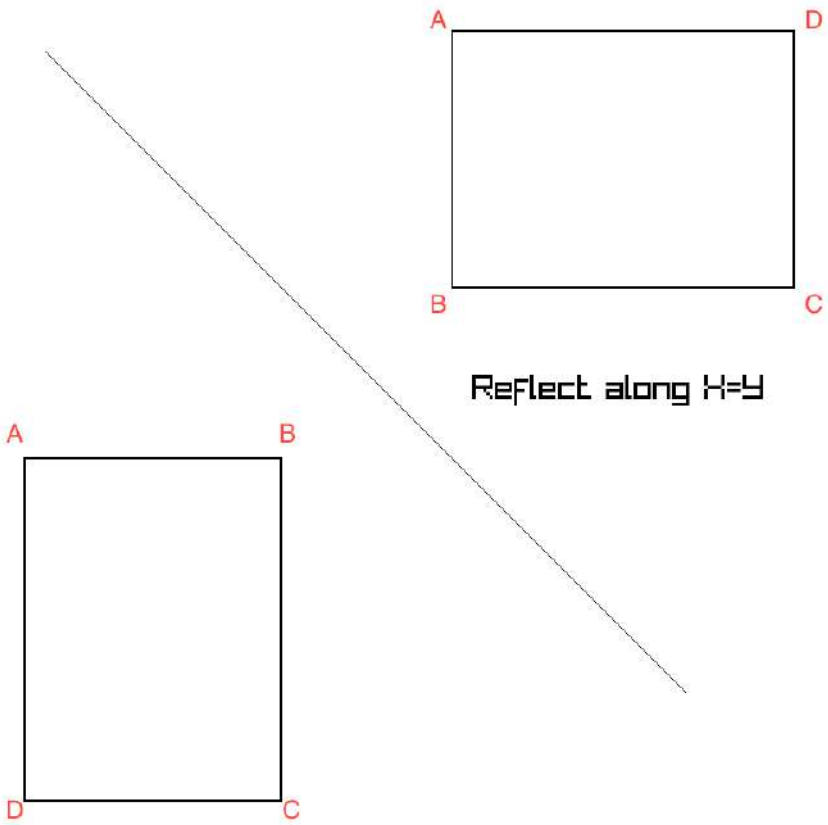
```
#include "apps.h"
#include "raylib.h"
#include <cmath>
#include <vector>
using namespace std;

#define pi 3.14

vector<vector<float>> mul(vector<vector<float>> a, vector<vector<float>> b) {
  vector<vector<float>> points(a.size(), vector<float>(3, 0));
  for (int i = 0; i < a.size(); i++) {
    for (int j = 0; j < b[0].size(); j++) {
      points[i][j] = 0;
      for (int k = 0; k < b.size(); k++) {
        points[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return points;
}

vector<vector<float>> translate(float sx, float sy) {
  vector<vector<float>> mat = {{1, 0, 0}, {0, 1, 0}, {sx, sy, 1}};
  return mat;
}

vector<vector<float>> reflect(string wrt) {
  vector<vector<float>> mat = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
  if (wrt == "x") {
    mat = {{1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
  } else if (wrt == "y") {
    mat = {{-1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
  } else if (wrt == "xy") {
    mat = {{-1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
  } else if (wrt == "x=y") {
    mat = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}};
  }
  return mat;
}

vector<vector<float>> shear(float sx, float sy) {
  vector<vector<float>> mat = {{1, sy, 0}, {sx, 1, 0}, {0, 0, 1}};
  return mat;
}

vector<vector<float>> scale(float sx, float sy) {
  vector<vector<float>> mat = {{sx, 0, 0}, {0, sy, 0}, {0, 0, 1}};
  return mat;
}
```

```
vector<vector<float>> rotatation(int angle) {
  float c = cos(angle * pi / 180);
  float s = sin(angle * pi / 180);
  vector<vector<float>> mat = {{c, s, 0}, {-s, c, 0}, {0, 0, 1}};
  return mat;
}

void drawRect(vector<vector<float>> &points, vector<float> p, vector<float> q) {
  for (float i = p[0]; i <= q[0]; i++) {
    points.push_back({i, p[1], 1});
    points.push_back({i, q[1], 1});
  }
  for (float i = p[1]; i <= q[1]; i++) {
    points.push_back({p[0], i, 1});
    points.push_back({q[0], i, 1});
  }
}

void drawPoints(vector<vector<float>> points) {
  for (auto p : points) {
    DrawPixel(p[0], p[1], WHITE);
  }
}

int transformations() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "transformations");

  vector<vector<float>> points;
  vector<float> start = {40, 40, 1};
  vector<float> end = {200, 160, 1};
  drawRect(points, start, end);

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(BLACK);

    DrawText("Regular", 95, 25, 12, WHITE);
    drawPoints(points);
    DrawText("Shear-X", 295, 25, 12, WHITE);
    drawPoints(
      mul(
        points,
        mul(
          mul(
            translate(-((float)start[0]+end[0])/2, -((float)start[1]+end[1])/2),
            mul(
              shear(0.4, 0),
              translate(((float)start[0]+end[0])/2, ((float)start[1]+end[1])/2)
            )
          ),
          translate(200, 0)
        )
```

```
    )
  );
  DrawText("Shear-Y", 560, 25, 12, WHITE);
  drawPoints(
    mul(
      points,
      mul(
        mul(
          translate(-((float)start[0]+end[0])/2, -((float)start[1]+end[1])/2),
          mul(
            shear(0, 0.4),
            translate(((float)start[0]+end[0])/2, ((float)start[1]+end[1])/2)
          )
        ),
        translate(450, 0)
      )
    )
  );
  DrawText("Rotation", 120, 250, 12, WHITE);
  drawPoints(
    mul(
      points,
      mul(
        mul(
          translate(-((float)start[0]+end[0])/2, -((float)start[1]+end[1])/2),
          mul(
            rotatation(30),
            translate(((float)start[0]+end[0])/2, ((float)start[1]+end[1])/2)
          )
        ),
        translate(0, 250)
      )
    )
  );
  DrawText("Scaling", 745, 300, 12, WHITE);
  drawPoints(
    mul(
      points,
      mul(
        mul(
          translate(-((float)start[0]+end[0])/2, -((float)start[1]+end[1])/2),
          mul(
            scale(0.5, 0.5),
            translate(((float)start[0]+end[0])/2, ((float)start[1]+end[1])/2)
          )
        ),
        translate(650, 250)
      )
    )
  );
  DrawText("Reflect along X", 75, 525, 12, WHITE);
  drawPoints(
    mul(
      points,
      mul(
        translate(0, -700),
        reflect("x")
      )
    )
  );
```

```
    drawPoints(
      mul(
        points,
          translate(450, 250)
      )
    );
    DrawText("Reflect along X=Y", 500, 450, 16, WHITE);
    DrawLine(300, 300, 600, 600, WHITE);
    drawPoints(
      mul(
        points,
        mul(
          translate(450, 250),
          reflect("x=y")
        )
      )
    );

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```





Regular

Scaling

Lab 5

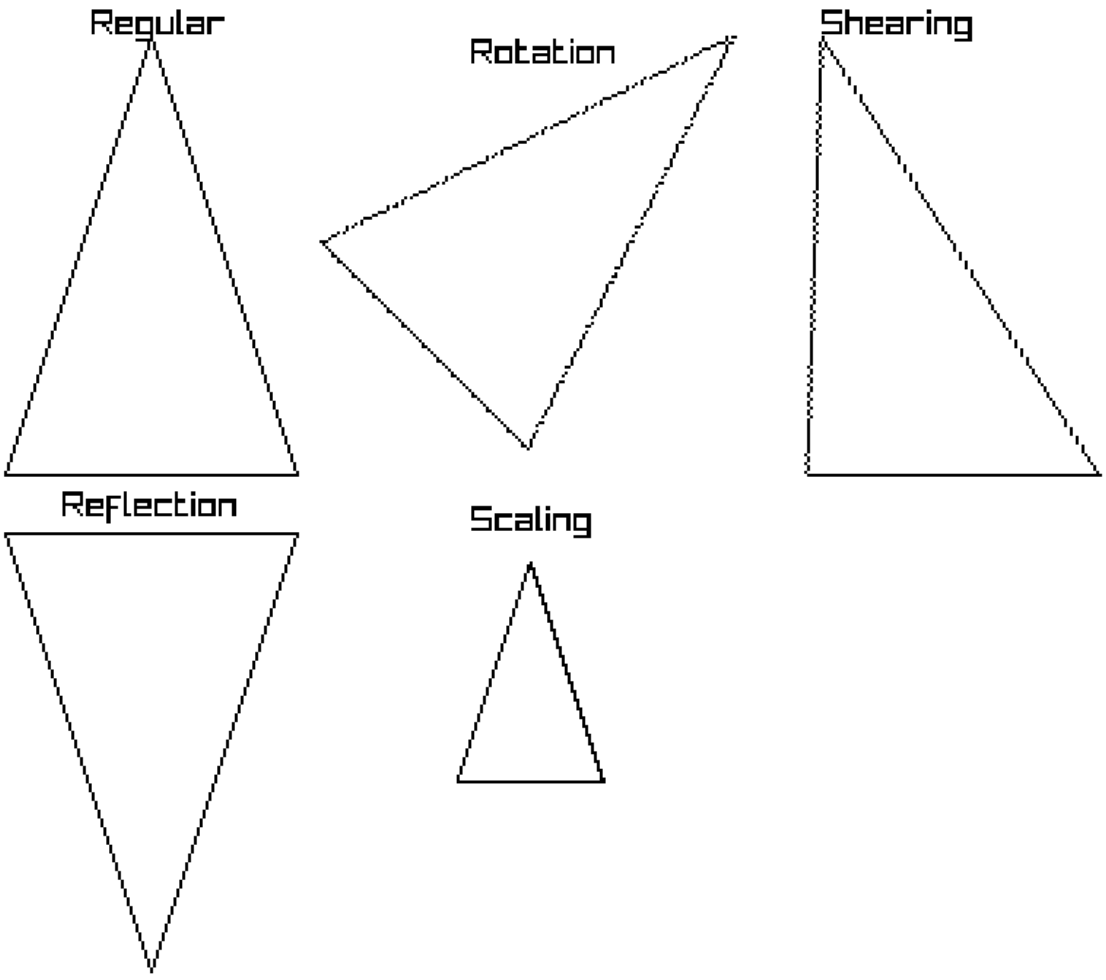
Q7. Apply all composite transformations on a trianlge.
Code –

```
#include "apps.h"
#include "raylib.h"
#include <cmath>
#include <vector>
using namespace std;

#define pi 3.14

static vector<vector<float>> mul(vector<vector<float>> a, vector<vector<float>> b) {
  vector<vector<float>> points(a.size(), vector<float>(3, 0));
  for (int i = 0; i < a.size(); i++) {
    for (int j = 0; j < b[0].size(); j++) {
      points[i][j] = 0;
      for (int k = 0; k < b.size(); k++) {
        points[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return points;
}

static vector<vector<float>> translate(float sx, float sy) {
  vector<vector<float>> mat = {{1, 0, 0}, {0, 1, 0}, {sx, sy, 1}};
  return mat;
}

static vector<vector<float>> reflect(string wrt) {
  vector<vector<float>> mat = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
  if (wrt == "x") {
    mat = {{1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
  } else if (wrt == "y") {
    mat = {{-1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
  } else if (wrt == "xy") {
    mat = {{-1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
  } else if (wrt == "x=y") {
    mat = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}};
  }
  return mat;
}

static vector<vector<float>> shear(float sx, float sy) {
  vector<vector<float>> mat = {{1, sy, 0}, {sx, 1, 0}, {0, 0, 1}};
  return mat;
}

static vector<vector<float>> scale(float sx, float sy) {
  vector<vector<float>> mat = {{sx, 0, 0}, {0, sy, 0}, {0, 0, 1}};
  return mat;
}
```

```
static vector<vector<float>> rotatation(int angle) {
  float c = cos(angle * pi / 180);
  float s = sin(angle * pi / 180);
  vector<vector<float>> mat = {{c, s, 0}, {-s, c, 0}, {0, 0, 1}};
  return mat;
}

static void bresenhamLDA(vector<vector<float>> &points, Vector2 init, Vector2 final) {
  int dy = final.y - init.y;
  int dx = final.x - init.x;
  int sx = dx < 0 ? -1 : 1;
  int sy = dy < 0 ? -1 : 1;
  float m = (float)dy / dx;
  dx = abs(dx);
  dy = abs(dy);
  int p;
  if (dx >= dy) {
    p = 2 * dy - dx;
  } else {
    p = 2 * dx - dy;
  }

  float x = init.x, y = init.y;
  if (m < 1 && m > -1) {
    for (int i = 0; i <= dx; i++) {
      points.push_back({x, y, 1});
      if (p < 0) {
        x += sx;
        p += 2 * dy;
      } else {
        x += sx;
        y += sy;
        p += 2 * dy - 2 * dx;
      }
    }
  } else {
    for (int i = 0; i <= dy; i++) {
      points.push_back({x, y, 1});
      if (p < 0) {
        y += sy;
        p += 2 * dx;
      } else {
        x += sx;
        y += sy;
        p += 2 * dx - 2 * dy;
      }
    }
  }
}

static void drawTri(vector<vector<float>> &points) {
  bresenhamLDA(points, (Vector2){.x = 20, .y = 200},
               (Vector2){.x = 120, .y = 200});
  bresenhamLDA(points, (Vector2){.x = 120, .y = 200},
               (Vector2){.x = 70, .y = 50});
  bresenhamLDA(points, (Vector2){.x = 70, .y = 50},
               (Vector2){.x = 20, .y = 200});
}

static void drawPoints(vector<vector<float>> points) {
```

```
  for (auto p : points) {
    DrawPixel(p[0], p[1], WHITE);
  }
}

int compo_transformations() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "transformations");

  vector<vector<float>> points;
  drawTri(points);

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(BLACK);

    DrawText("Regular", 50, 40, 12, WHITE);
    drawPoints(points);
    DrawText("Rotation", 180, 50, 12, WHITE);
    drawPoints(
      mul(
        points,
        mul(
          translate(-70, -50),
          mul(
            rotatation(45),
            translate(270, 50)
          )
        )
      )
    );
    DrawText("Shearing", 300, 40, 12, WHITE);
    drawPoints(
      mul(
        points,
        mul(
          translate(-70, -50),
          mul(
            shear(0.3, 0),
            translate(300, 50)
          )
        )
      )
    );
    DrawText("Reflection", 40, 205, 12, WHITE);
    drawPoints(
      mul(
        points,
        mul(
          translate(0, -300),
          mul(
            reflect("x"),
            translate(0, 120)
```

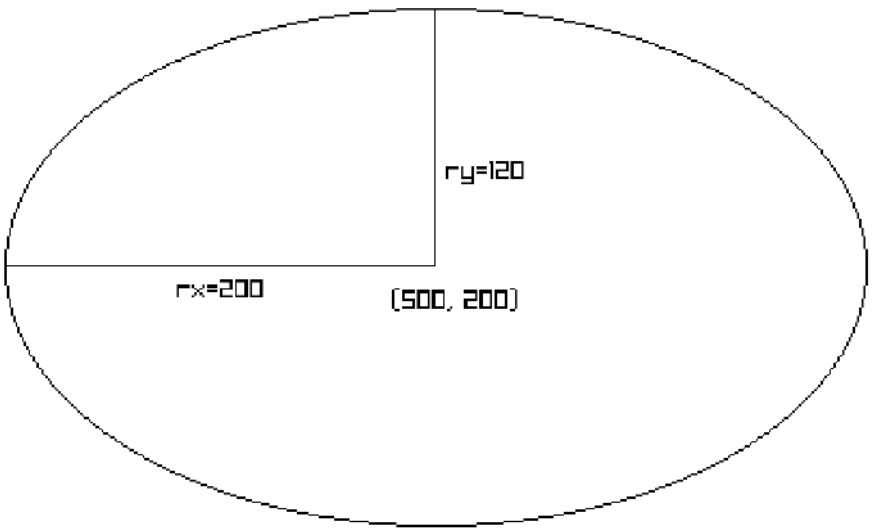
```
          )
        )
      )
    );
    DrawText("Reflection", 170, 210, 12, WHITE);
    drawPoints(
      mul(
        points,
        mul(
          translate(-70, -50),
          mul(
            scale(0.5, 0.5),
            translate(200, 230)
          )
        )
      )
    );

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```



Q8. Draw an ellipse using the midpoint ellipse algorithm.
Code –

```
#include "apps.h"
#include "raylib.h"
void draw_points(int x, int y, Vector2 m) {
  DrawPixelV((Vector2){.x = x + m.x, .y = y + m.y}, WHITE);
  DrawPixelV((Vector2){.x = -x + m.x, .y = y + m.y}, WHITE);
  DrawPixelV((Vector2){.x = -x + m.x, .y = -y + m.y}, WHITE);
  DrawPixelV((Vector2){.x = x + m.x, .y = -y + m.y}, WHITE);
}

void calc_ellipse(Vector2 origin, int rx, int ry) {
  int x = 0, y = ry;
  float p1i = ry * ry - rx * rx * ry + (float)rx * rx / 4;
  int dx = 2 * ry * ry * x;
  int dy = 2 * rx * rx * y;
  while (dx < dy) {
    draw_points(x, y, origin);
    if (p1i < 0) {
      x += 1;
      dx = 2 * ry * ry * x;
      p1i = p1i + dx + ry * ry;
    } else {
      x += 1;
      y -= 1;
      dx = 2 * ry * ry * x;
      dy = 2 * rx * rx * y;
      p1i = p1i + dx - dy + ry * ry;
    }
  }
  float p2i;
  p2i = ry * ry * ((float)x + (float)1 / 2) * ((float)x + (float)1 / 2) +
        rx * rx * (y - 1) * (y - 1) - rx * rx * ry * ry;
  while (dx >= dy && y >= 0) {
    draw_points(x, y, origin);
    if (p2i > 0) {
      y -= 1;
      dy = 2 * rx * rx * y;
      p2i = p2i - dy + rx * rx;
    } else {
      x += 1;
      y -= 1;
      dy = 2 * rx * rx * y;
      dx = 2 * ry * ry * x;
      p2i = p2i + dx - dy + rx * rx;
    }
  }
}

int midpointEllipse() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);
```

```
  // Create the window and OpenGL context
  InitWindow(1280, 800, "midpointEllipse");

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(BLACK);
    calc_ellipse((Vector2){.x = 500, .y = 200}, 200, 120);
    DrawLine(500, 200, 300, 200, WHITE);
    DrawLine(500, 200, 500, 80, WHITE);
    DrawText("(500, 200)", 480, 210, 12, WHITE);
    DrawText("ry=120", 505, 150, 12, WHITE);
    DrawText("rx=200", 380, 205, 12, WHITE);

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```



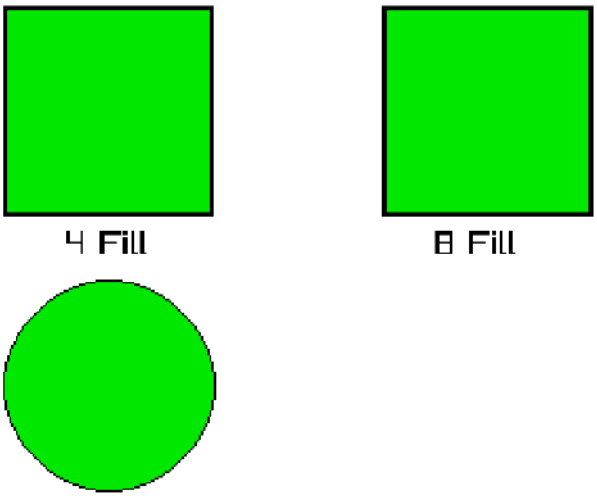

```
x = 0, y = 12
x = 1, y = 12
x = 2, y = 12
x = 3, y = 12
x = 4, y = 12
x = 5, y = 12
x = 6, y = 11
x = 7, y = 11
x = 8, y = 11
x = 9, y = 11
x = 10, y = 10
x = 11, y = 10
x = 12, y = 10
x = 13, y = 9
x = 14, y = 9
x = 15, y = 8
x = 16, y = 7
x = 17, y = 6
x = 18, y = 5
x = 19, y = 4
x = 19, y = 3
x = 20, y = 2
x = 20, y = 1
x = 20, y = 0
```

Lab 6

Q9. Implement the boundary fill algorithm.
Code –

```
#include "apps.h"
#include "raylib.h"

bool operator!=(Color a, Color b) {
  return !(a.a == b.a && a.b == b.b && a.r == b.r && a.g == b.g);
}

void fill_4(Image *img, Vector2 seed, Color boundary_color, Color fill_color) {
  // printf("%d %d %d\n", color.r, color.g, color.b);
  Color c = GetImageColor(*img, seed.x, seed.y);
  if (c != boundary_color && c != fill_color) {
    ImageDrawPixel(img, seed.x, seed.y, fill_color);
    fill_4(img, (Vector2){.x = seed.x + 1, .y = seed.y}, boundary_color,
           fill_color);
    fill_4(img, (Vector2){.x = seed.x, .y = seed.y + 1}, boundary_color,
           fill_color);
    fill_4(img, (Vector2){.x = seed.x - 1, .y = seed.y}, boundary_color,
           fill_color);
    fill_4(img, (Vector2){.x = seed.x, .y = seed.y - 1}, boundary_color,
           fill_color);
  }
}

void fill_8(Image *img, Vector2 seed, Color boundary_color, Color fill_color) {
  // printf("%d %d %d\n", color.r, color.g, color.b);
  Color c = GetImageColor(*img, seed.x, seed.y);
  if (c != boundary_color && c != fill_color) {
    ImageDrawPixel(img, seed.x, seed.y, fill_color);
    fill_8(img, (Vector2){.x = seed.x + 1, .y = seed.y}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x, .y = seed.y + 1}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x - 1, .y = seed.y}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x, .y = seed.y - 1}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x + 1, .y = seed.y + 1}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x - 1, .y = seed.y + 1}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x - 1, .y = seed.y - 1}, boundary_color,
           fill_color);
    fill_8(img, (Vector2){.x = seed.x + 1, .y = seed.y - 1}, boundary_color,
           fill_color);
  }
}

int boundary_fill() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);
```

```
  // Create the window and OpenGL context
  InitWindow(1280, 800, "boundary_fill");

  Color BOUNDARY_COLOR = BLACK, FILL_COLOR = GREEN;

  Image img = GenImageColor(1280, 800, WHITE);
  ImageDrawRectangleLines(
      &img, Rectangle{.x = 20, .y = 20, .width = 100, .height = 100}, 2, BLACK);
  ImageDrawCircleLines(&img, 70, 200, 50, BLACK);
  ImageDrawText(&img, "4 Fill", 50, 125, 16, BLACK);

  fill_4(&img, Vector2{.x = 50, .y = 50}, BOUNDARY_COLOR, FILL_COLOR);
  fill_4(&img, Vector2{.x = 70, .y = 200}, BOUNDARY_COLOR, FILL_COLOR);

  ImageDrawRectangleLines(
      &img, Rectangle{.x = 200, .y = 20, .width = 100, .height = 100}, 2,
      BLACK);
  ImageDrawText(&img, "8 Fill", 225, 125, 16, BLACK);

  fill_8(&img, Vector2{.x = 250, .y = 50}, BOUNDARY_COLOR, FILL_COLOR);

  Texture2D tex = LoadTextureFromImage(img);
  UnloadImage(img);

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(WHITE);

    DrawTexture(tex, 0, 0, WHITE);

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```



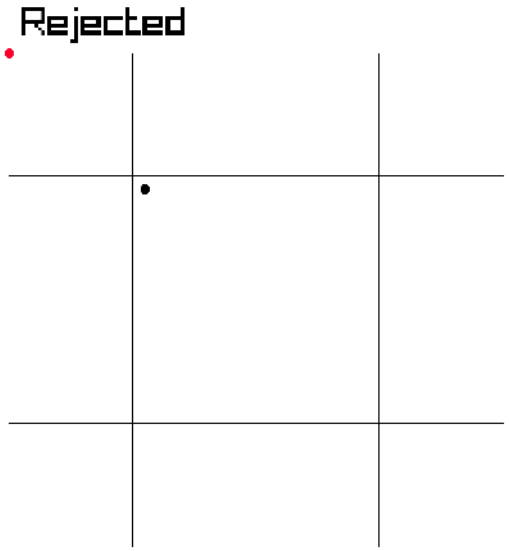
Q10. Implement the flood fill algorithm.
Code –

```
#include "apps.h"
#include "raylib.h"

static bool operator==(Color a, Color b) {
  return (a.a == b.a && a.b == b.b && a.r == b.r && a.g == b.g);
}
static bool operator!=(Color a, Color b) {
  return !(a.a == b.a && a.b == b.b && a.r == b.r && a.g == b.g);
}

static void fill_4(Image *img, Vector2 seed, Color fill_color, Color old_color) {
  Color c = GetImageColor(*img, seed.x, seed.y);
  if (c == old_color) {
    ImageDrawPixel(img, seed.x, seed.y, fill_color);
    fill_4(img, (Vector2){.x = seed.x + 1, .y = seed.y}, fill_color, old_color);
    fill_4(img, (Vector2){.x = seed.x, .y = seed.y + 1}, fill_color, old_color);
    fill_4(img, (Vector2){.x = seed.x - 1, .y = seed.y}, fill_color, old_color);
    fill_4(img, (Vector2){.x = seed.x, .y = seed.y - 1}, fill_color, old_color);
  }
}

static void fill_8(Image *img, Vector2 seed, Color fill_color, Color old_color) {
  // printf("%d %d %d\n", color.r, color.g, color.b);
  Color c = GetImageColor(*img, seed.x, seed.y);
  if (c == old_color) {
    ImageDrawPixel(img, seed.x, seed.y, fill_color);
    fill_8(img, (Vector2){.x = seed.x + 1, .y = seed.y}, fill_color, old_color);
    fill_8(img, (Vector2){.x = seed.x, .y = seed.y + 1}, fill_color, old_color);
    fill_8(img, (Vector2){.x = seed.x - 1, .y = seed.y}, fill_color, old_color);
    fill_8(img, (Vector2){.x = seed.x, .y = seed.y - 1}, fill_color, old_color);
    fill_8(img, (Vector2){.x = seed.x + 1, .y = seed.y + 1}, fill_color,
           old_color);
    fill_8(img, (Vector2){.x = seed.x - 1, .y = seed.y + 1}, fill_color,
           old_color);
    fill_8(img, (Vector2){.x = seed.x - 1, .y = seed.y - 1}, fill_color,
           old_color);
    fill_8(img, (Vector2){.x = seed.x + 1, .y = seed.y - 1}, fill_color,
           old_color);
  }
}

int flood_fill() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "flood_fill");

  Color FILL_COLOR = BLACK, OLD_COLOR = WHITE;
```

```
  Image img = GenImageColor(1280, 800, WHITE);
  ImageDrawRectangleLines(
      &img, Rectangle{.x = 20, .y = 170, .width = 100, .height = 100}, 2, BLACK);
  ImageDrawCircleLines(&img, 70, 70, 50, BLACK);

  fill_4(&img, Vector2{.x = 50, .y = 50}, FILL_COLOR, OLD_COLOR);
  fill_4(&img, Vector2{.x = 50, .y = 250}, FILL_COLOR, OLD_COLOR);
  ImageDrawText(&img, "4 Fill", 50, 125, 16, BLACK);

  ImageDrawRectangleLines(
      &img, Rectangle{.x = 200, .y = 20, .width = 100, .height = 100}, 2,
      BLACK);
  ImageDrawText(&img, "8 Fill", 225, 125, 16, BLACK);

  fill_8(&img, Vector2{.x = 250, .y = 50}, FILL_COLOR, OLD_COLOR);

  Texture2D tex = LoadTextureFromImage(img);
  UnloadImage(img);

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(BLACK);

    DrawTexture(tex, 0, 0, WHITE);

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```

Q11. Implement the point clipping algorithm.
Code –

```c
#include "apps.h"
#include "raylib.h"

void draw_point(Vector2 p, int wxman, int wxmin, int wymax, int wymin) {
  if (p.y > wymax || p.y < wymin || p.x > wxman || p.x < wxmin) {
    DrawCircleV(p, 2, RED);
    DrawText("Rejected", p.x+5, p.y-20, 16, BLACK);
    return;
  }
  DrawCircleV(p, 2, BLACK);
}

int point_clipping() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "point_clipping");

  Vector2 point1 = {.x = 105, .y = 105};
  Vector2 point2 = {.x = 50, .y = 50};
  int wxmax = 200, wxmin = 100, wymax = 200, wymin = 100;

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(WHITE);

    DrawLine(wxmin, 50, wxmin, 250, BLACK);
    DrawLine(wxmax, 50, wxmax, 250, BLACK);
    DrawLine(50, wymin, 250, wymin, BLACK);
    DrawLine(50, wymax, 250, wymax, BLACK);
    draw_point(point1, wxmax, wxmin, wymax, wymin);
    draw_point(point2, wxmax, wxmin, wymax, wymin);

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```


Rejected

Lab 7

Q12. Implement the line clipping algo using Cohen Sutherland method.
Code –

```
#include "apps.h"
#include "raylib.h"
#include <cstdio>
#include <stdio.h>
#include <utility>

#define LEFT 1
#define RIGHT 2
#define BOTTOM 4
#define TOP 8

int get_encoding(Vector2 p, Vector2 topleft, Vector2 bottomright) {
  int encoding = 0;

  if (p.x < topleft.x) {
    encoding |= LEFT;
  }
  if (p.x > bottomright.x) {
    encoding |= RIGHT;
  }
  if (p.y < bottomright.y) {
    encoding |= BOTTOM;
  }
  if (p.y > topleft.y) {
    encoding |= TOP;
  }

  return encoding;
}

void cohen(Vector2 p, Vector2 q, Vector2 topleft, Vector2 bottomright) {
  int encp = get_encoding(p, topleft, bottomright);
  int encq = get_encoding(q, topleft, bottomright);
  // DrawLineV(p, q, BLACK);

  if (encp & encq) {
    return;
  }
  if (encq == 0 && encp == 0) {
    DrawCircleV(p, 2, BLACK);
    DrawCircleV(q, 2, BLACK);
    return;
  }

  if (p.x > q.x || encp == 0) {
    Vector2 temp = p;
    p = q;
    q = temp;
    std::swap(encq, encp);
  }
```

```
  float m = (q.y - p.y) / (q.x - p.x);
  if (encp & LEFT) {
    p.y += (topleft.x - p.x) * m;
    p.x = topleft.x;
  }
  if (encp & RIGHT) {
    p.y += (bottomright.x - p.x) * m;
    p.x = bottomright.x;
  }
  if (encp & BOTTOM) {
    p.x += (bottomright.y - p.y) / m;
    p.y = bottomright.y;
  }
  if (encp & TOP) {
    p.x += (topleft.y - p.y) / m;
    p.y = topleft.y;
  }

  cohen(p, q, topleft, bottomright);
}

int cohen_sutherland() {
  // Tell the window to use vsync and work on high DPI displays
  SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_HIGHDPI);

  // Create the window and OpenGL context
  InitWindow(1280, 800, "cohen_sutherland");

  // game loop
  while (!WindowShouldClose()) {
    // drawing
    BeginDrawing();

    // Setup the back buffer for drawing (clear color and depth buffers)
    ClearBackground(WHITE);

    Vector2 a = {.x = 100, .y = 000};
    Vector2 b = {.x = 100, .y = 400};
    DrawLineV(a, b, BLACK);
    a.x += 300;
    b.x += 300;
    DrawLineV(a, b, BLACK);
    Vector2 c = {.x = 0, .y = 100};
    Vector2 d = {.x = 500, .y = 100};
    DrawLineV(c, d, BLACK);
    c.y += 200;
    d.y += 200;
    DrawLineV(c, d, BLACK);

    Vector2 bottomright = {.x = 400, .y = 100};
    Vector2 topleft = {.x = 100, .y = 300};
    // DrawCircleV(topleft, 2, BLACK);
    // DrawCircleV(bottomright, 2, BLACK);

    Vector2 p = {.x = 45, .y = 200};
    Vector2 q = {.x = 200, .y = 55};
    DrawLineV(p, q, BLACK);
    cohen(p, q, topleft, bottomright);
```

```
    p = {.x = 75, .y = 200};
    q = {.x = 225, .y = 200};
    DrawLineV(p, q, BLACK);
    cohen(p, q, topleft, bottomright);

    p = {.x = 250, .y = 50};
    q = {.x = 250, .y = 200};
    DrawLineV(p, q, BLACK);
    cohen(p, q, topleft, bottomright);

    p = {.x = 50, .y = 320};
    q = {.x = 200, .y = 320};
    DrawLineV(p, q, BLACK);
    cohen(p, q, topleft, bottomright);

    p = {.x = 50, .y = 275};
    q = {.x = 200, .y = 275};
    DrawLineV(p, q, BLACK);
    cohen(p, q, topleft, bottomright);

    EndDrawing();
  }

  // destroy the window and cleanup the OpenGL context
  CloseWindow();
  return 0;
}
```