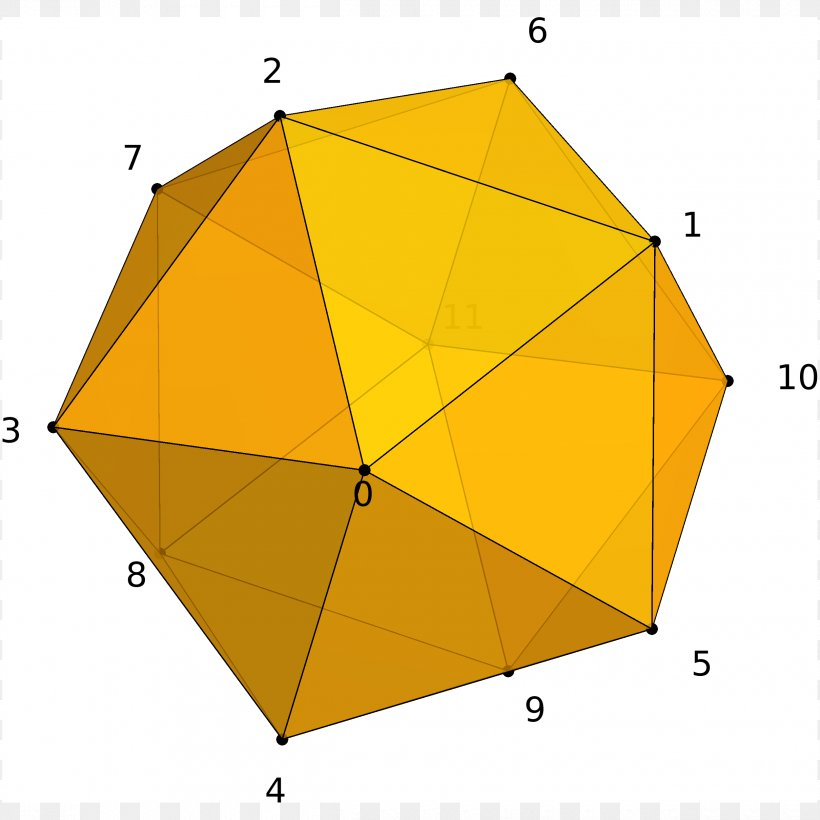


Atma Ram Sanatan Dharma College  
University of Delhi



Computer Graphics

Practical File for Paper Code 32341602

Submitted By  
Sudipto Ghosh  
College Roll No. 19/78003  
BSc (Hons) Computer Science

Submitted To  
Ms Manisha Bagri   
Department of Computer Science

# INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Objective | Date | Sign |
| 1 | Write a program to implement the DDA line drawing algorithm. |  |  |
| 2 | Write a program to implement Bresenham’s line drawing algorithm. |  |  |
| 3 | Write a program to implement Bresenham’s circle drawing algorithm. |  |  |
| 4 | Write a menu-driven program to perform 2D transformations. |  |  |
| 5 | Write a menu-driven program to perform 3D transformations on a 3D object and then apply parallel and perspective projection on it. |  |  |
| 6 | Write a program to clip a line using Cohen and Sutherland line clipping algorithm. |  |  |
| 7 | Write a program to clip a polygon using Sutherland Hodgman algorithm. |  |  |
| 8 | Write a program to draw Hermite/Bezier curve. |  |  |

# PRACTICAL 1

## Objective

Write a program to implement the DDA line drawing algorithm.

## Code

/\*\*

 \* Write a program to implement and draw a line using the Digital

 \* Differential Analyzer (DDA) algorithm.

 \*

 \* Written by Sudipto Ghosh for the University of Delhi

 \*/

#include <cmath>

#include <cstdlib>

#include <graphics.h>

#include <iostream>

using namespace std;

void ddaLine(int x0, int y0, int x1, int y1, int val)

{

  if (x0 == x1 && y0 == y1)

  {

    putpixel(x1, y1, val);

  }

  else

  {

    double x, y;

    int dx = x1 - x0;

    int dy = y1 - y0;

    bool isRTL = !(x1 > x0);

    float m = float(dy) / (float)(dx);

    if (abs(m) <= 1)

    {

      if (!isRTL)

      {

        for (x = x0, y = y0; x <= x1; x++)

        {

          putpixel(x, y, val);

          y += m;

        }

      }

      else

      {

        for (x = x1, y = y1; x >= x0; x--)

        {

          putpixel(x, y, val);

          y -= m;

        }

      }

    }

    else if (abs(m) > 1)

    {

      if (!isRTL)

      {

        for (x = x0, y = y0; y <= x1; y++)

        {

          putpixel(x, y, val);

          x += 1 / m;

        }

      }

      else

      {

        for (x = x1, y = y1; y >= x0; y--)

        {

          putpixel(x, y, val);

          x -= 1 / m;

        }

      }

    }

  }

  return;

}

int main(void)

{

  int x0, y0, x1, y1;

  cout << "Enter Left Endpoint (x0 y0): ";

  cin >> x0 >> y0;

  cout << "Enter Right Endpoint (x1 y1): ";

  cin >> x1 >> y1;

  cout << "Drawing Line..." << endl;

  int gd = DETECT, gm;

  initgraph(&gd, &gm, NULL);

  ddaLine(x0, y0, x1, y1, WHITE);

  delay(10e3);

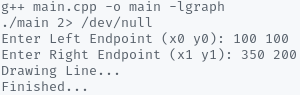
  closegraph();

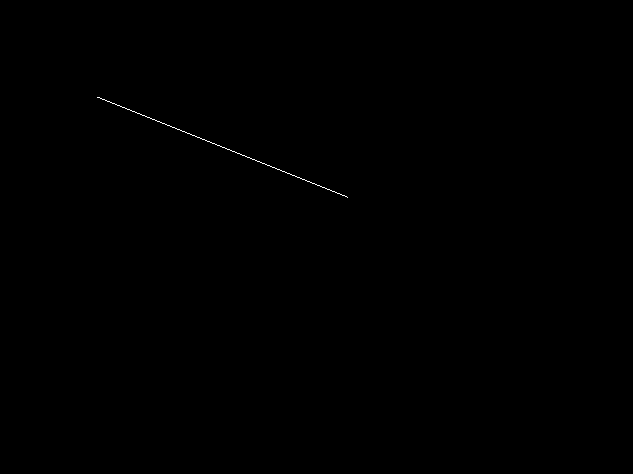
  cout << "Finished..." << endl;

  return 0;

}

## Output





# PRACTICAL 2

## Objective

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

## Code

/\*\*

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

 \*

 \* Written by Sudipto Ghosh for the University of Delhi

 \*/

#include <cmath>

#include <cstdlib>

#include <graphics.h>

#include <iostream>

using namespace std;

void bresenhamLine(int x0, int y0, int x1, int y1, int val)

{

  if (x0 == x1 && y0 == y1)

  {

    putpixel(x1, y1, val);

  }

  else

  {

    int dx = x1 - x0;

    int dy = y1 - y0;

    float m = float(dy) / (float)(dx);

    if (m >= 1 || m <= 0)

    {

      cout << "ERROR: Slope must be between 0 and 1." << endl;

      exit(1);

    }

    int d = 2 \* dy - dx;

    int del\_E = 2 \* dy;

    int del\_NE = 2 \* (dy - dx);

    int x = x0;

    int y = y0;

    putpixel(x, y, val);

    while (x < x1)

    {

      if (d <= 0)

      {

        d += del\_E;

        x += 1;

      }

      else

      {

        d += del\_NE;

        x += 1;

        y += 1;

      }

      putpixel(x, y, val);

    }

  }

  return;

}

int main(void)

{

  int x0, y0, x1, y1;

  cout << "Enter Left Endpoint (x0 y0): ";

  cin >> x0 >> y0;

  cout << "Enter Right Endpoint (x1 y1): ";

  cin >> x1 >> y1;

  cout << "Drawing Line..." << endl;

  int gd = DETECT, gm;

  initgraph(&gd, &gm, NULL);

  bresenhamLine(x0, y0, x1, y1, WHITE);

  delay(5e3);

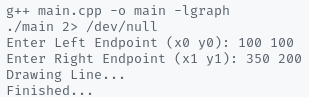
  closegraph();

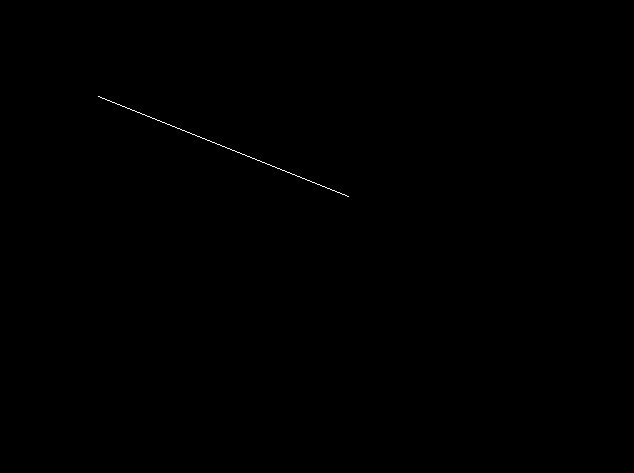
  cout << "Finished..." << endl;

  return 0;

}

## Output





# PRACTICAL 3

## Objective

Write a program to implement the Bresenham’s circle drawing algorithm.

## Code

#include <cmath>

#include <cstdlib>

#include <graphics.h>

#include <iostream>

using namespace std;

void drawCirclePoints(int x, int y, int val, int c\_x, int c\_y)

{

  putpixel(c\_x + x, c\_y + y, val);

  putpixel(c\_x + y, c\_y + x, val);

  putpixel(c\_x + y, c\_y + -x, val);

  putpixel(c\_x + x, c\_y + -y, val);

  putpixel(c\_x + -x, c\_y + -y, val);

  putpixel(c\_x + -y, c\_y + -x, val);

  putpixel(c\_x + -y, c\_y + x, val);

  putpixel(c\_x + -x, c\_y + y, val);

  return;

}

void midpointCircle(int r, int val, int c\_x = 0, int c\_y = 0)

{

  int x = 0;

  int y = r;

  int d = 1 - r;

  drawCirclePoints(x, y, val, c\_x, c\_y);

  while (y > x)

  {

    if (d < 0)

    {

      d += 2 \* x + 3;

      x += 1;

    }

    else

    {

      d += 2 \* (x - y) + 5;

      x += 1;

      y -= 1;

    }

    drawCirclePoints(x, y, val, c\_x, c\_y);

  }

  return;

}

int main(void)

{

  int x, y, r;

  cout << "Enter Centre (x y): ";

  cin >> x >> y;

  cout << "Enter Radius (r): ";

  cin >> r;

  cout << "Drawing Circle..." << endl;

  int gd = DETECT, gm;

  initgraph(&gd, &gm, NULL);

  midpointCircle(r, WHITE, x, y);

  delay(5e3);

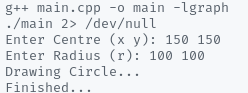
  closegraph();

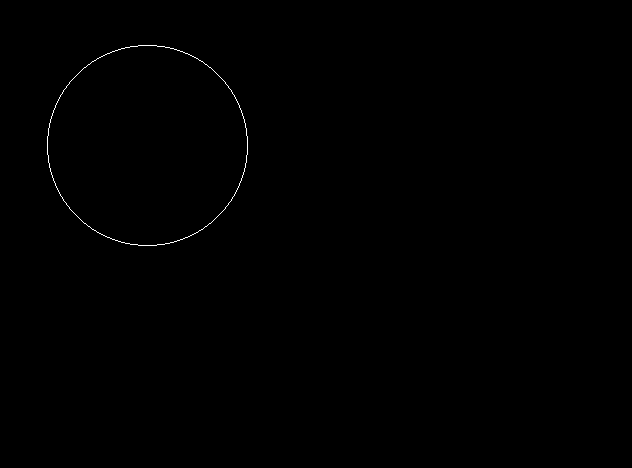
  cout << "Finished..." << endl;

  return 0;

}

## Output





# PRACTICAL 4

## Objective

Write a menu-driven program to perform 2D transformations.

## Code

/\*\*

 \* Write a menu-driven program to perform 2D transformations.

 \*

 \* Written by Sudipto Ghosh for the University of Delhi

 \*/

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <cstdlib>

#include <graphics.h>

#include <iostream>

#define COORD\_SHIFT 100

using namespace std;

void clrscr()

{

#ifdef \_WIN32

  system("cls");

#elif \_\_unix\_\_

  system("clear");

#endif

}

double \*\*inputFigure(int n)

{

  cout << "Enter the matrix for the 2-D shape (homogeneous):\n";

  double \*\*figure = NULL;

  figure = new double \*[n];

  for (int i = 0; i < n; i++)

  {

    figure[i] = new double[3];

    for (int j = 0; j < 3; j++)

    {

      cin >> figure[i][j];

    }

  }

  return figure;

}

void drawFigure(double \*\*points, int n)

{

  setcolor(WHITE);

  for (int i = 0; i < n; i++)

  {

    line(COORD\_SHIFT + points[i][0],

         COORD\_SHIFT + points[i][1],

         COORD\_SHIFT + points[(i + 1) % n][0],

         COORD\_SHIFT + points[(i + 1) % n][1]);

  }

  delay(5e3);

  cleardevice();

}

double \*\*translate(double \*\*figure, int dim, int m, int n)

{

  double \*\*\_figure = NULL;

  int T[dim][3] = {{1, 0, 0}, {0, 1, 0}, {m, n, 1}};

  \_figure = new double \*[dim];

  for (int i = 0; i < dim; i++)

  {

    \_figure[i] = new double[3];

    for (int j = 0; j < 3; j++)

    {

      for (int k = 0; k < dim; k++)

      {

        \_figure[i][j] += figure[i][k] \* T[k][j];

      }

    }

  }

  return \_figure;

}

double \*\*rotate(double \*\*figure, int dim, double theta)

{

  double \*\*\_figure = NULL;

  double T[dim][3] = {{cos(theta \* M\_PI / 180.0), sin(theta \* M\_PI / 180.0), 0},

                      {-sin(theta \* M\_PI / 180.0), cos(theta \* M\_PI / 180.0), 0},

                      {0, 0, 1}};

  \_figure = new double \*[dim];

  for (int i = 0; i < dim; i++)

  {

    \_figure[i] = new double[3];

    for (int j = 0; j < 2; j++)

    {

      for (int k = 0; k < dim; k++)

      {

        \_figure[i][j] += figure[i][k] \* T[k][j];

      }

    }

  }

  return \_figure;

}

double \*\*scale(double \*\*figure, int dim, int m, int n)

{

  double \*\*\_figure = NULL;

  int T[dim][3] = {{m, 0, 0}, {0, n, 0}, {0, 0, 1}};

  \_figure = new double \*[dim];

  for (int i = 0; i < dim; i++)

  {

    \_figure[i] = new double[3];

    for (int j = 0; j < 3; j++)

    {

      for (int k = 0; k < dim; k++)

      {

        \_figure[i][j] += figure[i][k] \* T[k][j];

      }

    }

  }

  return \_figure;

}

double \*\*reflect(double \*\*figure, int dim, int c)

{

  double \*\*\_figure = NULL;

  int T[dim][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

  switch (c)

  {

  case 1:

    T[1][1] = -1;

    break;

  case 2:

    T[0][0] = -1;

    break;

  case 3:

    T[0][0] = 0;

    T[0][1] = 1;

    T[1][0] = 1;

    T[1][1] = 0;

    break;

  case 4:

    T[0][0] = -1;

    T[1][1] = -1;

    break;

  default:

    return NULL;

    break;

  }

  \_figure = new double \*[dim];

  for (int i = 0; i < dim; i++)

  {

    \_figure[i] = new double[3];

    for (int j = 0; j < 3; j++)

    {

      for (int k = 0; k < dim; k++)

      {

        \_figure[i][j] += figure[i][k] \* T[k][j];

      }

    }

  }

  return \_figure;

}

double \*\*shear(double \*\*figure, int dim, int m, int n)

{

  double \*\*\_figure = NULL;

  int T[dim][3] = {{1, n, 0}, {m, 1, 0}, {0, 0, 1}};

  \_figure = new double \*[dim];

  for (int i = 0; i < dim; i++)

  {

    \_figure[i] = new double[3];

    for (int j = 0; j < 3; j++)

    {

      for (int k = 0; k < dim; k++)

      {

        \_figure[i][j] += figure[i][k] \* T[k][j];

      }

    }

  }

  return \_figure;

}

void menu(double \*\*figure, int dim)

{

  int ch = 0;

  double \*\*\_figure;

  do

  {

    clrscr();

    cout << "\nMenu\n-------\n(1) Translation\n(2) Rotation";

    cout << "\n(3) Scaling\n(4) Reflection\n(5) Shearing";

    cout << "\n(6) View Figure\n(7) Exit\n\nEnter Choice: ";

    cin >> ch;

    cout << endl;

    switch (ch)

    {

    case 1:

      int m, n;

      cout << "Enter translation in x-axis: ";

      cin >> m;

      cout << "Enter translation in y-axis: ";

      cin >> n;

      \_figure = translate(figure, dim, m, n);

      cout << "Drawing Original Figure...\n";

      drawFigure(figure, dim);

      cout << "Drawing Transformed Figure...\n";

      drawFigure(\_figure, dim);

      break;

    case 2:

      double theta;

      cout << "Enter rotation angle (degrees): ";

      cin >> theta;

      \_figure = rotate(figure, dim, theta);

      cout << "Drawing Original Figure...\n";

      drawFigure(figure, dim);

      cout << "Drawing Transformed Figure...\n";

      drawFigure(\_figure, dim);

      break;

    case 3:

      cout << "Enter scaling in x-axis: ";

      cin >> m;

      cout << "Enter scaling in y-axis: ";

      cin >> n;

      \_figure = scale(figure, dim, m, n);

      cout << "Drawing Original Figure...\n";

      drawFigure(figure, dim);

      cout << "Drawing Transformed Figure...\n";

      drawFigure(\_figure, dim);

      break;

    case 4:

      cout << "Reflect along\n(1) x-axis\n(2) y-axis\n(3) y = x\n(4) y = -x\n"

           << "\nEnter Choice: ";

      cin >> m;

      \_figure = reflect(figure, dim, m);

      cout << "Drawing Original Figure...\n";

      drawFigure(figure, dim);

      cout << "Drawing Transformed Figure...\n";

      drawFigure(\_figure, dim);

      break;

    case 5:

      cout << "Enter shearing in x-axis: ";

      cin >> m;

      cout << "Enter shearing in y-axis: ";

      cin >> n;

      \_figure = shear(figure, dim, m, n);

      cout << "Drawing Original Figure...\n";

      drawFigure(figure, dim);

      cout << "Drawing Transformed Figure...\n";

      drawFigure(\_figure, dim);

      break;

    case 6:

      cout << "Drawing Original Figure...\n";

      drawFigure(figure, dim);

      break;

    case 7:

    default:

      break;

    }

    delete \_figure;

    cout << endl

         << "Finished..."

         << endl;

    if (ch != 7)

    {

      cout << "\nPress Enter to continue ...\n";

      cin.ignore();

      cin.get();

    }

  } while (ch != 7);

};

int main(void)

{

  int n;

  double \*\*fig;

  int gd = DETECT, gm;

  initgraph(&gd, &gm, NULL);

  cout << "Enter number of points in the figure: ";

  cin >> n;

  fig = inputFigure(n);

  menu(fig, n);

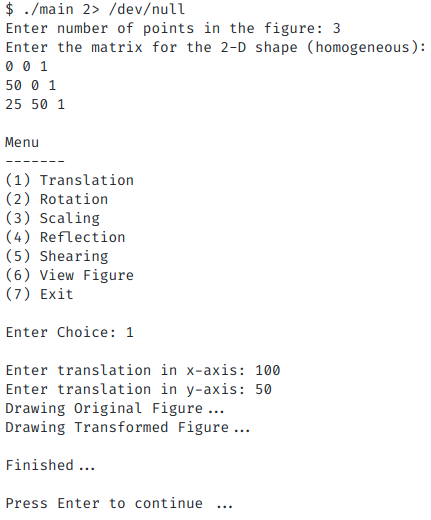
  delete fig;

  closegraph();

  return 0;

}

## Output



|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

# PRACTICAL 5

## Objective

Write a menu-driven program to perform 3D transformations on a 3D object and then apply parallel and perspective projection on it.

## Code

#define \_USE\_MATH\_DEFINES

#include <conio.h>

#include <dos.h>

#include <graphics.h>

#include <iostream.h>

#include <math.h>

#include <process.h>

#include <stdio.h>

int gd = DETECT, gm;

double x1, x2, y1, y2;

void drawObj(double edges[20][3])

{

  int choice, i;

  double p, q, r, temp, temp1, theta;

  double \_edges[20][3];

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] = edges[i][0];

    \_edges[i][1] = edges[i][1];

    \_edges[i][2] = edges[i][2];

  }

  cout << "\nProjection:" << endl;

  cout << "1. Orthographic Projection on xy-plane" << endl;

  cout << "2. Axonometric Projection (Isometric)" << endl;

  cout << "3. Perspective Projection" << endl;

  cout << "\nEnter your choice: ";

  cin >> choice;

  initgraph(&gd, &gm, "..\\bgi");

  switch (choice)

  {

  case 1:

    // Orthographic Parallel Projection - xy plane

    for (i = 0; i < 19; i++)

    {

      x1 = edges[i][0];

      y1 = edges[i][1];

      x2 = edges[i + 1][0];

      y2 = edges[i + 1][1];

      line(x1 + 320, 240 - y1, x2 + 320, 240 - y2);

    }

    break;

  case 2:

    // Axonometric Projection - Isometric

    for (i = 0; i < 19; i++)

    {

      x1 = edges[i][0] + edges[i][2] \* (cos(2.3562));

      y1 = edges[i][1] - edges[i][2] \* (sin(2.3562));

      x2 = edges[i + 1][0] + edges[i + 1][2] \* (cos(2.3562));

      y2 = edges[i + 1][1] - edges[i + 1][2] \* (sin(2.3562));

      line(x1 + 320, 240 - y1, x2 + 320, 240 - y2);

    }

    setcolor(YELLOW);

    line(320, 240, 320, 25);

    line(320, 240, 550, 240);

    line(320, 240, 150, 410);

    setcolor(WHITE);

    break;

  case 3:

    // Perspective Projection

    cout << "\nEnter p, q and r: ";

    cin >> p >> q >> r;

    for (i = 0; i < 20; i++)

    {

      \_edges[i][0] /= (p \* \_edges[i][0] +

                       q \* \_edges[i][1] +

                       r \* \_edges[i][2] + 1);

      \_edges[i][1] /= (p \* \_edges[i][0] +

                       q \* \_edges[i][1] +

                       r \* \_edges[i][2] + 1);

      \_edges[i][2] /= (p \* \_edges[i][0] +

                       q \* \_edges[i][1] +

                       r \* \_edges[i][2] + 1);

    }

    for (i = 0; i < 19; i++)

    {

      x1 = \_edges[i][0] + \_edges[i][2] \* (cos(2.3562));

      y1 = \_edges[i][1] - \_edges[i][2] \* (sin(2.3562));

      x2 = \_edges[i + 1][0] + \_edges[i + 1][2] \* (cos(2.3562));

      y2 = \_edges[i + 1][1] - \_edges[i + 1][2] \* (sin(2.3562));

      line(x1 + 320, 240 - y1, x2 + 320, 240 - y2);

    }

    break;

  }

  getch();

  closegraph();

}

void scale(double edges[20][3])

{

  int i;

  double a, b, c;

  double \_edges[20][3];

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] = edges[i][0];

    \_edges[i][1] = edges[i][1];

    \_edges[i][2] = edges[i][2];

  }

  cout << "Enter Scaling Factors (in x y z): ";

  cin >> a >> b >> c;

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] \*= a;

    \_edges[i][1] \*= b;

    \_edges[i][2] \*= c;

  }

  drawObj(\_edges);

}

void translate(double edges[20][3])

{

  int i, a, b, c;

  double \_edges[20][3];

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] = edges[i][0];

    \_edges[i][1] = edges[i][1];

    \_edges[i][2] = edges[i][2];

  }

  cout << "\nEnter Translation Factors (in x y z): ";

  cin >> a >> b >> c;

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] += a;

    \_edges[i][0] += b;

    \_edges[i][0] += c;

  }

  drawObj(\_edges);

}

void rotate(double edges[20][3])

{

  int i, ch;

  double temp, theta, temp1;

  double \_edges[20][3];

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] = edges[i][0];

    \_edges[i][1] = edges[i][1];

    \_edges[i][2] = edges[i][2];

  }

  cout << "\nRotation About:" << endl;

  cout << "1. x-axis " << endl;

  cout << "2. z-axis" << endl;

  cout << "3. y-axis " << endl;

  cout << "Enter Choice: ";

  cin >> ch;

  cout << "\nEnter Angle: ";

  cin >> theta;

  theta = (theta \* M\_PI) / 180;

  switch (ch)

  {

  case 1:

    for (i = 0; i < 20; i++)

    {

      \_edges[i][0] = \_edges[i][0];

      temp = \_edges[i][1];

      temp1 = \_edges[i][2];

      \_edges[i][1] = temp \* cos(theta) - temp1 \* sin(theta);

      \_edges[i][2] = temp \* sin(theta) + temp1 \* cos(theta);

    }

    break;

  case 2:

    for (i = 0; i < 20; i++)

    {

      \_edges[i][1] = \_edges[i][1];

      temp = \_edges[i][0];

      temp1 = \_edges[i][2];

      \_edges[i][0] = temp \* cos(theta) + temp1 \* sin(theta);

      \_edges[i][2] = -temp \* sin(theta) + temp1 \* cos(theta);

    }

    break;

  case 3:

    for (i = 0; i < 20; i++)

    {

      \_edges[i][2] = \_edges[i][2];

      temp = \_edges[i][0];

      temp1 = \_edges[i][1];

      \_edges[i][0] = temp \* cos(theta) - temp1 \* sin(theta);

      \_edges[i][1] = temp \* sin(theta) + temp1 \* cos(theta);

    }

    break;

  }

  drawObj(\_edges);

}

void reflect(double edges[20][3])

{

  int i, ch;

  double \_edges[20][3];

  for (i = 0; i < 20; i++)

  {

    \_edges[i][0] = edges[i][0];

    \_edges[i][1] = edges[i][1];

    \_edges[i][2] = edges[i][2];

  }

  cout << "\nReflection About:" << endl;

  cout << "1. x-axis " << endl;

  cout << "2. y-axis" << endl;

  cout << "3. z-axis " << endl;

  cout << "Enter Choice: ";

  cin >> ch;

  clrscr();

  switch (ch)

  {

  case 1:

    for (i = 0; i < 20; i++)

    {

      \_edges[i][0] = \_edges[i][0];

      \_edges[i][1] = -\_edges[i][1];

      \_edges[i][2] = -\_edges[i][2];

    }

    break;

  case 2:

    for (i = 0; i < 20; i++)

    {

      \_edges[i][1] = \_edges[i][1];

      \_edges[i][0] = -\_edges[i][0];

      \_edges[i][2] = -\_edges[i][2];

    }

    break;

  case 3:

    for (i = 0; i < 20; i++)

    {

      \_edges[i][2] = \_edges[i][2];

      \_edges[i][0] = -\_edges[i][0];

      \_edges[i][1] = -\_edges[i][1];

    }

    break;

  }

  drawObj(\_edges);

}

void main()

{

  int choice;

  double edges[20][3] = {100, 0, 0,

                         100, 100, 0,

                         0, 100, 0,

                         0, 100, 100,

                         0, 0, 100,

                         0, 0, 0,

                         100, 0, 0,

                         100, 0, 100,

                         100, 75, 100,

                         75, 100, 100,

                         100, 100, 75,

                         100, 100, 0,

                         100, 100, 75,

                         100, 75, 100,

                         75, 100, 100,

                         0, 100, 100,

                         0, 100, 0,

                         0, 0, 0,

                         0, 0, 100,

                         100, 0, 100};

  while (1)

  {

    clrscr();

    cout << "\nMenu\n-------\n(1) Translation\n(2) Rotation";

    cout << "\n(3) Scaling\n(4) Reflection\n(5) View Figure";

    cout << "\n(6) Exit\n\nEnter Choice: ";

    cin >> choice;

    switch (choice)

    {

    case 1:

      translate(edges);

      break;

    case 2:

      rotate(edges);

      break;

    case 3:

      scale(edges);

      break;

    case 4:

      reflect(edges);

      break;

    case 5:

      drawObj(edges);

      break;

    case 6:

      exit(0);

    default:

      break;

    }

  }

  closegraph();

}

## Output

Text

Description automatically generated

Shape

Description automatically generated

Diagram

Description automatically generated

Shape

Description automatically generated

Shape

Description automatically generated

A picture containing graphical user interface

Description automatically generated

A picture containing shape

Description automatically generated

A picture containing engineering drawing

Description automatically generated

Text

Description automatically generated

A picture containing application

Description automatically generated

# PRACTICAL 6

## Objective

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

## Code

#include <conio.h>

#include <graphics.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

typedef unsigned int outcode;

enum

{

  TOP = 0x1,

  BOTTOM = 0x2,

  RIGHT = 0x4,

  LEFT = 0x8

};

outcode computeOutcode(double x, double y, double xmin, double xmax, double ymin, double ymax)

{

  outcode code = 0;

  if (y > ymax)

    code |= TOP;

  else if (y < ymin)

    code |= BOTTOM;

  if (x > xmax)

    code |= RIGHT;

  else if (x < xmin)

    code |= LEFT;

  return code;

}

void clipLine(double x0, double yo, double x1, double y1, double xmin, double xmax, double ymin, double ymax)

{

  int accept = 0, done = 0;

  outcode outcode0, outcode1, outcodeout;

  outcode0 = computeOutcode(x0, yo, xmin, xmax, ymin, ymax);

  outcode1 = computeOutcode(x1, y1, xmin, xmax, ymin, ymax);

  do

  {

    if (!(outcode0 | outcode1))

    {

      accept = 1;

      done = 1;

    }

    else if (outcode0 & outcode1)

    {

      done = 1;

    }

    else

    {

      double x, y;

      outcodeout = outcode0 ? outcode0 : outcode1;

      if (outcodeout & TOP)

      {

        x = x0 + (ymax - yo) \* (x1 - x0) / (y1 - yo);

        y = ymax;

      }

      else if (outcodeout & BOTTOM)

      {

        x = x0 + (ymin - yo) \* (x1 - x0) / (y1 - yo);

        y = ymin;

      }

      else if (outcodeout & LEFT)

      {

        y = yo + (xmin - x0) \* (y1 - yo) / (x1 - x0);

        x = xmin;

      }

      else

      {

        y = yo + (xmax - x0) \* (y1 - yo) / (x1 - x0);

        x = xmax;

      }

      if (outcodeout == outcode0)

      {

        x0 = x;

        yo = y;

        outcode0 = computeOutcode(x0, yo, xmin, xmax, ymin, ymax);

      }

      else

      {

        x1 = x;

        y1 = y;

        outcode1 = computeOutcode(x1, y1, xmin, xmax, ymin, ymax);

      }

    }

  } while (done == 0);

  if (accept)

    line(x0, yo, x1, y1);

}

int main(void)

{

  int gd = DETECT, gm;

  double x0, x1, y0, y1;

  double xmin, ymin, xmax, ymax;

  initgraph(&gd, &gm, "..\\BGI");

  cout << "Enter Point A (x0, y0): ";

  cin >> x0 >> y0;

  cout << "Enter Point B (x1, y1): ";

  cin >> x1 >> y1;

  cout << "Enter Bounds of Clipping Rectangle : ";

  cout << "\n\txmin: ";

  cin >> xmin;

  cout << "\tymin: ";

  cin >> ymin;

  cout << "\txmax: ";

  cin >> xmax;

  cout << "\tymax: ";

  cin >> ymax;

  cleardevice();

  line(xmin, ymin, xmax, ymin);

  line(xmin, ymin, xmin, ymax);

  line(xmin, ymax, xmax, ymax);

  line(xmax, ymin, xmax, ymax);

  line(x0, y0, x1, y1);

  getch();

  cleardevice();

  line(xmin, ymin, xmax, ymin);

  line(xmin, ymin, xmin, ymax);

  line(xmin, ymax, xmax, ymax);

  line(xmax, ymin, xmax, ymax);

  clipLine(x0, y0, x1, y1, xmin, xmax, ymin, ymax);

  getch();

  closegraph();

  return 0;

}

## Output

]

Text

Description automatically generated

A picture containing shape

Description automatically generated

Graphical user interface, application

Description automatically generated

# PRACTICAL 7

## Objective

Write a program to clip a polygon using Sutherland Hodgman algorithm.

## Code

#include <conio.h>

#include <graphics.h>

#include <iostream.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

typedef unsigned int outcode;

outcode compOutcode(double x, double y);

enum

{

  TOP = 0x1,

  BOTTOM = 0x2,

  RIGHT = 0x4,

  LEFT = 0x8

};

double xmin, xmax, ymin, ymax;

outcode compOutcode(double x, double y)

{

  outcode code = 0;

  if (y > ymax)

    code |= TOP;

  else if (y < ymin)

    code |= BOTTOM;

  if (x > xmax)

    code |= RIGHT;

  else if (x < xmin)

    code |= LEFT;

  return code;

}

void clipPolygon(int x0, int y0, int x1, int y1)

{

  int accept = 0, done = 0;

  outcode outcode0, outcode1, outcodeOut;

  outcode0 = compOutcode(x0, y0);

  outcode1 = compOutcode(x1, y1);

  do

  {

    if (!(outcode0 | outcode1))

    {

      accept = 1;

      done = 1;

    }

    else if (outcode0 & outcode1)

      done = 1;

    else

    {

      double x, y;

      outcodeOut = outcode0 ? outcode0 : outcode1;

      if (outcodeOut & TOP)

      {

        x = x0 + (x1 - x0) \* (ymax - y0) / (y1 - y0);

        y = ymax;

      }

      else if (outcodeOut & BOTTOM)

      {

        x = x0 + (x1 - x0) \* (ymin - y0) / (y1 - y0);

        y = ymin;

      }

      else if (outcodeOut & RIGHT)

      {

        y = y0 + (y1 - y0) \* (xmax - x0) / (x1 - x0);

        x = xmax;

      }

      else

      {

        y = y0 + (y1 - y0) \* (xmin - x0) / (x1 - x0);

        x = xmin;

      }

      if (outcodeOut == outcode0)

      {

        x0 = x;

        y0 = y;

        outcode0 = compOutcode(x0, y0);

      }

      else

      {

        x1 = x;

        y1 = y;

        outcode1 = compOutcode(x1, y1);

      }

    }

  } while (done == 0);

  if (accept)

    line(x0, y0, x1, y1);

}

void main()

{

  int i, n;

  int gd = DETECT, gm;

  int poly[24];

  initgraph(&gd, &gm, "..\\BGI");

  cout << "Enter Bounds of Clipping Rectangle : ";

  cout << "\n\txmin: ";

  cin >> xmin;

  cout << "\tymin: ";

  cin >> ymin;

  cout << "\txmax: ";

  cin >> xmax;

  cout << "\tymax: ";

  cin >> ymax;

  cout << "Enter Number of Edges in Polygon : ";

  cin >> n;

  cout << "Enter Coordinates of the Polygon : ";

  for (i = 0; i < 2 \* n; i++)

    cin >> poly[i];

  poly[2 \* n] = poly[0];

  poly[2 \* n + 1] = poly[1];

  cleardevice();

  rectangle(xmin, ymin, xmax, ymax);

  drawpoly(n + 1, poly);

  getch();

  cleardevice();

  rectangle(xmin, ymin, xmax, ymax);

  for (i = 0; i < n; i++)

    clipPolygon(poly[2 \* i], poly[(2 \* i) + 1], poly[(2 \* i) + 2], poly[(2 \* i) + 3]);

  getch();

  closegraph();

}

## Output

Text

Description automatically generatedGraphical user interface

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generated

# PRACTICAL 8

## Objective

Write a program to draw Hermite/Bezier curve.

## Code

### Hermite Curve

#include <conio.h>

#include <graphics.h>

#include <iostream.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

struct point

{

  int x, y;

};

void hermite(point p1, point p4, double r1, double r4)

{

  float x, y, t;

  for (t = 0.0; t <= 1.0; t += 0.00005)

  {

    x = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* p1.x +

        (-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* p4.x +

        (pow(t, 3) - 2 \* pow(t, 2) + t) \* r1 +

        (pow(t, 3) - pow(t, 2)) \* r4;

    y = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* p1.y +

        (-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* p4.y +

        (pow(t, 3) - 2 \* pow(t, 2) + 1) \* r1 +

        (pow(t, 3) - pow(t, 2)) \* r4;

    putpixel(x, y, WHITE);

  }

  circle(p1.x, p1.y, 3);

  circle(p4.x, p4.y, 3);

}

void main()

{

  point p1, p4;

  double r1, r4;

  int gd = DETECT, gm;

  initgraph(&gd, &gm, "..\\BGI");

  cout << "Enter Point 1 (x, y): ";

  cin >> p1.x >> p1.y;

  cout << "Enter Point 2 (x, y): ";

  cin >> p4.x >> p4.y;

  cout << "Enter Tangent at Point 1: ";

  cin >> r1;

  cout << "Enter Tangent at Point 4: ";

  cin >> r4;

  hermite(p1, p4, r1, r4);

  getch();

  closegraph();

}

### Bezier Curve

#include <conio.h>

#include <graphics.h>

#include <iostream.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

void bezier(int x[4], int y[4])

{

  for (double t = 0.0; t < 1.0; t += 0.00005)

  {

    double xt = pow(1 - t, 3) \* x[0] + 3 \* t \* pow(1 - t, 2) \* x[1] + 3 \* pow(t, 2) \* (1 - t) \* x[2] + pow(t, 3) \* x[3];

    double yt = pow(1 - t, 3) \* y[0] + 3 \* t \* pow(1 - t, 2) \* y[1] + 3 \* pow(t, 2) \* (1 - t) \* y[2] + pow(t, 3) \* y[3];

    putpixel(xt, yt, WHITE);

  }

  for (int i = 0; i < 4; i++)

  {

    circle(x[i], y[i], 3);

  }

  getch();

  closegraph();

  return;

}

void main()

{

  int i;

  int x[4], y[4];

  int gd = DETECT, gm, errorcode;

  initgraph(&gd, &gm, "..\\BGI");

  for (i = 0; i < 4; i++)

  {

    cout << "Enter Point " << i + 1 << " (x, y): ";

    cin >> x[i] >> y[i];

  }

  bezier(x, y);

  return;

}

## Output

### Hermite Curve

Text

Description automatically generated

### Bezier Curve

A picture containing diagram

Description automatically generated