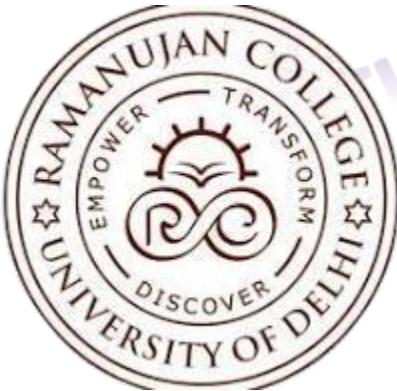


Computer Graphics Practical

UNIVERSITY OF DELHI



Ramanujan College

DSC 09: Computer Graphics

Semester-3

(2024-25)

Submitted By:

~Ratnesh Kumar

20231432(23020570034)

Submitted To:

~Dr. Aakash

Acknowledgement

I would like to take this opportunity to acknowledge everyone who has helped us in every stage of this project.

I am deeply indebted to my mathematics Professor, Dr Aakash for his guidance and suggestions in completing this project. The completion of this project was possible under his guidance and support.

I am also very thankful to my parents and my friends who have boosted me up morally with their continuous support.

At last but not least, I am very thankful to God almighty for showering his blessings upon me.

Index

S.no	Topic	Page No	Remarks
1.	Write a program to implement Bresenham's line drawing algorithm.	1	
2.	Write a program to implement mid-point circle drawing algorithm.	2	
3.	Write a program to clip a line using Cohen and Sutherland line clipping algorithm.	3 - 4	
4.	Write a program to clip a polygon using Sutherland Hodgeman algorithm.	5 - 6	
5.	Write a program to fill a polygon using Scan line fill algorithm.	7 - 8	
6.	Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).	9 - 10	
7.	Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.	11 - 12	
8 a).	Write a program to draw Bezier curve.	13	
8 b).	Write a program to draw Hermite curve.	14	

1. Write a program to implement Bresenham's line drawing algorithm.

Code:

```
#include <graphics.h>
void bresenhamLine(int x1, int y1, int x2, int y2) {
    int dx = abs(x1 - x2);int dy = abs(y1 - y2);
    int p = 2 * dy - dx;int twoDy = 2 * dy;
    int twoDyDx = 2 * (dy - dx);
    int x, y,xend;
    if (x1 > x2) {
        x = x2;y = y2;
        xend = x1;
    } else {
        x = x1;y = y1;
        xend = x2;}
    putpixel(x, y, WHITE);
    while (x < xend) {
        x++;
        if (p < 0) p += twoDy;
        else {
            y++;
            p += twoDyDx;}
        putpixel(x, y, WHITE);
        delay(2);
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"Bresenham's Line Drawing Algorithm");
    int x1 = 100, y1 = 100, x2 = 400, y2 = 400;
    bresenhamLine(x1, y1, x2, y2);
    getch();closegraph();
    return 0;}
```

Output:



2. Write a program to implement mid-point circle drawing algorithm.

Code:

```
#include <graphics.h>
void circleplotPoints(int xc,int yc,int x,int y){
    putpixel(xc + x, yc + y, WHITE);
    putpixel(xc - x, yc + y, WHITE);
    putpixel(xc + x, yc - y, WHITE);
    putpixel(xc - x, yc - y, WHITE);
    putpixel(xc + y, yc + x, WHITE);
    putpixel(xc - y, yc + x, WHITE);
    putpixel(xc + y, yc - x, WHITE);
    putpixel(xc - y, yc - x, WHITE);}
void midpointCircle(int xc, int yc, int r) {
    int x = 0, y = r;int p = 1 - r;
    circleplotPoints(xc,yc,x,y);
    while (x <= y) {
        x++;
        if (p < 0) {
            p += 2 * x + 1;
        } else {
            y--;
            p += 2 * (x - y) + 1;
        }
        circleplotPoints(xc,yc,x,y);
        delay(4);}}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    int xc = 300, yc = 250, radius = 150;
    midpointCircle(xc, yc, radius);
    getch();closegraph();
    return 0;}
```

Output:



3. Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

Code:

```
#include <iostream>
#include <graphics.h>
using namespace std;
const int INSIDE = 0; const int LEFT = 1;
const int RIGHT = 2; const int BOTTOM = 4;
const int TOP = 8;
int computeRegionCode(double x, double y, double x_min, double x_max, double y_min, double y_max) {
    int code = INSIDE;
    if (x < x_min) code |= LEFT;
    else if (x > x_max) code |= RIGHT;
    if (y < y_min) code |= BOTTOM;
    else if (y > y_max) code |= TOP;
    return code;
}
void cohenSutherlandClip(double x0, double y0, double x1, double y1, double x_min, double x_max, double y_min, double y_max) {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);
    rectangle(x_min, y_min, x_max, y_max);
    setcolor(RED);
    line(x0, y0, x1, y1);
    int code0 = computeRegionCode(x0, y0, x_min, x_max, y_min, y_max);
    int code1 = computeRegionCode(x1, y1, x_min, x_max, y_min, y_max);
    bool accept = false;
    while (true) {
        if ((code0 == 0) && (code1 == 0)) {
            accept = true; break;
        } else if (code0 & code1) {break;
        } else {double x, y;
            int outcode = (code0 != 0) ? code0 : code1;
            if (outcode & TOP) {
                x = x0 + (x1 - x0) * (y_max - y0) / (y1 - y0);
                y = y_max;
            } else if (outcode & BOTTOM) {
                x = x0 + (x1 - x0) * (y_min - y0) / (y1 - y0);
                y = y_min;
            } else if (outcode & RIGHT) {
                y = y0 + (y1 - y0) * (x_max - x0) / (x1 - x0);
                x = x_max;
            } else if (outcode & LEFT) {
                y = y0 + (y1 - y0) * (x_min - x0) / (x1 - x0);
                x = x_min;
            }
            if (outcode == code0) {
                x0 = x; y0 = y;
            } else {
                x1 = x; y1 = y;
            }
        }
    }
}
```

```

code0 = computeRegionCode(x0, y0, x_min, x_max, y_min, y_max);
} else {
    x1 = x; y1 = y;
    code1 = computeRegionCode(x1, y1, x_min, x_max, y_min, y_max);}}
if (accept) {
    setcolor(GREEN);
    line(x0, y0, x1, y1);
    cout << "Line accepted from (" << x0 << ", " << y0 << ") to (" << x1 << ", " << y1 << ")" << endl;
} else {
    cout << "Line rejected" << endl;
getch(); closegraph(); }
int main() {
    double x_min = 150.0, y_min = 150.0, x_max = 350.0, y_max = 350.0;
    double x0 = 100.0, y0 = 200.0, x1 = 400.0, y1 = 300.0;
    cohenSutherlandClip(x0, y0, x1, y1, x_min, x_max, y_min, y_max);
    return 0;}

```

Output:



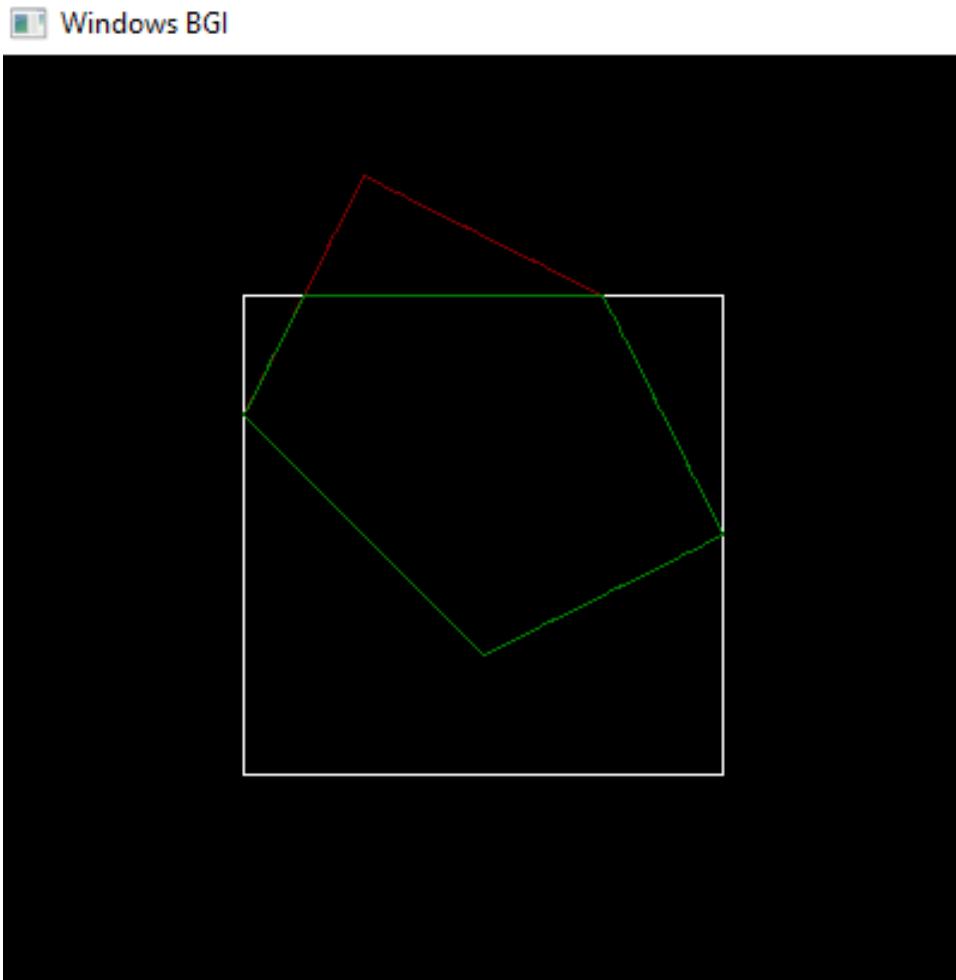
4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

Code:

```
#include <graphics.h>
#include <vector>
#include <iostream>
struct Point {
    int x, y;};
bool isInside(Point p, Point clipEdgeStart, Point clipEdgeEnd) {
    return (clipEdgeEnd.x - clipEdgeStart.x) * (p.y - clipEdgeStart.y) -
        (clipEdgeEnd.y - clipEdgeStart.y) * (p.x - clipEdgeStart.x) >= 0;}
Point getIntersection(Point s, Point e, Point clipEdgeStart, Point clipEdgeEnd) {
    int x1 = s.x, y1 = s.y, int x2 = e.x, y2 = e.y;
    int x3 = clipEdgeStart.x, y3 = clipEdgeStart.y;
    int x4 = clipEdgeEnd.x, y4 = clipEdgeEnd.y;
    int denom = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
    int intersectX = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 * x4)) / denom;
    int intersectY = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 * x4)) / denom;
    return { intersectX, intersectY };}
std::vector<Point> sutherlandHodgmanClip(std::vector<Point>& polygon, std::vector<Point>& clipWindow) {
    std::vector<Point> clippedPolygon = polygon;
    for (size_t i = 0; i < clipWindow.size(); i++) {
        std::vector<Point> newPolygon;
        Point clipEdgeStart = clipWindow[i];
        Point clipEdgeEnd = clipWindow[(i + 1) % clipWindow.size()];
        for (size_t j = 0; j < clippedPolygon.size(); j++) {
            Point current = clippedPolygon[j];
            Point prev = clippedPolygon[(j + clippedPolygon.size() - 1) % clippedPolygon.size()];
            bool currInside = isInside(current, clipEdgeStart, clipEdgeEnd);
            bool prevInside = isInside(prev, clipEdgeStart, clipEdgeEnd);
            if (currInside) {
                if (!prevInside)
                    newPolygon.push_back(getIntersection(prev, current, clipEdgeStart, clipEdgeEnd));
                newPolygon.push_back(current);
            } else if (prevInside) {
                newPolygon.push_back(getIntersection(prev, current, clipEdgeStart, clipEdgeEnd));}}
        clippedPolygon = newPolygon;}
    return clippedPolygon;}
void drawPolygon(const std::vector<Point>& polygon, int color) {
    setcolor(color);
    int n = polygon.size();
    for (int i = 0; i < n; i++) { delay(50);
        line(polygon[i].x, polygon[i].y, polygon[(i + 1) % n].x, polygon[(i + 1) % n].y);
        delay(50);}}
int main() {
    int gd = DETECT, gm; initgraph(&gd, &gm, NULL);
```

```
std::vector<Point> polygon = {{100, 150}, {200, 250}, {300, 200}, {250, 100}, {150, 50}};  
std::vector<Point> clipWindow = {{100, 100}, {300, 100}, {300, 300}, {100, 300}};  
std::vector<Point> clippedPolygon = sutherlandHodgmanClip(polygon, clipWindow);  
drawPolygon(polygon, RED);  
delay(900); drawPolygon(clipWindow, WHITE);  
delay(900); drawPolygon(clippedPolygon, GREEN);  
getch(); closegraph();  
return 0; }
```

Output:



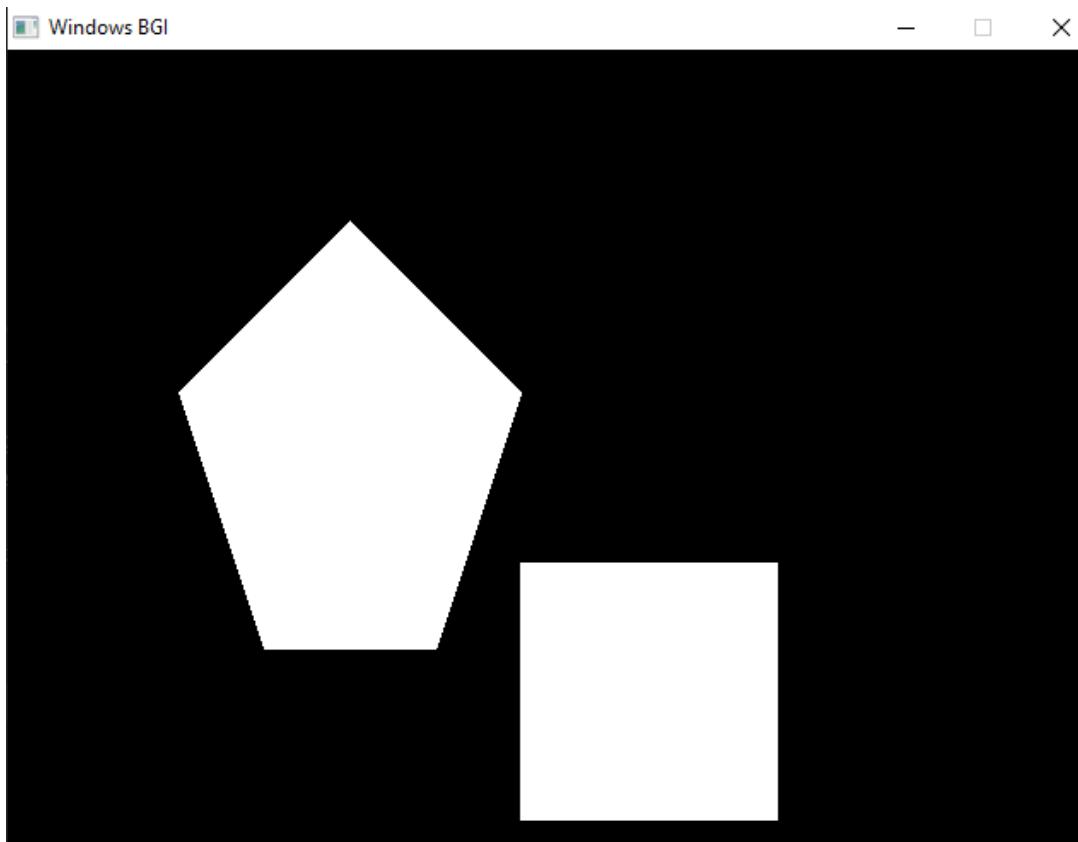
5. Write a program to fill a polygon using Scan line fill algorithm.

Code:

```
#include <graphics.h>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Point {
    int x, y;};
void scanLineFill(vector<Point> polygon) {
    int n = polygon.size();
    int ymin = INT_MAX, ymax = INT_MIN;
    for (int i = 0; i < n; i++) {
        delay(1);
        ymin = min(ymin, polygon[i].y);
        ymax = max(ymax, polygon[i].y);}
    for (int y = ymin; y <= ymax; y++) {
        vector<int> intersections;
        for (int i = 0; i < n; i++) {
            delay(1);
            Point p1 = polygon[i];
            Point p2 = polygon[(i + 1) % n];
            if (p1.y == p2.y) continue;
            if (y > min(p1.y, p2.y) && y <= max(p1.y, p2.y)) {
                int x = p1.x + (y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y);
                intersections.push_back(x);}}
        sort(intersections.begin(), intersections.end());
        for (int i = 0; i < intersections.size(); i += 2) {
            delay(1);
            line(intersections[i], y, intersections[i + 1], y);}}}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    vector<Point> polygon = {{200, 100}, {300, 200}, {250, 350}, {150, 350}, {100, 200}}; //A Pentagon
    vector<Point> rect = {{300, 300}, {450, 300}, {450, 450}, {300, 450}}; //A quadrilateral
    for (int i = 0; i < polygon.size(); i++) {
        Point p1 = polygon[i];
        Point p2 = polygon[(i + 1) % polygon.size()];
        line(p1.x, p1.y, p2.x, p2.y);}
    for (int i = 0; i < rect.size(); i++) {
        Point p1 = rect[i];
        Point p2 = rect[(i + 1) % rect.size()];
```

```
line(p1.x, p1.y, p2.x, p2.y);}  
scanLineFill(polygon);  
scanLineFill(rect);  
getch();closegraph();  
return 0;}
```

Output:



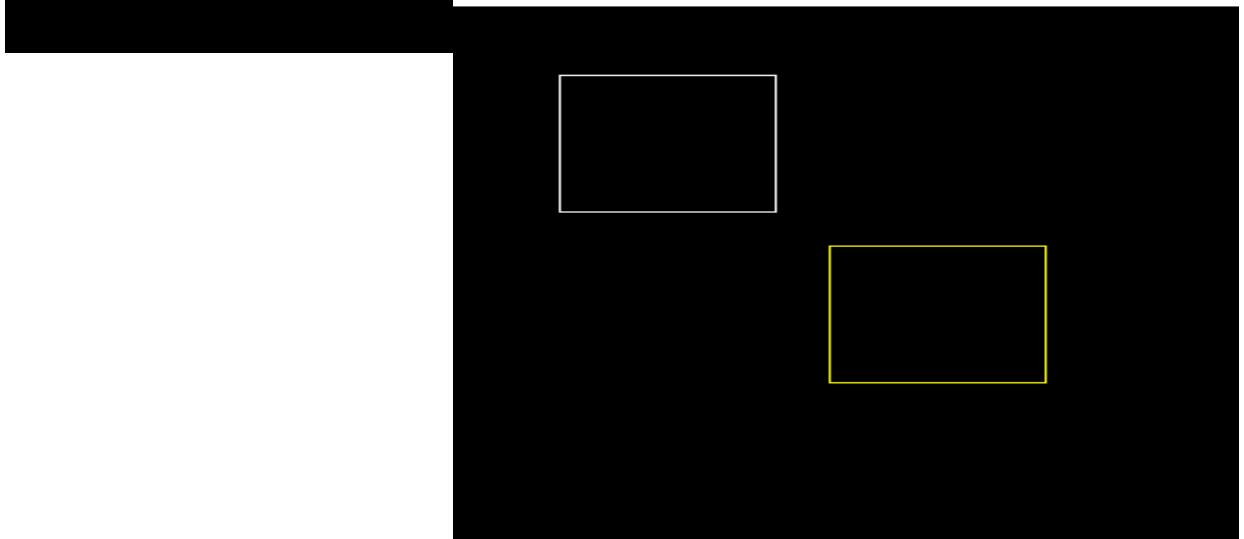
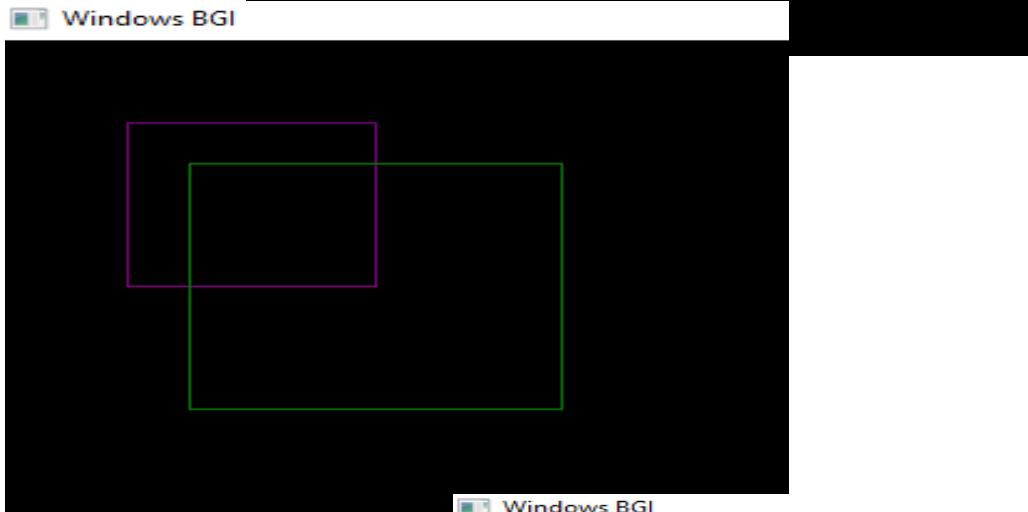
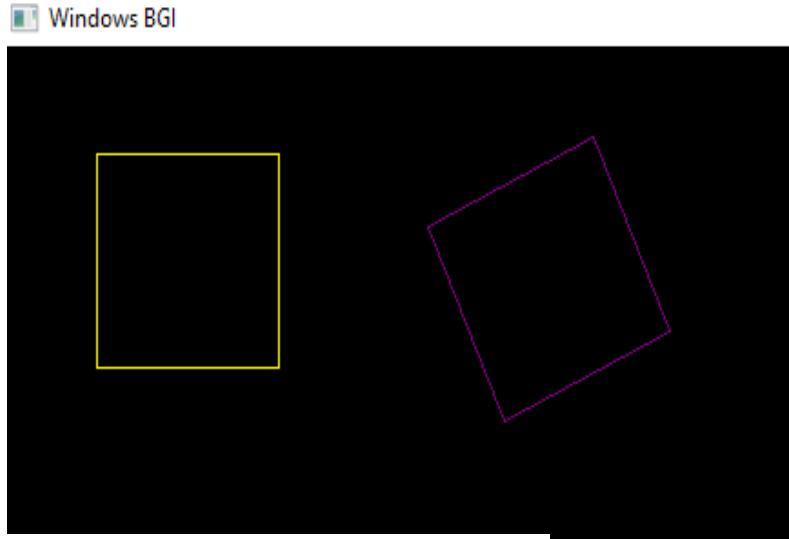
6. Write a program to apply various 2D transformations on a 2D object .

Code:

```
#include <graphics.h>
#include <iostream>
#include <cmath>
using namespace std;
struct Point {
    int x, y, w;
};
void translate(Point transformedshape[], int tx, int ty, int n) {
    for (int i = 0; i < n; i++) {
        transformedshape[i].x += tx;
        transformedshape[i].y += ty;
    }
}
void scale(Point originalshape[], Point transformedshape[], float sx, float sy, int n) {
    for (int i = 0; i < n; i++) {
        transformedshape[i].x = static_cast<int>(originalshape[i].x * sx);
        transformedshape[i].y = static_cast<int>(originalshape[i].y * sy);
    }
}
void rotate(Point originalshape[], Point transformedshape[], float angle, int n) {
    for (int i = 0; i < n; i++) {
        float rad = angle * M_PI / 180.0;
        int x_new = static_cast<int>(originalshape[i].x * cos(rad) - originalshape[i].y * sin(rad));
        int y_new = static_cast<int>(originalshape[i].x * sin(rad) + originalshape[i].y * cos(rad));
        transformedshape[i].x = x_new;
        transformedshape[i].y = y_new;
    }
}
void drawShape(Point shape[], int n) {
    for (int i = 0; i < n; i++) {
        delay(10);
        line(shape[i].x, shape[i].y, shape[(i + 1) % n].x, shape[(i + 1) % n].y);
        delay(10);
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    Point originalshape[] = {{50, 50, 1}, {150, 50, 1}, {150, 150, 1}, {50, 150, 1}}; // A quadrilateral
    int n = sizeof(originalshape) / sizeof(originalshape[0]);
    Point transformedshape[4];
    copy(begin(originalshape), end(originalshape), begin(transformedshape));
    int tx = 125, ty = 125; float sx = 1.5, sy = 1.5;
    float angle = -25;
    drawShape(originalshape, n);
    translate(transformedshape, tx, ty, n);
    setcolor(YELLOW);
    drawShape(transformedshape, n);
    getch(); cleardevice();
    drawShape(originalshape, n);
    rotate(transformedshape, transformedshape, angle, n);
    setcolor(MAGENTA);
```

```
drawShape(transformedshape, n);
getch(); cleardevice();
drawShape(originalshape, n);
scale(originalshape, transformedshape, sx, sy,n);
setcolor(GREEN);
drawShape(transformedshape, n); getch();
closegraph();
return 0;}
```

Output:



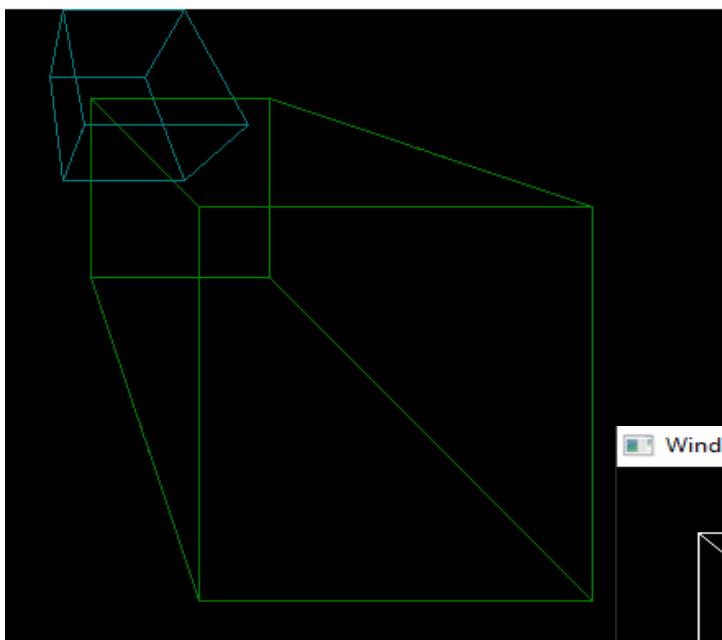
7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

Code:

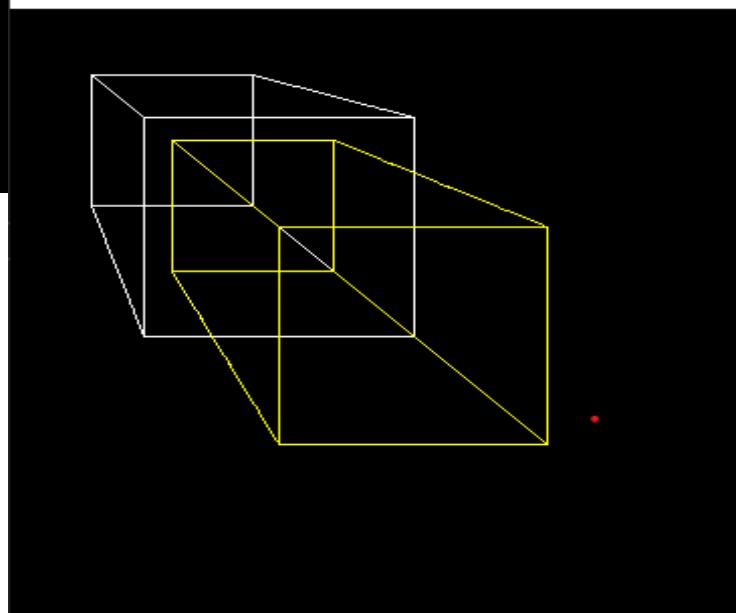
```
#include <graphics.h>
#include <conio.h>
#include <iostream>
#include <cmath>
using namespace std;
struct Point3D {
    int x, y, z;
};
Point3D project(Point3D p, int d = 200) {
    Point3D projected;
    projected.x = p.x * d / (d + p.z);
    projected.y = p.y * d / (d + p.z);
    projected.z = p.z;
    return projected;
}
void drawCube(Point3D points[]) {
    for (int i = 0; i < 4; i++) {
        Point3D p1 = project(points[i]);
        Point3D p2 = project(points[(i + 1) % 4]);
        Point3D p3 = project(points[i + 4]);
        Point3D p4 = project(points[(i + 1) % 4 + 4]);
        line(p1.x, p1.y, p2.x, p2.y);
        line(p3.x, p3.y, p4.x, p4.y);
        line(p1.x, p1.y, p3.x, p3.y);
    }
}
void translate(Point3D points[], Point3D original[], int tx, int ty, int tz) {
    for (int i = 0; i < 8; i++) {
        points[i].x = original[i].x + tx;
        points[i].y = original[i].y + ty;
        points[i].z = original[i].z + tz;
    }
}
void scale(Point3D points[], Point3D original[], float sx, float sy, float sz) {
    for (int i = 0; i < 8; i++) {
        points[i].x = original[i].x * sx;
        points[i].y = original[i].y * sy;
        points[i].z = original[i].z * sz;
    }
}
void rotateX(Point3D points[], Point3D original[], float angle) {
    float rad = angle * M_PI / 180.0;
    for (int i = 0; i < 8; i++) {
        points[i].x = original[i].x;
        points[i].y = original[i].y * cos(rad) - original[i].z * sin(rad);
        points[i].z = original[i].y * sin(rad) + original[i].z * cos(rad);
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);
    Point3D originalCube[8] = {
```

```
{50, 50, 50}, {150, 50, 50}, {150, 150, 50}, {50, 150, 50},  
{50, 50, -50}, {150, 50, -50}, {150, 150, -50}, {50, 150, -50}};  
Point3D transformedCube[8];  
copy(begin(originalCube), end(originalCube), begin(transformedCube));  
drawCube(transformedCube);  
setcolor(YELLOW); translate(transformedCube, originalCube, 50, 50, 0);  
drawCube(transformedCube);  
getch(); cleardevice();  
drawCube(transformedCube);  
setcolor(GREEN); scale(transformedCube, originalCube, 1.5, 1.5, 1.5);  
drawCube(transformedCube);  
getch(); cleardevice();  
drawCube(transformedCube);  
setcolor(CYAN); rotateX(transformedCube, originalCube, 45);  
drawCube(transformedCube);  
getch(); cleardevice();  
closegraph(); return 0;}
```

Output:



Windows BGI



8 a). Write a program to draw Bezier curve.

Code:

```
#include <graphics.h>
#include <iostream>
#include <math.h>
using namespace std;
void drawBezierCurve(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3) {
    for (float t = 0; t <= 1; t += 0.001) {
        delay(5);
        float xt = pow(1 - t, 3) * x0 + 3 * t * pow(1 - t, 2) * x1 + 3 * pow(t, 2) * (1 - t) * x2 + pow(t, 3) * x3;
        float yt = pow(1 - t, 3) * y0 + 3 * t * pow(1 - t, 2) * y1 + 3 * pow(t, 2) * (1 - t) * y2 + pow(t, 3) * y3;
        putpixel(xt, yt, WHITE);}}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*) "");
    int x0 = 100, y0 = 400;
    int x1 = 200, y1 = 100;
    int x2 = 400, y2 = 100;
    int x3 = 500, y3 = 400;
    drawBezierCurve(x0, y0, x1, y1, x2, y2, x3, y3);
    getch();closegraph();
    return 0;}
```

Output:



8 b). Write a program to draw Hermite curve.

Code:

```
#include <graphics.h>
#include <conio.h>
#include <iostream>
#include <math.h>
using namespace std;
void drawHermiteCurve(int x0, int y0, int x1, int y1, int rx0, int ry0, int rx1, int ry1) {
    for (float t = 0.0; t <= 1.0; t += 0.001) {
        delay(2);
        float h1 = 2*t*t*t - 3*t*t + 1;
        float h2 = -2*t*t*t + 3*t*t;
        float h3 = t*t*t - 2*t*t + t;
        float h4 = t*t*t - t*t;
        int x = h1*x0 + h2*x1 + h3*rx0 + h4*rx1;
        int y = h1*y0 + h2*y1 + h3*ry0 + h4*ry1;
        putpixel(x, y, WHITE);}}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*) "");
    int x0 = 100, y0 = 400;
    int x1 = 400, y1 = 100;
    int rx0 = 150, ry0 = 200;
    int rx1 = 100, ry1 = 200;
    drawHermiteCurve(x0, y0, x1, y1, rx0, ry0, rx1, ry1);
    getch();closegraph();return 0;
}
```

Output:

