

Name: - Rohan Jain

Subject: - Computer Graphics

Roll No.: - 23020570039

Submitted to: - DR. Aakash

| **S. No** | **Practical Title** | **Page no.** |
| --- | --- | --- |
| 1 | Write a program to implement Bresenham’s line drawing algorithm. | 1 |
| 2 | Write a program to implement a midpoint circle drawing algorithm. | 5 |
| 3 | Write a program to clip a line using Cohen and Sutherland line clipping algorithm. | 10 |
| 4 | Write a program to clip a polygon using Sutherland Hodgemann algorithm. | 15 |
| 5 | Write a program to fill a polygon using the Scan line fill algorithm. | 20 |
| 6 | Write a program to apply various 2D transformations on a 2D object (use homogeneous Coordinates). | 25 |
| 7 | Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it. |  |
| 8 | Write a program to draw Hermite /Bezier curve. |  |

**Acknowledgment**

I am deeply grateful to my instructor, DR. Aakash, for giving me the opportunity to work on this series of practical. Their insightful guidance and unwavering support have been pivotal in helping me grasp the fundamental concepts of computer graphics.

I would also like to extend my gratitude to Ramanujan College, University of Delhi, for designing such a comprehensive curriculum that emphasizes both technical learning and hands-on experience. Exploring the practical applications of various algorithms and transformation techniques has significantly enriched my understanding of the subject.

Lastly, I sincerely thank my peers for their encouragement and collaborative spirit throughout this project. Their contributions have made the learning process more engaging and fostered an environment conducive to effective problem-solving.

**Q1. Write a program to implement Bresenham’s line drawing algorithm**.

**Code:**

#include <iostream>

#include <graphics.h>

using namespace std;

void bresenhamLine(int x1, int y1, int x2, int y2) {

    int dx = x2 - x1;

    int dy = y2 - y1;

    int p = 2 \* dy - dx;

    int x = x1, y = y1;

    while (x <= x2) {

        putpixel(x, y, WHITE);

        if (p >= 0) {

            y++;

            p += 2 \* dy - 2 \* dx;

        } else {

            p += 2 \* dy;

        }

        x++;

    }

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    int x1 = 100, y1 = 100, x2 = 200, y2 = 150;

    bresenhamLine(x1, y1, x2, y2);

    getch();

    closegraph();

    return 0;

}

**Output:**

A white line in a black background

Description automatically generated

**Q2. Write a program to implement a midpoint circle drawing algorithm.**

**Code:**

#include <iostream>

#include <graphics.h>

using namespace std;

void midpointCircle(int xc, int yc, int r) {

    int x = 0, y = r;

    int p = 1 - r;

    while (x <= 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);

        if (p < 0) {

            p += 2 \* x + 3;

        } else {

            p += 2 \* (x - y) + 5;

            y--;

        }

        x++;

    }

}

int main() {

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "");

    int xc = 200, yc = 200, r = 100;

    midpointCircle(xc, yc, r);

    getch();

    closegraph();

    return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Q3. 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 xmin = 100, ymin = 100, xmax = 300, ymax = 300;

int computeCode(int x, int y) {

    int code = INSIDE;

    if (x < xmin) code |= LEFT;

    else if (x > xmax) code |= RIGHT;

    if (y < ymin) code |= BOTTOM;

    else if (y > ymax) code |= TOP;

    return code;

}

void cohenSutherlandClip(int x1, int y1, int x2, int y2) {

    int code1 = computeCode(x1, y1);

    int code2 = computeCode(x2, y2);

    bool accept = false;

    while (true) {

        if ((code1 == 0) && (code2 == 0)) {

            accept = true;

            break;

        } else if (code1 & code2) {

            break;

        } else {

            int x, y;

            int outcode = code1 ? code1 : code2;

            if (outcode & TOP) {

                x = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);

                y = ymax;

            } else if (outcode & BOTTOM) {

                x = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);

                y = ymin;

            } else if (outcode & RIGHT) {

                y = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);

                x = xmax;

            } else {

                y = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);

                x = xmin;

            }

            if (outcode == code1) {

                x1 = x;

                y1 = y;

                code1 = computeCode(x1, y1);

            } else {

                x2 = x;

                y2 = y;

                code2 = computeCode(x2, y2);

            }

        }

    }

    if (accept) {

        setcolor(WHITE);

        rectangle(xmin, ymin, xmax, ymax);

        setcolor(GREEN);

        line(x1, y1, x2, y2);

    }

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    int x1 = 50, y1 = 150, x2 = 250, y2 = 350;

    cohenSutherlandClip(x1, y1, x2, y2);

    getch();

    closegraph();

    return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**4. Write a program to clip a polygon using Sutherland Hodgemann algorithm.**

**Code:**

#include <graphics.h>

#include <iostream>

#include <vector>

using namespace std;

struct Point {

    int x, y;

};

// Clip boundary values

int x\_min = 100, y\_min = 100;

int x\_max = 400, y\_max = 400;

// Function to check if a point is inside the clipping window

bool inside(Point p, int edge) {

    switch (edge) {

        case 1: return p.x >= x\_min; // Left edge

        case 2: return p.x <= x\_max; // Right edge

        case 3: return p.y >= y\_min; // Bottom edge

        case 4: return p.y <= y\_max; // Top edge

    }

    return false;

}

// Function to find the intersection point between polygon edge and window boundary

Point intersection(Point p1, Point p2, int edge) {

    Point i;

    float m = (float)(p2.y - p1.y) / (p2.x - p1.x);

    switch (edge) {

        case 1: // Left edge

            i.x = x\_min;

            i.y = p1.y + (x\_min - p1.x) \* m;

            break;

        case 2: // Right edge

            i.x = x\_max;

            i.y = p1.y + (x\_max - p1.x) \* m;

            break;

        case 3: // Bottom edge

            i.y = y\_min;

            i.x = p1.x + (y\_min - p1.y) / m;

            break;

        case 4: // Top edge

            i.y = y\_max;

            i.x = p1.x + (y\_max - p1.y) / m;

            break;

    }

    return i;

}

// Sutherland-Hodgman clipping algorithm

vector<Point> clipPolygon(vector<Point> polygon, int edge) {

    vector<Point> clippedPolygon;

    int n = polygon.size();

    Point s = polygon[n - 1]; // Starting point (previous vertex)

    for (int i = 0; i < n; i++) {

        Point p = polygon[i]; // Current vertex

        if (inside(p, edge)) {

            if (inside(s, edge)) {

                // Both points are inside, add the current point

                clippedPolygon.push\_back(p);

            } else {

                // Previous point is outside, current point is inside, add intersection and current point

                clippedPolygon.push\_back(intersection(s, p, edge));

                clippedPolygon.push\_back(p);

            }

        } else {

            if (inside(s, edge)) {

                // Previous point is inside, current point is outside, add intersection

                clippedPolygon.push\_back(intersection(s, p, edge));

            }

        }

        s = p; // Update previous point

    }

    return clippedPolygon;

}

// Function to draw a polygon

void drawPolygon(vector<Point> polygon, int color) {

    int n = polygon.size();

    setcolor(color);

    for (int i = 0; i < n; i++) {

        line(polygon[i].x, polygon[i].y, polygon[(i + 1) % n].x, polygon[(i + 1) % n].y);

    }

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    // Original polygon vertices

    vector<Point> polygon = {{150, 150}, {300, 50}, {450, 150}, {350, 250}, {200, 250}};

    // Draw clipping window

    rectangle(x\_min, y\_min, x\_max, y\_max);

    // Draw the original polygon in RED

    drawPolygon(polygon, RED);

    getch();

    // Clip polygon against the four edges of the clipping window

    for (int edge = 1; edge <= 4; edge++) {

        polygon = clipPolygon(polygon, edge);

    }

    // Clear the screen and draw the clipped polygon in GREEN

    cleardevice();

    rectangle(x\_min, y\_min, x\_max, y\_max);

    drawPolygon(polygon, GREEN);

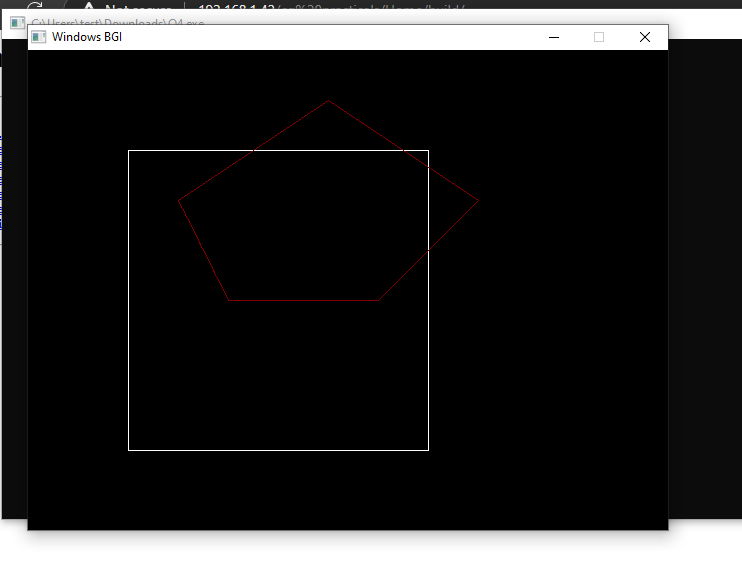
    getch();

    closegraph();

    return 0;

}

**Output:**



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

**Code:**

#include <iostream>

#include <graphics.h>

using namespace std;

struct Point {

    int x, y;

};

void scanLine(Point polygon[], int n) {

    int ymin = polygon[0].y, ymax = polygon[0].y;

    for (int i = 1; i < n; i++) {

        if (polygon[i].y < ymin) ymin = polygon[i].y;

        if (polygon[i].y > ymax) ymax = polygon[i].y;

    }

    for (int y = ymin; y <= ymax; y++) {

        int xIntersections[20], k = 0;

        for (int i = 0; i < n; i++) {

            int x1 = polygon[i].x, y1 = polygon[i].y;

            int x2 = polygon[(i + 1) % n].x, y2 = polygon[(i + 1) % n].y;

            if (y1 != y2) {

                if (y >= min(y1, y2) && y <= max(y1, y2)) {

                    int x = x1 + (y - y1) \* (x2 - x1) / (y2 - y1);

                    xIntersections[k++] = x;

                }

            }

        }

        for (int i = 0; i < k - 1; i += 2) {

            line(xIntersections[i], y, xIntersections[i + 1], y);

        }

    }

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    Point polygon[] = {{200, 100}, {300, 200}, {250, 300}, {150, 300}, {100, 200}};

    int n = sizeof(polygon) / sizeof(polygon[0]);

    setcolor(WHITE);

    for (int i = 0; i < n; i++) {

        line(polygon[i].x, polygon[i].y, polygon[(i + 1) % n].x, polygon[(i + 1) % n].y);

    }

    scanLine(polygon, n);

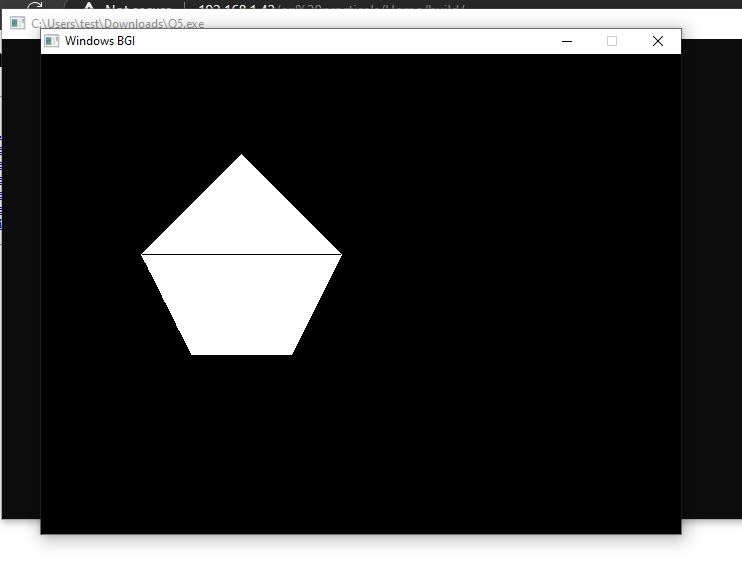
    getch();

    closegraph();

    return 0;

}

**Output:**



**Q6. Write a program to apply various 2D transformations on a 2D object (use homogeneous Coordinates).**

**Code:**

#include <iostream>

#include <graphics.h>

#include <cmath>

using namespace std;

void applyTransformation(float matrix[3][3], int &x, int &y) {

    int xNew = matrix[0][0] \* x + matrix[0][1] \* y + matrix[0][2];

    int yNew = matrix[1][0] \* x + matrix[1][1] \* y + matrix[1][2];

    x = xNew;

    y = yNew;

}

void drawTransformedObject(float matrix[3][3]) {

    int x1 = 100, y1 = 100;

    int x2 = 200, y2 = 100;

    int x3 = 200, y3 = 200;

    int x4 = 100, y4 = 200;

    applyTransformation(matrix, x1, y1);

    applyTransformation(matrix, x2, y2);

    applyTransformation(matrix, x3, y3);

    applyTransformation(matrix, x4, y4);

    line(x1, y1, x2, y2);

    line(x2, y2, x3, y3);

    line(x3, y3, x4, y4);

    line(x4, y4, x1, y1);

}

void translate(int tx, int ty) {

    float translationMatrix[3][3] = {

        {1, 0, static\_cast<float>(tx)},

        {0, 1, static\_cast<float>(ty)},

        {0, 0, 1}

    };

    drawTransformedObject(translationMatrix);

}

void scale(float sx, float sy) {

    float scaleMatrix[3][3] = {

        {sx, 0, 0},

        {0, sy, 0},

        {0,  0, 1}

    };

    drawTransformedObject(scaleMatrix);

}

void rotate(float angle) {

    float rad = angle \* M\_PI / 180.0;

    float rotationMatrix[3][3] = {

        {cos(rad), -sin(rad), 0},

        {sin(rad), cos(rad), 0},

        {0, 0, 1}

    };

    drawTransformedObject(rotationMatrix);

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    // Original object

    setcolor(WHITE);

    int x1 = 100, y1 = 100, x2 = 200, y2 = 100, x3 = 200, y3 = 200, x4 = 100, y4 = 200;

    line(x1, y1, x2, y2);

    line(x2, y2, x3, y3);

    line(x3, y3, x4, y4);

    line(x4, y4, x1, y1);

    // Apply transformations

    setcolor(YELLOW);

    translate(50, 50);

    setcolor(RED);

    scale(0.5, 0.5);

    setcolor(GREEN);

    rotate(45);

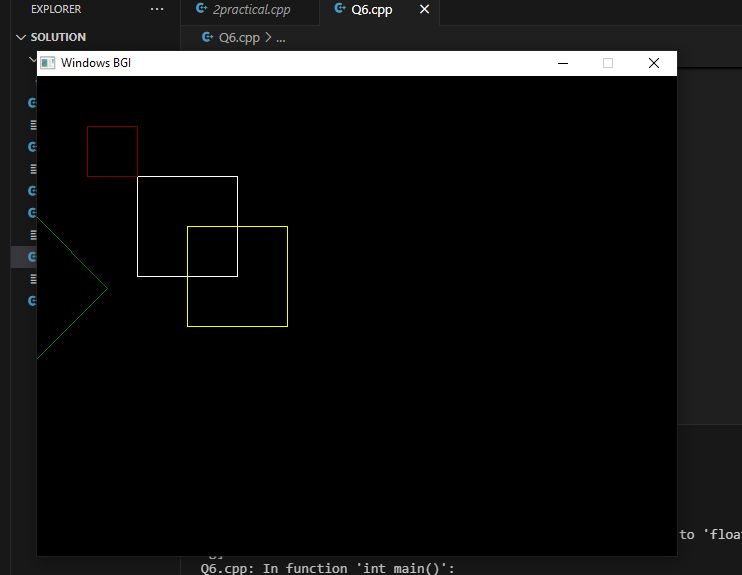
    getch();

    closegraph();

    return 0;

}

**Output:**



**Q7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.**

**Code: parallel projection**

#include <iostream>

#include <graphics.h>

#include <cmath>

using namespace std;

// Function to apply 3D transformations and then parallel projection

void apply3DTransformation(float matrix[4][4], int &x, int &y, int &z) {

    int xNew = matrix[0][0] \* x + matrix[0][1] \* y + matrix[0][2] \* z + matrix[0][3];

    int yNew = matrix[1][0] \* x + matrix[1][1] \* y + matrix[1][2] \* z + matrix[1][3];

    int zNew = matrix[2][0] \* x + matrix[2][1] \* y + matrix[2][2] \* z + matrix[2][3];

    x = xNew;

    y = yNew;

    z = zNew;

}

// Function to project 3D points onto a 2D plane (Parallel Projection)

void parallelProjection(int x, int y, int z, int &x2D, int &y2D) {

    // In parallel projection, the z-coordinate is ignored

    x2D = x;

    y2D = y;

}

// Draw a transformed 3D object projected in 2D

void drawTransformed3DObject(float matrix[4][4]) {

    // Cube vertices

    int x1 = 100, y1 = 100, z1 = 100;

    int x2 = 200, y2 = 100, z2 = 100;

    int x3 = 200, y3 = 200, z3 = 100;

    int x4 = 100, y4 = 200, z4 = 100;

    int x5 = 100, y5 = 100, z5 = 200;

    int x6 = 200, y6 = 100, z6 = 200;

    int x7 = 200, y7 = 200, z7 = 200;

    int x8 = 100, y8 = 200, z8 = 200;

    // Apply transformation

    apply3DTransformation(matrix, x1, y1, z1);

    apply3DTransformation(matrix, x2, y2, z2);

    apply3DTransformation(matrix, x3, y3, z3);

    apply3DTransformation(matrix, x4, y4, z4);

    apply3DTransformation(matrix, x5, y5, z5);

    apply3DTransformation(matrix, x6, y6, z6);

    apply3DTransformation(matrix, x7, y7, z7);

    apply3DTransformation(matrix, x8, y8, z8);

    // Project 3D points to 2D (Parallel Projection)

    int x1\_2D, y1\_2D, x2\_2D, y2\_2D, x3\_2D, y3\_2D, x4\_2D, y4\_2D;

    int x5\_2D, y5\_2D, x6\_2D, y6\_2D, x7\_2D, y7\_2D, x8\_2D, y8\_2D;

    parallelProjection(x1, y1, z1, x1\_2D, y1\_2D);

    parallelProjection(x2, y2, z2, x2\_2D, y2\_2D);

    parallelProjection(x3, y3, z3, x3\_2D, y3\_2D);

    parallelProjection(x4, y4, z4, x4\_2D, y4\_2D);

    parallelProjection(x5, y5, z5, x5\_2D, y5\_2D);

    parallelProjection(x6, y6, z6, x6\_2D, y6\_2D);

    parallelProjection(x7, y7, z7, x7\_2D, y7\_2D);

    parallelProjection(x8, y8, z8, x8\_2D, y8\_2D);

    // Draw cube edges

    setcolor(WHITE);

    line(x1\_2D, y1\_2D, x2\_2D, y2\_2D);

    line(x2\_2D, y2\_2D, x3\_2D, y3\_2D);

    line(x3\_2D, y3\_2D, x4\_2D, y4\_2D);

    line(x4\_2D, y4\_2D, x1\_2D, y1\_2D);

    line(x5\_2D, y5\_2D, x6\_2D, y6\_2D);

    line(x6\_2D, y6\_2D, x7\_2D, y7\_2D);

    line(x7\_2D, y7\_2D, x8\_2D, y8\_2D);

    line(x8\_2D, y8\_2D, x5\_2D, y5\_2D);

    line(x1\_2D, y1\_2D, x5\_2D, y5\_2D);

    line(x2\_2D, y2\_2D, x6\_2D, y6\_2D);

    line(x3\_2D, y3\_2D, x7\_2D, y7\_2D);

    line(x4\_2D, y4\_2D, x8\_2D, y8\_2D);

}

// 3D Translation

void translate3D(int tx, int ty, int tz) {

    float translationMatrix[4][4] = {

        {1, 0, 0, tx},

        {0, 1, 0, ty},

        {0, 0, 1, tz},

        {0, 0, 0, 1}

    };

    drawTransformed3DObject(translationMatrix);

}

// 3D Scaling

void scale3D(float sx, float sy, float sz) {

    float scaleMatrix[4][4] = {

        {sx, 0, 0, 0},

        {0, sy, 0, 0},

        {0, 0, sz, 0},

        {0, 0, 0, 1}

    };

    drawTransformed3DObject(scaleMatrix);

}

// 3D Rotation around Z-axis

void rotate3D(float angle) {

    float rad = angle \* M\_PI / 180.0;

    float rotationMatrix[4][4] = {

        {cos(rad), -sin(rad), 0, 0},

        {sin(rad), cos(rad), 0, 0},

        {0, 0, 1, 0},

        {0, 0, 0, 1}

    };

    drawTransformed3DObject(rotationMatrix);

}

int main() {

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "");

    // Apply transformations and projections

    setcolor(YELLOW);

    translate3D(50, 50, 50);

    setcolor(RED);

    scale3D(1.5, 1.5, 1.5);

    setcolor(GREEN);

    rotate3D(45);

    getch();

    closegraph();

    return 0;

}**Output:** A screenshot of a computer

Description automatically generated

**Code: perspective projection**

#include <iostream>

#include <graphics.h>

#include <cmath>

using namespace std;

// Function to apply 3D transformations

void apply3DTransformation(float matrix[4][4], int &x, int &y, int &z) {

    int xNew = matrix[0][0] \* x + matrix[0][1] \* y + matrix[0][2] \* z + matrix[0][3];

    int yNew = matrix[1][0] \* x + matrix[1][1] \* y + matrix[1][2] \* z + matrix[1][3];

    int zNew = matrix[2][0] \* x + matrix[2][1] \* y + matrix[2][2] \* z + matrix[2][3];

    x = xNew;

    y = yNew;

    z = zNew;

}

// Function to project 3D points onto a 2D plane using Perspective Projection

void perspectiveProjection(int x, int y, int z, int &x2D, int &y2D, int d) {

    x2D = static\_cast<float>(x) \* d / (z + d); // Cast to float for precision

    y2D = static\_cast<float>(y) \* d / (z + d);

}

// Draw a transformed 3D object projected in 2D using Perspective Projection

void drawTransformed3DObject(float matrix[4][4], int d) {

    // Cube vertices

    int x1 = 100, y1 = 100, z1 = 100;

    int x2 = 200, y2 = 100, z2 = 100;

    int x3 = 200, y3 = 200, z3 = 100;

    int x4 = 100, y4 = 200, z4 = 100;

    int x5 = 100, y5 = 100, z5 = 200;

    int x6 = 200, y6 = 100, z6 = 200;

    int x7 = 200, y7 = 200, z7 = 200;

    int x8 = 100, y8 = 200, z8 = 200;

    // Apply transformation

    apply3DTransformation(matrix, x1, y1, z1);

    apply3DTransformation(matrix, x2, y2, z2);

    apply3DTransformation(matrix, x3, y3, z3);

    apply3DTransformation(matrix, x4, y4, z4);

    apply3DTransformation(matrix, x5, y5, z5);

    apply3DTransformation(matrix, x6, y6, z6);

    apply3DTransformation(matrix, x7, y7, z7);

    apply3DTransformation(matrix, x8, y8, z8);

    // Project 3D points to 2D (Perspective Projection)

    int x1\_2D, y1\_2D, x2\_2D, y2\_2D, x3\_2D, y3\_2D, x4\_2D, y4\_2D;

    int x5\_2D, y5\_2D, x6\_2D, y6\_2D, x7\_2D, y7\_2D, x8\_2D, y8\_2D;

    perspectiveProjection(x1, y1, z1, x1\_2D, y1\_2D, d);

    perspectiveProjection(x2, y2, z2, x2\_2D, y2\_2D, d);

    perspectiveProjection(x3, y3, z3, x3\_2D, y3\_2D, d);

    perspectiveProjection(x4, y4, z4, x4\_2D, y4\_2D, d);

    perspectiveProjection(x5, y5, z5, x5\_2D, y5\_2D, d);

    perspectiveProjection(x6, y6, z6, x6\_2D, y6\_2D, d);

    perspectiveProjection(x7, y7, z7, x7\_2D, y7\_2D, d);

    perspectiveProjection(x8, y8, z8, x8\_2D, y8\_2D, d);

    // Draw cube edges using 2D projected points

    setcolor(WHITE);

    line(x1\_2D, y1\_2D, x2\_2D, y2\_2D);

    line(x2\_2D, y2\_2D, x3\_2D, y3\_2D);

    line(x3\_2D, y3\_2D, x4\_2D, y4\_2D);

    line(x4\_2D, y4\_2D, x1\_2D, y1\_2D);

    line(x5\_2D, y5\_2D, x6\_2D, y6\_2D);

    line(x6\_2D, y6\_2D, x7\_2D, y7\_2D);

    line(x7\_2D, y7\_2D, x8\_2D, y8\_2D);

    line(x8\_2D, y8\_2D, x5\_2D, y5\_2D);

    line(x1\_2D, y1\_2D, x5\_2D, y5\_2D);

    line(x2\_2D, y2\_2D, x6\_2D, y6\_2D);

    line(x3\_2D, y3\_2D, x7\_2D, y7\_2D);

    line(x4\_2D, y4\_2D, x8\_2D, y8\_2D);

}

// 3D Translation

void translate3D(int tx, int ty, int tz, int d) {

    float translationMatrix[4][4] = {

        {1, 0, 0, static\_cast<float>(tx)},

        {0, 1, 0, static\_cast<float>(ty)},

        {0, 0, 1, static\_cast<float>(tz)},

        {0, 0, 0, 1}

    };

    drawTransformed3DObject(translationMatrix, d);

}

// 3D Scaling

void scale3D(float sx, float sy, float sz, int d) {

    float scaleMatrix[4][4] = {

        {sx, 0, 0, 0},

        {0, sy, 0, 0},

        {0, 0, sz, 0},

        {0, 0, 0, 1}

    };

    drawTransformed3DObject(scaleMatrix, d);

}

// 3D Rotation around Z-axis

void rotate3D(float angle, int d) {

    float rad = angle \* M\_PI / 180.0;

    float rotationMatrix[4][4] = {

        {cos(rad), -sin(rad), 0, 0},

        {sin(rad), cos(rad), 0, 0},

        {0, 0, 1, 0},

        {0, 0, 0, 1}

    };

    drawTransformed3DObject(rotationMatrix, d);

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    // Distance from the projection plane

    int d = 500;

    // Apply transformations and perspective projection

    setcolor(YELLOW);

    translate3D(50, 50, 50, d);

    setcolor(RED);

    scale3D(1.5, 1.5, 1.5, d);

    setcolor(GREEN);

    rotate3D(45, d);

    getch();

    closegraph();

    return 0;

}

**Output:** A screenshot of a computer

Description automatically generated

**Q8. Write a program to draw Hermite /Bezier curve.**

**Code:**

#include <graphics.h>

#include <iostream>

using namespace std;

void drawHermiteCurve(int x0, int y0, int x1, int y1, int rx0, int ry0, int rx1, int ry1) {

    float t, h00, h10, h01, h11;

    int x, y;

    for (t = 0.0; t <= 1.0; t += 0.001) {

        h00 = 2\*t\*t\*t - 3\*t\*t + 1;

        h10 = t\*t\*t - 2\*t\*t + t;

        h01 = -2\*t\*t\*t + 3\*t\*t;

        h11 = t\*t\*t - t\*t;

        x = h00\*x0 + h10\*rx0 + h01\*x1 + h11\*rx1;

        y = h00\*y0 + h10\*ry0 + h01\*y1 + h11\*ry1;

        putpixel(x, y, WHITE);

    }

}

int main() {

    int gd = DETECT, gm;

    char data[] = "C:\\MinGW\\lib\\libbgi.a";

    initgraph(&gd, &gm, data);

    int x0 = 100, y0 = 200, x1 = 300, y1 = 300;

    int rx0 = 100, ry0 = 100, rx1 = 200, ry1 = 150;

    drawHermiteCurve(x0, y0, x1, y1, rx0, ry0, rx1, ry1);

    getch();

    closegraph();

    return 0;

}

**Output:** 