Certificate

Name: Richa Mishra Class: 7C(3)

Roll No: 1RV16CS194 Exam No: Lab

Institution R.V. College of Engineering

This is certified to be the bonafide work of the student in the Computer Graphics ~~Data Science & ML~~ *Laboratory during the academic year 20 20/ 20 21.*

No. of practicals certified _____ *out of* _____ *in the subject of* Computer Graphics ~~Data Science & Machine learning~~

.....
Teacher In-charge

.....
Examiner's Signature

.....
Principal

Date: .....

Institution Rubber Stamp

(N.B: The candidate is expected to retain his/her journal till he/she passes in the subject.)

Expt. No. _____ Page No. _____

Program - 1

Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one and slopes less than one. User can specify inputs through Keyboard/mouse.

```
#include <iostream>
#include <GLUT/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}
void draw_line()
{
    int dx, dy, i, e;
    int incx, incy, inc1, inc2;
    int x, y;
    dx = x2 - x1;
    dy = y2 - yc1;
    if (dx < 0) dx = -dx;
    if (dy < 0) dy = -dy;
```

Teacher's Signature _____

```
#include <iostream>
#include <GLUT/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{
        glColor3f(1, 0, 0);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
        glFlush();

}
void draw_line()
{
        int dx, dy, i, e;
        int incx, incy, inc1, inc2;
        int x, y;
        dx = x2 - x1;
        dy = y2 - yc1;
        if (dx < 0)dx = -dx;
        if (dy < 0)dy = -dy;
        incx = 1;
        if (x2 < x1)
                incx = -1;
         incy = 1;
```

Expt. No. _____ Date _____

Page No. _____

Program-1

Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one and slopes less than one. User can specify inputs through keyboard/mouse.

```
#include <iostream>
#include <GLUT/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}
void draw_line()
{
    int dx, dy, i, e;
    int incx, incy, inc1, inc2;
    int x, y;
    dx = x2 - x1;
    dy = y2 - yc1;
    if (dx < 0) dx = -dx;
    if (dy < 0) dy = -dy;
```

Teacher's Signature _____

```
#include <iostream>
#include <GLUT/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{
        glColor3f(1, 0, 0);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
        glFlush();

}
void draw_line()
{
        int dx, dy, i, e;
        int incx, incy, inc1, inc2;
        int x, y;
        dx = x2 - x1;
        dy = y2 - yc1;
        if (dx < 0)dx = -dx;
        if (dy < 0)dy = -dy;
        incx = 1;
        if (x2 < x1)
                incx = -1;
        incy = 1;
        if (y2 < yc1)
                incy = -1;
        x = x1;
        y = yc1;
        if (dx > dy)
        {
                draw_pixel(x, y);
                e = 2 * dy - dx;
                inc1 = 2 * (dy - dx);
                inc2 = 2 * dy;
                for (i = 0; i < dx; i++)
                {
                        if (e > 0)
                        {
                                y += incy;
                                e += inc1;
                        }
                        else
                                e += inc2;
                        x += incx;
                        draw_pixel(x, y);
                }
        }
        else
        {

                draw_pixel(x, y);
                e = 2 * dx - dy;
```

```
                inc1 = 2 * (dx - dy);
                inc2 = 2 * dx;
                for (i = 0; i < dy; i++)
                {
                        if (e > 0)
                        {
                                x += incx;
                                e += inc1;

                        }
                        else

                                e += inc2;
                        y += incy;

                        draw_pixel(x, y);
                }
        }
        glFlush();
}
void myinit()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1, 1, 1, 1);
        gluOrtho2D(-250, 250, -250, 250);
}
void MyMouse(int button, int state, int x, int y)
{
        switch (button)
        {
        case GLUT_LEFT_BUTTON:
                if (state == GLUT_DOWN)
                {
                        if (flag == 0)
                        {
                                printf("Defining x1,y1");
                                x1 = x - 250;
                                yc1 = 250 - y;
                                flag++;
                                cout << x1 << " " << yc1 << " \n";
                        }
                        else
                        {
                                printf("Defining x2,y2");
                                x2 = x - 250;
                                y2 = 250 - y;
                                flag = 0;
                                cout << x2 << " " << y2 << " \n";
                                draw_line();

                        }

                }
                break;
        }
}
void display()
```

```
t main(int ac, char* av[])

    glutInit(&ac, av);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 200);
    glutCreateWindow("LINE");
    myinit();
    glutMouseFunc(MyMouse);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Expt. No. _____ Date _____

Page No. _____

```
incx = 1;
if (x2 < x1)
        incx = -1;
incy = 1;
if (y2 < yc1)
        incy = -1;
x = x1;
y = yc1;
if (dx > dy)
{
        drawpixel(x, y);
        e = 2* dy - dx;
        inc1 = 2* (dy - dx);
        inc2 = 2* dy;
        for (i=0; i<dx; i++)
        {
                if(e>0)
                {
                        y += incy;
                        e += inc1;
                }
                else
                        e += inc2;
                x += incx;
                drawpixel(x, y);
        }
}
else
{
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
draw_pixel(x,y);
e = 2*dx-dy;
inc1 = 2*(dx-dy);
inc2 = 2*dx;
for(i=0; i<dy; i++)
{
    if(e>0)
    {
        x += incx;
        e += inc1;
    }
    else
        e += inc2;
    y += incy;
    draw_pixel(x,y);
}
}
glFlush();
}
void myinit()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1,1,1,1);
    gluOrtho2D(-250,250,-250,250);
}
void MyMouse(int button, int state, int x, int y)
{
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
switch (button)
{
 case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
        {
            if (flag==0)
            {
                printf("Defining x1, y1");
                x1 = x - 250;
                yc1 = 250 - y;
                flag++;
                cout<<x1<<" "<<yc1
                    <<"\n";
            }
            else
            {
                print f("Defining x2, y2");
                x2 = x - 250;
                y2 = 250 - y;
                flag = 0;
                cout<<x2<<" "<<y2<<"\n";
                draw_line();
            }
        }
    }
    break;
}
```

Teacher's Signature _____

Date

Expt. No. Page No.

```
}
void display()
{}
int main(int ac, char * av[])
{
        glut Init(&ac, av);
        glut InitDisplay Mode(GLUT_SINGLE |
                              GLUT_RGB);
        glut InitWindowSize(500, 500);
        glut Init Window Position(100, 200);
        glut Create Window("LINE");
        my init();
        glut Mouse Func(My Mouse);
        glut DisplayFunc(display);
        glut Main Loop();
}
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

Program-2

Write a program to generate a circle using Bresenham's circle drawing technique. User can specify inputs through keyboard / mouse.

```
#include <GLUT/GLUT.h>
#include <studio.h>
#include <math.h>
int xc, yc, r;
int rx, ry, xce, yce;
void draw_circle(int xc, int yc, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(xc + x, yc + y);
    glVertex2i(xc - x, yc + y);
    glVertex2i(xc + x, yc - y);
    glVertex2i(xc - x, yc - y);
    glVertex2i(xc + y, yc + x);
    glVertex2i(xc - y, yc + x);
    glVertex2i(xc + y, yc - x);
    glVertex2i(xc - y, yc - x);
    glEnd();
}
void circlebres()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int x = 0, y = r;
    int d = 3 - 2 * r;
    while (x <= y)
```

Teacher's Signature _____

```
#include<GLUT/GLUT.h>
#include<stdlib.h>
#include<algorithm>
#include<iostream>


using namespace std;
float x[100], y[100];

int n, m;
int wx = 500, wy = 500;
static float intx[10] = { 0 };

void draw_line(float x1, float y1, float x2, float y2) {

    glColor3f(1, 0, 0);
    glBegin(GL_LINES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd();
    glFlush();

}

void edgeDetect(float x1, float y1, float x2, float y2, int scanline)
{

    float temp;
    if (y2 < y1) {

        temp = x1; x1 = x2; x2 = temp;
        temp = y1; y1 = y2; y2 = temp;
    }

    if (scanline > y1 && scanline < y2)
        intx[m++] = x1 + (scanline - y1) * (x2 - x1) / (y2 - y1);
}

void scanfill(float x[], float y[]) {
    for (int s1 = 0; s1 <= wy; s1++) {
        m = 0;
        for (int i = 0; i < n; i++) {

            edgeDetect(x[i], y[i], x[(i + 1) % n], y[(i + 1) % n],
s1);
        }
        sort(intx, (intx + m));
        if (m >= 2)
            for (int i = 0; i < m; i = i + 2)
                draw_line(intx[i], s1, intx[i + 1], s1);

    }

}

void display_filled_polygon() {
```

```
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(2);
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < n; i++)
        glVertex2f(x[i], y[i]);
    glEnd();
    scanfill(x, y);

}

void myInit() {

    glClearColor(1, 1, 1, 1);
    glColor3f(0, 0, 1);
    glPointSize(1);

    gluOrtho2D(0, wx, 0, wy);

}
int main(int ac, char* av[]) {
    glutInit(&ac, av);
    printf("Enter no. of sides: \n");
    scanf("%d", &n);
    printf("Enter coordinates of endpoints: \n");
    for (int i = 0; i < n; i++)
    {
        printf("X-coord Y-coord: \n");
        scanf("%f %f", &x[i], &y[i]);
    }
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("scanline");
    glutDisplayFunc(display_filled_polygon);
    myInit();
    glutMainLoop();

}
```

Expt. No. Date

Page No.

```
{
    draw_circle(xc, yc, x, y);
    x++;
    if (d<0)
        d=d + 4* x+6;
    else
    {
        y--;
        d=d+4*(x-y)+10;
    }
    draw_circle(xc, yc, x, y);
}
    glFlush();
}
int p1_x, p2_x, p1_y, p2_y;
int point1_done = 0;
void myMouseFunccircle(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) && point1_done == 0)
    {
        p1_x = x - 250;
        p1_y = 250 - y;
        point1_done = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        p2_x = x - 250;
```

Expt. No. _____ Date _____

Page No. _____

```
        p2_y = 250 - y;
        xc = p1_x;
        yc = p1_y;
        float exp = (p2_x - p1_x) * (p2_x - p1_x)
        + (p2_y - p1_y) * (p2_y - p1_y);
        r = (int)(sqrt(exp));
        circlebres();
        point1_done = 0;
    }
}

void draw_ellipse(int xce, int yce, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x + xce, y + yce);
    glVertex2i(-x + xce, y + yce);
    glVertex2i(x + xce, -y + yce);
    glVertex2i(-x + xce, -y + yce);
    glEnd();
}

void midptellipse()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float dx, dy, d1, d2, x, y;
    x = 0;
    y = ry;
```

Teacher's

Expt. No. _____ Date _____

Page No. _____

```
d1 = (ry * ry) - (rx * rx * ry) +
     (0.25 * rx * rx);
dx = 2 * ry * ry * x;
dy = 2 * rx * rx * y;
while (dx < dy)
{

  draw ellipse (xce, yce, x, y);

  if (d1 < 0)
  {
    x++;
    dx = dx + (2 * ry * ry);
    d1 = d1 + dx + (ry * ry);
  }
  else
  {
    x++;
    y--;
    dx = dx + (2 * ry * ry);
    dy = dy - (2 * rx * rx);
    d1 = d1 + dx - dy + (ry * ry);
  }
}
```

Expt. No. _____ Date _____

Page No. _____

```
d2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) +
     ((rx * rx) * ((y - 1) * (y - 1))) -
     (rx * rx * ry * ry);

while (y > 0)
{

    draw_ellipse (xce, yce, x, y);

    if (d2 > 0)
    {
        y--;
        dy = dy - (2 * rx * rx);
        d2 = d2 + (rx * rx) - dy;
    }
    else
    {
        y--;
        x++;
        dx = dx + (2 * ry * ry);
        dy = dy - (2 * rx * rx);
        d2 = d2 + dx - dy + (rx * rx);
    }
}
glFlush ();
}
```

Teacher's Signature _____

Expt. No. Date

Page No.

```
int p1e_x, p2e_x, p1e_y, p2e_y, p3e_x, p3e_y;
int pointle_done = 0;
void myMouseFunc(int button, int state, int x, int y)
{
    if( button == GLUT_LEFT_BUTTON && state == GLUT_
    DOWN && pointle_down == 0)
    {
        p1e_x = x-250;
        p1e_y = 250 - y;
        xce = p1e_x;
        yce = p1e_y;
    pointle_done = 1;
    }
    else if( button == GLUT_LEFT_BUTTON && state ==
    GLUT_DOWN && pointle_done == 1)
    {
        p2e_x = x - 250;
        p2e_y = 250 - y;
        float exp = (p2e_x - p1e_x) * (p2e_x - p1ex)
        + (p2e_y - p1e_y) * (p2e_y - p1e_y);
        rx = (int) (sqrt(exp));

    pointle_done = 2;
    }
    else if (button == GLUT_LEFT_BUTTON && state
    == GLUT_DOWN && pointle_done == 2)
    {
        p3e_x = x - 250;
        p3e_y = 250 - y;
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

```
    float exp = (p3e_x - p1e_x)*(p3e_x - p1e_x) +
    (p3e_y - p1e_y) * (p3e_y - p1e_y);
    ry = (int) (sqrt (exp));
    midptellipse ();
    pointle_done = 0;
  }
}
void myDrawing ()
 {}
void myDrawingc()
 {}

Void minit()
{
    glClearColor(1,1,1,1);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize (3.0);
    gluOrtho2D (-250, 250, -250, 250);
}
&
int main (int argc, char* argv [])
{

    glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (0, 0);

   printf ("Enter 1 to draw Circle, 2 to draw ellipse\n");
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
int ch;
scanf("%d", &ch);
switch(ch){
case 1:
printf("Enter coordinates of centre of circle and radius\n");
scanf("%d %d %d", &xc, &yc, &r);
glutCreateWindow("circle");
glutDisplayFunc(circlebres);
break;
case 2:
printf("Enter coordinates of centre of ellipse and major
and minor radius\n");
scanf("%d %d %d %d", &xce, &yce, &rx &ry);
glutCreateWindow("Ellipse");
glutDisplayFunc(midptellipse);
break;
}

minit();
glutMainLoop();
}
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

Program - 3

Write a program to fill any given polygon using scan-line area filling algorithm.

```
#include <gl/glut.h>
#include <GLUT/GLUT.h>
#include <studio.h>

int m;
typedef float point[3];
point tetra[4] = { {0,100,-100}, {0,0,100},
{100,-100,-100}, {-100,-100,-100} };
void tetrahedron (void);
void myinit (void);
void divide_triangle (point a, Point b, point c, int m);
void draw_triangle (point p1, Point p2, point p3);
int main (int argv, char ** argc)
{
    // int m;
    printf("Enter the number of iterations: ");
    scanf("%d", &m);
    glutInit(&argv, argc);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
    GLUT_DEPTH);
    glutInitWindowPosition (100,200);
    glutInitWindowSize (500,500);
    glutCreateWindow ("Seirpinski Gasket");
    glutDisplayFunc (tetrahedron);
```

Teacher's Signature _____

```
#include<gl/glut.h>
#include<GLUT/GLUT.h>
#include<stdio.h>

int m;
typedef float point[3];
point tetra[4] = { {0,100,-100},{0,0,100},{100,-100,-100},
{-100,-100,-100} };
void tetrahedron(void);
void myinit(void);
void divide_triangle(point a, point b, point c, int m);
void draw_triangle(point p1, point p2, point p3);
int main(int argv, char** argc)
{
    //int m;
    printf("Enter the number of iterations: ");
    scanf("%d", &m);
    glutInit(&argv, argc);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(100, 200);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Seirpinski Gasket");
    glutDisplayFunc(tetrahedron);
    glEnable(GL_DEPTH_TEST);
    myinit();
    glutMainLoop();
}
void divide_triangle(point a, point b, point c, int m)
{
    point v1, v2, v3;
    int j;
    if (m > 0) {
        for (j = 0; j < 3; j++)
            v1[j] = (a[j] + b[j]) / 2;
        for (j = 0; j < 3; j++)
            v2[j] = (a[j] + c[j]) / 2;
        for (j = 0; j < 3; j++)
            v3[j] = (b[j] + c[j]) / 2;

        divide_triangle(a, v1, v2, m - 1);
        divide_triangle(c, v2, v3, m - 1);
        divide_triangle(b, v3, v1, m - 1);
    }
    else
        draw_triangle(a, b, c);
}
void myinit()
{
    glClearColor(1, 1, 1, 1);

    glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
}
void tetrahedron(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
```

```
    divide_triangle(tetra[0], tetra[1], tetra[2], m);
    glColor3f(0.0, 1.0, 0.0);
    divide_triangle(tetra[3], tetra[2], tetra[1], m);
    glColor3f(0.0, 0.0, 1.0);
    divide_triangle(tetra[0], tetra[3], tetra[1], m);
    glColor3f(0.0, 0.0, 0.0);
    divide_triangle(tetra[0], tetra[2], tetra[3], m);
    glFlush();
}
void draw_triangle(point p1, point p2, point p3)
{
    glBegin(GL_TRIANGLES);
    glVertex3fv(p1);
    glVertex3fv(p2);
    glVertex3fv(p3);
    glEnd();
}
```

Expt. No. _____ Date _____ Page No. _____

```
glEnable (GL_DEPTH_TEST);
myinit ();
glutMainLoop ();
}
void divide_triangle (point a, point b, point c, int m)
{
  point v1, v2, v3;
  int j;
  if (m>0) {
    for (j=0; j<3; j++)
      v1[j] = (a[j] + b[j]) /2;
    for (j=0; j<3; j++)
      v2[j] = (a[j] + c[j]) /2;
    for (j=0; j<3; j++)
      v3[j] = (b[j] + c[j]) /2;
    divide_triangle (a, v1, v2, m-1);
    divide_triangle (c, v2, v3, m-1);
    divide_triangle (b, v3, v1, m-1);
  }
  else
    draw_triangle (a, b, c);
}
void myinit()
{
  glClearColor (1,1,1,1);
  glOrtho (-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
}
```

Teacher's Signature _____

Date

Page No.

Expt. No.

```
void tetrahedron(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER
          _BIT);
  glColor3f(1.0,0.0,0.0);
  divide_triangle(tetra[0], tetra[1], tetra[2], m);
  glColor3f(0.0, 1.0, 0.0);
  divide_triangle(tetra[3], tetra[2], tetra[1]
  ,m);
  glColor3f(0.0, 0.0, 1.0);
  divide_triangle(tetra[0], tetra[3], tetra[1], m);
  glColor3f(0.0, 0.0, 0.0);
  divide_triangle(tetra[0], tetra[2], tetra[3], m);
    glFlush();
}
void draw_triangle(point p1, point p2, point p3)
{
    glBegin(GL_TRIANGLES);
    glVertex3fv(p1);
    glVertex3fv(p2);
    glVertex3fv(p3);
    glEnd();
}
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

Program - 4

Write a program to recursively subdivide a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is to specified at execution time.

```
#include <GLUT/GLUT.h>
#include <stdlib.h>
#include <algorithm>
#include <iostream>


using namespace std;
float x[100], y[100];

int n, m;
int wx = 500, wy = 500;

void draw_line(float x1, float y1, float x2, float y2){

  glColor3f(1, 0, 0);
  glBegin(GL_LINES);
  glVertex2f(x1, y1);
  glVertex2f(x2, y2);
  glEnd();
  glFlush();

}
void edgeDetect(float x1, float y1, float x2, float y2, int
```

Teacher's Signature _____

```
#include <stdio.h>
#include <GLUT/GLUT.h>
#include "Header.h"


//RIGHT CLICK TO SHOW REFLECTED HOUSE


float house[11][2] = { { 100,200 },{ 200,250 },{ 300,200 },{ 100,200
},{ 100,100 },{ 175,100 },{ 175,150 },{ 225,150 },{ 225,100 },{
300,100 },{ 300,200 } };
int angle;
float m, c, theta;
void display()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //NORMAL HOUSE
        glColor3f(1, 0, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glFlush();
        //ROTATED HOUSE
        glPushMatrix();
        glTranslatef(100, 100, 0);
        glRotatef(angle, 0, 0, 1);
        glTranslatef(-100, -100, 0);
        glColor3f(1, 1, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glPopMatrix();
        glFlush();
}
void display2()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //normal house
        glColor3f(1, 0, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
```

```
        glEnd();
        glFlush();
        // line
        float x1 = 0, x2 = 500;
        float y1 = m * x1 + c;
        float y2 = m * x2 + c;
        glColor3f(1, 1, 0);
        glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glEnd();
        glFlush();

        //Reflected
        glPushMatrix();
        glTranslatef(0, c, 0);
        theta = atan(m);
        theta = theta * 180 / 3.14;
        glRotatef(theta, 0, 0, 1);
        glScalef(1, -1, 1);
        glRotatef(-theta, 0, 0, 1);
        glTranslatef(0, -c, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glPopMatrix();
        glFlush();
}
void myInit() {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 0.0, 0.0);
        glLineWidth(2.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
}
void mouse(int btn, int state, int x, int y) {
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
                display();
        }
        else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
                display2();
        }
}
void main(int argc, char** argv)
{
        printf("Enter the rotation angle\n");
        scanf("%d", &angle);
        printf("Enter c and m value for line y=mx+c\n");
        scanf("%f %f", &c, &m);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(900, 900);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("House Rotation");
        glutDisplayFunc(display);
```

```
glutMouseFunc(mouse);
myInit();
glutMainLoop();
```

Date

Expt. No. _____ Page No. _____

```
scanline){

float temp;
if (y2<y1){
    temp = x1; x1 = x2; x2 = temp;
}   temp = y1; y1 = y2; y2 = temp;

if (scanline > y1 && scanline < y2)
    int x[m++] = x1 + (scanline - y1) * (x2 - x1)
    / (y2 - y1);
}

void scanfill (float x[], float y[]){
    for(int s1=0; s1<=wy; s1++){
        m=0;
        for(int i=0, i<n; i++){

            edgeDetect (x[i], y[i], x[(i+1)%n]
        }, y[(i+1)%n], s1, 0);
        sort (int x, (int x+m));
        if (m>=2)
            for(int i=0; in<m; i=i+2)
                drawline(int x[i], s1, int x[i+1],
                s1);

        }
```

Teacher's Signature _____

Date

Expt. No.

Page No.

```
}
void display_filled_polygon () {
    glClear (GL_COLOR_BUFFER_BIT);
    glLineWidth (2);
    glBegin (GL_LINE_LOOP);
    for (int i = 0 ; i < n ; i++)
        glVertex2f (x[i], y[i]);
    glEnd ();
    scanfiV (x , y);
}

void myInit () {

    glClearColor (1.1,1.1,1.1);
    glColor3f (0.0,0.0,1.0);
    glPointSize (1);

    gluOrtho2D (0, wx , 0, wy);

}
int main (int ac, char * av[]) {
    glutInit (&ac, av);
    printf ("Enter no. of sides : \n");
    scanf ("%d", &n);
    printf ("Enter coordinates of end points : \n");
    for (int i = 0; i n ; i++)
```

Teacher's Signature

Expt. No. Date

Page No.

```
{
    printf("x-chord y-coord : \n");
    scanf("%f %f", &x[i], &y[i]);
}
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("scanline");
glutDisplayFunc(display_filled_polygon);
myInit();
glutMainLoop();
}
```

Expt. No. _____ Date _____

Page No. _____

Program - 5

Write a program to create a house like figure and rotate and reflect it about a given fixed point and an axis, defined by $y = mx + c$ using openGL/CUDA transformations functions.

```
#include <stdio.h>
#include <GLUT/GLUT.h>
#include "Header.h"

//RIGHT CLICK TO SHOW REFLECTED HOUSE

float house[11][2] = {{100,200},{200,250},
{300,200},{100,200},{100,100},{175,100},
{175,150},{225,150},{225,100},{300,100},
{300,200}};
int angle;
float m, c, theta;
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_
    DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho2D(-450,450,-450,450);
    glMatrixMode(GL_MODELVIEW);
```

Teacher's Signature _____

```
#include<GLUT/GLUT.h>
#include<stdio.h>
#include<math.h>
int xc, yc, r;
int rx, ry, xce, yce;
void draw_circle(int xc, int yc, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(xc + x, yc + y);
    glVertex2i(xc - x, yc + y);
    glVertex2i(xc + x, yc - y);
    glVertex2i(xc - x, yc - y);
    glVertex2i(xc + y, yc + x);
    glVertex2i(xc - y, yc + x);
    glVertex2i(xc + y, yc - x);
    glVertex2i(xc - y, yc - x);
    glEnd();
}
void circlebres()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int x = 0, y = r;
    int d = 3 - 2 * r;
    while (x <= y)
    {
        draw_circle(xc, yc, x, y);
        x++;
        if (d < 0)
            d = d + 4 * x + 6;
        else
        {
            y--;
            d = d + 4 * (x - y) + 10;
        }
        draw_circle(xc, yc, x, y);
    }
    glFlush();
}
int p1_x, p2_x, p1_y, p2_y;
int point1_done = 0;
void myMouseFunccircle(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
point1_done == 0)
    {
        p1_x = x - 250;
        p1_y = 250 - y;
        point1_done = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        p2_x = x - 250;
        p2_y = 250 - y;
        xc = p1_x;
        yc = p1_y;
        float exp = (p2_x - p1_x) * (p2_x - p1_x) + (p2_y - p1_y) *
```

```
(p2_y - p1_y);
        r = (int)(sqrt(exp));
        circlebres();
        point1_done = 0;
    }
}


void draw_ellipse(int xce, int yce, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x + xce, y + yce);
    glVertex2i(-x + xce, y + yce);
    glVertex2i(x + xce, -y + yce);
    glVertex2i(-x + xce, -y + yce);
    glEnd();
}
void midptellipse()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float dx, dy, d1, d2, x, y;
    x = 0;
    y = ry;


    d1 = (ry * ry) - (rx * rx * ry) +
        (0.25 * rx * rx);
    dx = 2 * ry * ry * x;
    dy = 2 * rx * rx * y;

    while (dx < dy)
    {

        draw_ellipse(xce, yce, x, y);



        if (d1 < 0)
        {
            x++;
            dx = dx + (2 * ry * ry);
            d1 = d1 + dx + (ry * ry);
        }
        else
        {
            x++;
            y--;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d1 = d1 + dx - dy + (ry * ry);
        }
    }


    d2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) +
        ((rx * rx) * ((y - 1) * (y - 1))) -
        (rx * rx * ry * ry);
```

```
    while (y >= 0)
    {

        draw_ellipse(xce, yce, x, y);

        if (d2 > 0)
        {
            y--;
            dy = dy - (2 * rx * rx);
            d2 = d2 + (rx * rx) - dy;
        }
        else
        {
            y--;
            x++;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d2 = d2 + dx - dy + (rx * rx);
        }
    }
    glFlush();
}
int p1e_x, p2e_x, p1e_y, p2e_y, p3e_x, p3e_y;
int point1e_done = 0;
void myMouseFunc(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
point1e_done == 0)
    {
        p1e_x = x - 250;
        p1e_y = 250 - y;
        xce = p1e_x;
        yce = p1e_y;
        point1e_done = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
point1e_done == 1)
    {
        p2e_x = x - 250;
        p2e_y = 250 - y;
        float exp = (p2e_x - p1e_x) * (p2e_x - p1e_x) + (p2e_y -
p1e_y) * (p2e_y - p1e_y);
        rx = (int)(sqrt(exp));

        point1e_done = 2;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
point1e_done == 2)
    {
        p3e_x = x - 250;
        p3e_y = 250 - y;
        float exp = (p3e_x - p1e_x) * (p3e_x - p1e_x) + (p3e_y -
p1e_y) * (p3e_y - p1e_y);
```

```
        ry = (int)(sqrt(exp));
        midptellipse();
        pointle_done = 0;
    }
}
void myDrawing()
{ }
void myDrawingc()
{ }

void minit()
{
    glClearColor(1, 1, 1, 1);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(3.0);
    gluOrtho2D(-250, 250, -250, 250);
}

int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);

    printf("Enter 1 to draw circle , 2 to draw ellipse\n");
    int ch;
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("Enter coordinates of centre of circle and radius\n");
    scanf("%d%d%d",&xc,&yc,&r);
    glutCreateWindow("Circle");
    glutDisplayFunc(circlebres);
    break;
    case 2:
    printf("Enter coordinates of centre of ellipse and major and minor
radius\n");
    scanf("%d%d%d%d",&xce,&yce,&rx,&ry);
    glutCreateWindow("Ellipse");
    glutDisplayFunc(midptellipse);
    break;
    }

    minit();
    glutMainLoop();
    }
```

Date

Page No.

```
glLoadIdentity ();
//NORMAL HOUSE
glColor3f (1, 0, 0);
glBegin (GL_LINE_LOOP);
for (int i=0; i<11; i++)
        glVertex2fv(house[i]);
glEnd();
glFlush ();
 // ROTATED HOUSE
glPushMatrix ();
glTranslatef(100, 100, 0);
glRotatef(angle, 0, 0, 1);
glTranslatef(-100, -100, 0);
glColor3f(1, 1, 0);
glBegin(GL_LINE_LOOP);
for (int i=0; i<11, i++)
            glVertex2fv(house[i]);
 glEnd();
  glPopMatrix ();
   glFlush();
}
void display2()
{
   glClearColor(1, 1, 1, 0);
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH
     BUFFER_BIT);
  glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D(-450, 450, -450, 450);
```

Teacher's Signature

Date

Expt. No.

Page No.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// normal house
glColor3f(1,0,0);
glBegin(GL_LINE_LOOP);
for(int i=0; i<11; i++)
    glVertex2fv(house[i]);
glEnd();
glFlush();
// line
float x1=0, x2=500;
float y1 = m*x1 + c;
float y2 = m*x2 + c;
glColor3f(1,1,0);
glBegin(GL_LINES);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glEnd();
glFlush();

// Reflected
glPushMatrix();
glTranslatef(0, c, 0);
theta = atan(m);
theta = theta * 180/3.14;
glRotatef(theta, 0, 0, 1);
glScalef(1, -1, 1);
glRotatef(-theta, 0, 0, 1);
glTranslatef(0, -c, 0);
```

Teacher's Signature

Expt. No. Date

Page No.

```
    glBegin(GL_LINE_LOOP);
      for(int i=0; i<11; i++)
            glVertex2fv(house[i]);
    glEnd();
    glPopMatrix();
    glFlush();
}
void myInit(){
      glClearColor(1.0, 1.0, 1.0, 1.0);
      glColor3f(1.0, 0.0, 0.0);
      glLineWidth(2.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-450, 450, -450, 450);
}
void mouse(int btn, int state, int x, int y){
    if(btn==GLUT_LEFT_BUTTON && state==GLUT
    _DOWN){
            display();
    }
    else if(btn==GLUT_RIGHT_BUTTON && state
    ==GLUT_DOWN){
            display2();
    }
}
void main(int argc, char** argv)
{
      printf("Enter the rotation angle\n");
      scanf("%d", &angle);
```

Teacher's Signature

Date

Page No.

```
printf("Enter c and m value for line y=mx+c\n");
scanf("%f %f", &c, &m);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(900, 900);
glutInitWindowPosition(100, 100);
glutCreateWindow("Mouse Rotation");
glutDisplayFunc(display);
glutMouseFunc(mouse);
myInit();
glutMainLoop();
}
```

Teacher's Signature

Expt. No. _____ Page No. _____

Program - 6

Write a program to implement the Cohen-Sutherland line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping and viewport for displaying the clipped image.

```
//
// lineClip.cpp
// CG Lab
//

#include <stdio.h>
#include <GLUT/GLUT.h>
#include "Header.h"

#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, ymax;
double xvmin, yvmin, xvmax, yvmax;

const int RIGHT = 4;
const int LEFT = 8;
const int TOP = 1;
const int BOTTOM = 2;

int n;
struct line_segment {
```

Teacher's Signature _____

```
#include <stdio.h>
#include<stdlib.h>
#include<GLUT/GLUT.h>
#include "Header.h"

#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, ymax;
double xvmin, yvmin, xvmax, yvmax;


const int RIGHT = 4;
const int LEFT = 8;
const int TOP = 1;
const int BOTTOM = 2;

int n;
struct line_segment {
    int x1;
    int y1;
    int x2;
    int y2;
};
struct line_segment ls[10];

outcode computeoutcode(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 cohensuther(double x0, double y0, double x1, double y1)
{
    outcode outcode0, outcode1, outcodeout;
    bool accept = false, done = false;

    outcode0 = computeoutcode(x0, y0);
    outcode1 = computeoutcode(x1, y1);

    do
    {
        if (!(outcode0 | outcode1))
        {
            accept = true;
            done = true;
        }
```

```
else if (outcode0 & outcode1)
    done = true;
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 = computeoutcode(x0, y0);
    }
    else
    {
        x1 = x;
        y1 = y;
        outcode1 = computeoutcode(x1, y1);
    }
}

} while (!done);

if (accept)
{
    double sx = (xvmax - xvmin) / (xmax - xmin);
    double sy = (yvmax - yvmin) / (ymax - ymin);
    double vx0 = xvmin + (x0 - xmin) * sx;
    double vy0 = yvmin + (y0 - ymin) * sy;
    double vx1 = xvmin + (x1 - xmin) * sx;
    double vy1 = yvmin + (y1 - ymin) * sy;

    glColor3f(1, 0, 0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xvmin, yvmin);
    glVertex2f(xvmax, yvmin);
    glVertex2f(xvmax, yvmax);
```

```
        glVertex2f(xvmin, yvmax);
        glEnd();

        glColor3f(0, 0, 1);
        glBegin(GL_LINES);
        glVertex2d(vx0, vy0);
        glVertex2d(vx1, vy1);
        glEnd();
    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for (int i = 0; i < n; i++)
    {
        glBegin(GL_LINES);
        glVertex2d(ls[i].x1, ls[i].y1);
        glVertex2d(ls[i].x2, ls[i].y2);
        glEnd();
    }

    for (int i = 0; i < n; i++)
        cohensuther(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);

    glFlush();
}
void myinit()
{
    glClearColor(1, 1, 1, 1);
    glColor3f(1, 0, 0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500);
}

void main_6(int argc, char** argv)
{
    printf("Enter window coordinates (xmin ymin xmax ymax): \n");
    scanf("%lf%lf%lf%lf", &xmin, &ymin, &xmax, &ymax);
    printf("Enter viewport coordinates (xvmin yvmin xvmax yvmax)
:\n");
    scanf("%lf%lf%lf%lf", &xvmin, &yvmin, &xvmax, &yvmax);
    printf("Enter no. of lines:\n");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        printf("Enter line endpoints (x1 y1 x2 y2):\n");
```

```
    scanf("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);
  }
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("clip");
  myinit();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

```
    int x1;
    int y1;
    int x2;
    int y2;
};
struct line_segment ls[10];

outcode computeoutcode (double x, double y)
{
    outcode code = 0;
    if (y > ymax)
        code |= TOP;
    else if (y < ymin)
        code |= RIGHT;
    else if (x < xmin)
        code |= LEFT;

    return code;
}

void cohensuther (double x0, double y0, double x1,
double y1)
{
    outcode outcode0, outcode1, outcodeout;
    bool accept = false, done = false;

    outcode0 = computeoutcode (x0, y0);
    outcode1 = computeoutcode (x1, y1);
```

Expt. No. _____ Date _____

Page No. _____

```
do
{
  if (!(outcode0 | outcode1))
  {
     accept = true;
     done = true;
  }
  else if (outcode0 & outcode1)
       done = true;
  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;
       }
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
else
{
  y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
  x = xmin;
}

if (outcodeout == outcode0)
{
   x0 = x;
   y0 = y;
   outcode0 = computecode(x0, y0);
}
else
{
   x1 = x;
   y1 = y;
   outcode1 = computecode(x1, y1);
}
}
} while (!done);

if (accept)
{
  double sx = (xvmax - xvmin) / (xmax - xmin);
  double sy = (yvmax - yvmin) / (ymax - ymin);
  double vx0 = xvmin + (x0 - xmin) * sx;
  double vy0 = yvmin + (y0 - ymin) * sy;
  double vx1 = xvmin + (x1 - xmin) * sx;
```

Teacher's Signature _____

Expt. No. Date

Page No.

```
double vy1 = yvmin + (y1 - ymin) * sy;

glColor3f (1, 0, 0);
glBegin (GL_LINE_LOOP);
glVertex2f (xvmin, yvmin);
glVertex2f (xvmax, yvmin);
glVertex2f (xvmax, yvmax);
glVertex2f (xvmin, yvmax);
glEnd();

glColor3f(0, 0, 1);
glBegin(GL_LINES);
glVertex2d(vx0, vy0);
glVertex2d(vx1, vy1);
glEnd();
}

void display()
{
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(0, 0, 1);
glBegin(GL_LINE_LOOP);
glVertex2f(xmin, ymin);
glVertex2f(xmax, ymin);
glVertex2f(xmax, ymax);
glVertex2f(xmin, ymax);
glEnd();
```

Teacher's Signature

Expt. No. Date

Page No.

```
for (int i = 0; i < n; i++)
{
    glBegin (GL_LINES);
    glVertex2d (ls[i].x1, ls[i].y1);
    glVertex2d (ls[i].x2, ls[i].y2);
    glEnd();
}

for (int i = 0; i < n ; i ++)
    cohensuther (ls[i].x1, ls[i].y1,
    ls[i].x2, ls[i].y2);

glFlush();
}
void myinit ()
{
    glClearColor (1, 1, 1, 1);
    glColor3f (1, 0, 0);
    glPointSize (1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho2D (0, 500, 0, 500);
}

void main_6 (int argc, char** argv)
{
    printf ("Enter window coordinates (xmin ymin
    xmax ymax) : \n");
    scanf ("%lf %lf %lf %lf", &xmin, &ymin, &xmax,
```

Teacher's Signature

```
&ymax);
printf("Enter viewport coordinates (xvmin, yvmin
xvmax  yvmax) : \n");
pscanf("%lf %lf %lf %lf", &xvmin, &yvmin,
&xvmax, &yvmax);
printf("Enter no. of lines : \n");
scanf("%d", &n);
for (int i = 0; i < n; i++)
{
    printf("Enter line endpoints(x1 y1 x2 y2)
    : \n");
    scanf("%d %d %d %d", &ls[i].x1, &ls
    [i].y1, &ls[i].x2, &ls[i].y2);
}
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("clip");
myinit();
glutDisplayFunc(display);
glutMainLoop();
}
```

Expt. No. _____ Date _____

Page No. _____

Program-7

Write a program be implement the Liang-Barsky line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping & viewport for displaying the clipped image.

```
#include<stdio.h>
#include<GL/GLUT/GLUT.h>
#include "Header.h"
double xmin, ymin, xmax, ymax; //50 50 100 100
double xvmin, yvmin, xvmax, yvmax; //200 200 300 30

int n;

struct line_segment {
    int x1;
    int y1;
    int x2;
    int y2;
};
struct line_segment ls[10];
int cliptest (double p, double q, double *u1,
double *u2)
{
    double r;
    if(p) r = q/p; // to check whether p
    if (p<0.0) // potentially entry point, update
    {
        if(r>*u1) *u1 = r;
```

Teacher's Signature _____

```
#include <stdio.h>
#include <GLUT/GLUT.h>
#include "Header.h"
double xmin, ymin, xmax, ymax; //50 50 100 100
double xvmin, yvmin, xvmax, yvmax; //200 200 300 300

int n;

struct line_segment {
    int x1;
    int y1;
    int x2;
    int y2;
};
struct line_segment ls[10];
int cliptest(double p, double q, double* u1, double* u2)
{
    double r;
    if (p) r = q / p;  // to check whether p
    if (p < 0.0)     // potentially entry point, update te
    {
        if (r > * u1)* u1 = r;
        if (r > * u2) return(false); // line portion is outside
    }
    else
        if (p > 0.0)     //  Potentially leaving point, update tl
        {
            if (r < *u2)* u2 = r;
            if (r < *u1) return(false); // line portion is outside
        }
        else
            if (p == 0.0)
            {
                if (q < 0.0) return(false); // line parallel to edge
but outside
            }
    return(true);
}

void LiangBarskyLineClipAndDraw(double x0, double y0, double x1,
double y1)
{
    double dx = x1 - x0, dy = y1 - y0, u1 = 0.0, u2 = 1.0;
    //draw a red colored viewport
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xvmin, yvmin);
    glVertex2f(xvmax, yvmin);
    glVertex2f(xvmax, yvmax);
    glVertex2f(xvmin, yvmax);
    glEnd();
    if (cliptest(-dx, x0 - xmin, &u1, &u2))  // inside test wrt left
edge
        if (cliptest(dx, xmax - x0, &u1, &u2)) // inside test wrt
right edge
            if (cliptest(-dy, y0 - ymin, &u1, &u2)) // inside test wrt
```

```
bottom edge
                if (cliptest(dy, ymax - y0, &u1, &u2)) // inside test
wrt top edge
                {
                    if (u2 < 1.0)
                    {
                        x1 = x0 + u2 * dx;
                        y1 = y0 + u2 * dy;
                    }
                    if (u1 > 0.0)
                    {
                        x0 = x0 + u1 * dx;
                        y0 = y0 + u1 * dy;
                    }
                    // Window to viewport mappings
                    double sx = (xvmax - xvmin) / (xmax - xmin); //
 Scale parameters
                    double sy = (yvmax - yvmin) / (ymax - ymin);
                    double vx0 = xvmin + (x0 - xmin) * sx;
                    double vy0 = yvmin + (y0 - ymin) * sy;
                    double vx1 = xvmin + (x1 - xmin) * sx;
                    double vy1 = yvmin + (y1 - ymin) * sy;

                    glColor3f(0.0, 0.0, 1.0); // draw blue colored
 clipped line
                    glBegin(GL_LINES);
                    glVertex2d(vx0, vy0);
                    glVertex2d(vx1, vy1);
                    glEnd();
                }
 }// end of line clipping

 void display()
 {
     glClear(GL_COLOR_BUFFER_BIT);
     //draw the line with red color
     glColor3f(1.0, 0.0, 0.0);
     for (int i = 0; i < n; i++)
     {
         glBegin(GL_LINES);
         glVertex2d(ls[i].x1, ls[i].y1);
         glVertex2d(ls[i].x2, ls[i].y2);
         glEnd();
     }
     //draw a blue colored window
     glColor3f(0.0, 0.0, 1.0);
     glBegin(GL_LINE_LOOP);
     glVertex2f(xmin, ymin);
     glVertex2f(xmax, ymin);
     glVertex2f(xmax, ymax);
     glVertex2f(xmin, ymax);
     glEnd();
     for (int i = 0; i < n; i++)
         LiangBarskyLineClipAndDraw(ls[i].x1, ls[i].y1, ls[i].x2,
   ls[i].y2);
     glFlush();
 }
```

```
void myinit()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(1.0, 0.0, 0.0);
    glLineWidth(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

void main_7(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);

    printf("Enter window coordinates: (xmin ymin xmax ymax) \n");
    scanf("%lf%lf%lf%lf", &xmin, &ymin, &xmax, &ymax);
    printf("Enter viewport coordinates: (xvmin yvmin xvmax yvmax)
\n");
    scanf("%lf%lf%lf%lf", &xvmin, &yvmin, &xvmax, &yvmax);
    printf("Enter no. of lines:\n");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        printf("Enter coordinates: (x1 y1 x2 y2)\n");
        scanf("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);
    }
    glutCreateWindow("Liang Barsky Line Clipping Algorithm");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

Expt. No. _____ Date _____

Page No. _____

```
        if (r > *u2) return(false); // line portion is outside
    }
    else
        if (p > 0.0)  // Potentially leaving point, update tl
        {
            if (r < *u2) *u2 = r;
        if (r < *u1) return (false); // line portion is outside
        }
        else
            if (p == 0.0)
            {
                if (q < 0.0) return(false); //line parallel to
                edge but outside
            }
        return (true);
    }

void LiangBarskyLineClipAndDraw (double x0, double y0,
double x1, double y1)
{
    double dx = x1 - x0, dy = y1 - y0, u1 = 1.0;
    // draw a red colored view port
    glColor3f (1.0, 0.0, 0.0);
    glBegin (GL_LINE_LOOP);
    glVertex2f (xvmin, yvmin);
    glVertex2f (xvmax, yvmin);
    glVertex2f (xvmax, yvmax);
    glVertex2f (xvmin, yvmax);
    glEnd ();
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
if (cliptest (-dx, x0 - xmin, &u1 &u2))
//inside test wrt left edge
    if (cliptest (dx, xmax -x0, &u1, &u2))
    //inside test wrt right edge
        if (cliptest (-dy, y0- ymin, &u1, &u2))
            if (cliptest(dy, ymax-y0, &u1, &u2))
              //inside test wrt top edge
            {
              if (u2<1.0)
              {
                 x1 = x0 + u2 * dx;
                 y1 = y0 + u2 * dy;
              }
              if (u1>0.0)
              {
                 x0 = x0 + u1 * dx;
                 y0 = y0 + u1 * dy;
              }
//Window to viewport mappings
double sx = (xvmax - xvmin) / (xmax -
xmin). // Scale parameters
double sy = (yvmax - yvmin) / (ymax-
ymin);
double vx0 = xvmin + (x0-xmin) * sx;
double vy0 = yvmin + (y0 - ymin) * sy;
double vx1 = xvmin + (x1 - xmin) * sx;
double vy1 = yvmin + (y1 - ymin) * sy;

glColor3f(0.0, 0.0, 1.0); //draw blue coloured
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

clipped line

```
            glBegin(GL_LINES);
            glVertex2d(vx0, vy0);
            glVertex2d(vx1, vy1);
            glEnd();
        }
} // end of line clipping

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    // draw the line with red color
    glColor3f(1.0, 0.0, 0.0);
    for(int i=0; i<n; i++)
    {
        glBegin(GL_LINES);
        glVertex2d(ls[i].x1, ls[i].y1);
        glVertex2d(ls[i].x2, ls[i].y2);
        glEnd();
    }
    // draw a blue coloured window
    glColor3f(0.0, 0.0, 1.0);
    glBegin(GL_LINE_LOOP);
      glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for(int i=0, i<n; i++)
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
        LiangBarskylineClipAndDraw(ls[i].x1, ls[i].y1,
        ls[i].x2, ls[i].y2);
    glFlush();
}

void myinit()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(1.0, 0.0, 0.0);
    glLineWidth(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

void main_7(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);

    Printf("Enter window coordinates: (xmin ymin xmax ymax)
    \n");
    scanf("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
    printf("Enter viewport coordinates: (xvmin yvmin xvmax
    yvmax) \n");
    Scanf("%lf %lf %lf %lf", &xvmin, &yvmin, &xvmax,
    &yvmax);
```

Teacher's Signature _____

Expt. No. Date

Page No.

```
printf("Enter no. of lines: \n");
scanf("%d",&n);

for(int i=0;i<n;i++)
{
    printf("Enter coordinates: (x1 y1 x2 y2)\n");
    scanf("%d%d%d%d",&ls[i].x1,&ls[i].y1,
    &ls[i].x2,&ls[i].y2);
}
glutCreateWindow("Liang Barsky line Clipping
Algorithm");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

Teacher's Signature

Expt. No. Date

Page No.

Program - 8

Write a program to implement the Cohen-Hodgeman polygon clipping algorithm. Make provision to specify the input polygon & window for clipping.

```
#include <iostream>
#include <GLUT/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size,
org_poly_points[20][2], clipper_size, clipper_
points [20][2];
const int MAX_POINTS = 20;


// Returns x-value of point of intersection of two
// lines

void drawPoly(int p[][2], int n) {
        glBegin(GL_POLYGON);
        for(int i=0; i<n; i++)
            glVertex2f(p[i][0], p[i][1]);
        glEnd();
}

int x_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2) * (x3-x4) -
```

Teacher's Signature

```
#include<GLUT/glut.h>
#include<math.h>
#include<stdio.h>
#include "Header.h"
#define CAR 1
#define WHEEL 2
float s = 1;
void carlist() {
    glNewList(CAR, GL_COMPILE);
    glColor3f(1, 1, 0);
    glBegin(GL_POLYGON);
    glVertex3f(0, 25, 0);
    glVertex3f(90, 25, 0);
    glVertex3f(90, 55, 0);
    glVertex3f(80, 55, 0);
    glVertex3f(20, 75, 0);
    glVertex3f(0, 55, 0);
    glEnd();
    glEndList();
}
void wheellist() {
    glNewList(WHEEL, GL_COMPILE_AND_EXECUTE);
    glColor3f(1, 0, 0);
    glutSolidSphere(10, 25, 25);
    glEndList();
}
void mykeyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 't': glutPostRedisplay();
        break;
    case 'q': exit(0);
    default: break;

    }
}

void myInit() {
    glClearColor(0, 0, 0, 0);
    glOrtho(0, 600, 0, 600, 0, 600);

}
void draw_wheel() {
    glColor3f(1, 0, 0);
    glutSolidSphere(10, 25, 25);

}

void moveCar(float s) {
    glTranslatef(s, 0.0, 0.0);
    glCallList(CAR);
    glPushMatrix();
    glTranslatef(25, 25, 0.0);      //move to first wheel position
    draw_wheel();
    glCallList(WHEEL);
    glPopMatrix();
    glPushMatrix();
```

```
    glTranslatef(75, 25, 0.0);      //move to 2nd wheel position
    draw_wheel();
    glCallList(WHEEL);
    glPopMatrix();
    glFlush();
}
void myDisp() {
    glClear(GL_COLOR_BUFFER_BIT);
    carlist();
    moveCar(s);
    wheellist();

}
void mouse(int btn, int state, int x, int y) {
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        s += 7;
        myDisp();
    }
    else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        s += 1;
        myDisp();
    }
}


int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("car");
    myInit();
    glutDisplayFunc(myDisp);
    glutMouseFunc(mouse);
    glutKeyboardFunc(mykeyboard);
    glutMainLoop();
}
```

Expt. No. _____ Date _____

Page No. _____

```
        (x1-x2)*(x3*y4-y3*x4);
    int den = (x1-x2)*(y3-y4)-(y1-y2)*(x3-x4);
    return num/den;
}

// Returns y-value of point of intersection of
// two lines
int y_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2-y1*x2)*(y3-y4)-
              (y1-y2)*(x3*y4-y3*x4);
    int den = (x1-x2)*(y3-y4)-(y1-y2)*
              (x3-x4);
    return num/den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
void clip(int poly_points[][2], int &poly_size,
          int x1, int y1, int x2, int y2)
{
    int new_points[MAX_POINTS][2], new_poly_size=0;

    //(ix,iy),(kx,ky) are the coordinate values of
    // the points.
    for (int i=0; i<poly_size; i++)
    {
        // i & k form a line in polygon
```

Teacher's Signature _____

Date

Page No.

```
int k = (i+1) % poly_size;
int ix = poly_points [i][0], iy = poly_points
         [i][1];
int kx = poly_points [k][0], ky = poly_points
         [k][i];

// calculating position of first point
// w.r.t. clipper line
int i_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix - x1);

// Calculating position of second point
// w.r.t clipper line
int k_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx - x1);

// Case 1 : when both points are inside
if (i_pos >= 0 && k_pos >= 0)
{
    // Only second point is added
    new_points [new_poly_size][0] = kx;
    new_points [new_poly_size][1] = ky;
    new_poly_size ++;
}

// Case 2 : when only first point is outside
else if (i_pos < 0 && k_pos >= 0)
{
    // Point of intersection with edge
    // and the second point is added
    new_points [new_poly_size][0] = x_intersect(x1,
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

```
                y1, x2, y2, ix, iy, kx, ky);
    new_points[new_poly_size][1] = y_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
    new_poly_size++;

    new_points[new_poly_size][0] = kx;
    new_points[new_poly_size][1] = ky;
    new_poly_size++;
}

// Case 3 : when only second point is outside
else if (i_pos >= 0 && k_pos < 0)
{
    // Only point of intersection with edge is added
    new_points[new_poly_size][0] = x_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
    new_points[new_poly_size][1] = y_intersect(
                x1, y1, x2, y2, ix, iy, kx, ky);
    new_poly_size++;
}

// Case 4 : when both points are outside
else
{
    // No points are added
}
}

// Copying new points into original array
```

Teacher's Signature _____

Expt. No. Date

Page No.

```
// and changing the no. of vertices
poly_size = new_poly_size;
for(int i = 0, i < poly_size; i++)
{
    poly_points[i][0] = new_points[i][0];
    poly_points[i][1] = new_points[i][1];
}

}

void init () {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);
    glClear(GL_COLOR_BUFFER_BIT);
}

// Implements Sutherland Hodgman algorithm
void display()
{
    init();
    glColor3f(1.0f, 0.0f, 0.0f);
    drawPoly(clipper_points, clipper_size);

    glColor3f(0.0f, 1.0f, 0.0f);
    drawPoly(org_poly_points, org_poly_size);
// i and k are two consecutive indexes.
```

Teacher's Signature

Expt. No. Date

Page No.

```
for(int i=0; i<clipper_size; i++)
{
    int k=(i+1) % clipper_size;

    // We pass the current array of vertices,
    // its size and the end points of the selected
    // clipper line.

    clip(poly_points, poly_size, clipper_points
        [i][0],
        clipper_points[i][1], clipper_points
        [k][0],
        clipper_points[k][1]);
}

glColor3f(0.0f, 0.0f, 1.0f);
drawPoly(poly_points, poly_size);
glFlush();
}

// Driver Code
int main(int argc, char **argv[])
{
    printf("Enter no. of vertices: \n");
    scanf("%d", &poly_size);
    for(poly_size = poly_size;
    for(int i=0; i<poly_size; i++)
    {
        printf("Polygon Vertex: \n");
```

Teacher's Signature

Date

Expt. No.

Page No.

```
scanf("%d", &clipper_size);
for(int i = 0; i < clipper_size; i++)
{
        printf("Clip Vertex :\n");
        scanf("%d %d", &clipper_points
        [i][0], &clipper_points[i][1]);
}

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(100, 100);
glutInitCreateWindow("Polygon Clipping!!")
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Teacher's Signature

```
// algorithm for polygon clipping
#include<iostream>
#include<GLUT/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, org_poly_points[20]
[2], clipper_size, clipper_points[20][2];
const int MAX_POINTS = 20;


// Returns x-value of point of intersection of two
// lines

void drawPoly(int p[][2], int n) {
        glBegin(GL_POLYGON);
        for (int i = 0; i < n; i++)
                glVertex2f(p[i][0], p[i][1]);
        glEnd();
}

int x_intersect(int x1, int y1, int x2, int y2,
        int x3, int y3, int x4, int y4)
{
        int num = (x1 * y2 - y1 * x2) * (x3 - x4) -
                (x1 - x2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        return num / den;
}

// Returns y-value of point of intersectipn of
// two lines
int y_intersect(int x1, int y1, int x2, int y2,
        int x3, int y3, int x4, int y4)
{
        int num = (x1 * y2 - y1 * x2) * (y3 - y4) -
                (y1 - y2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        return num / den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
void clip(int poly_points[][2], int& poly_size,
        int x1, int y1, int x2, int y2)
{
        int new_points[MAX_POINTS][2], new_poly_size = 0;

        // (ix,iy),(kx,ky) are the co-ordinate values of
        // the points
        for (int i = 0; i < poly_size; i++)
        {
                // i and k form a line in polygon
                int k = (i + 1) % poly_size;
                int ix = poly_points[i][0], iy = poly_points[i][1];
                int kx = poly_points[k][0], ky = poly_points[k][1];

                // Calculating position of first point
```

```
            // w.r.t. clipper line
            int i_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix -
x1);

            // Calculating position of second point
            // w.r.t. clipper line
            int k_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx -
x1);

            // Case 1 : When both points are inside
            if (i_pos >= 0 && k_pos >= 0)
            {
                    //Only second point is added
                    new_points[new_poly_size][0] = kx;
                    new_points[new_poly_size][1] = ky;
                    new_poly_size++;
            }

            // Case 2: When only first point is outside
            else if (i_pos < 0 && k_pos >= 0)
            {
                    // Point of intersection with edge
                    // and the second point is added
                    new_points[new_poly_size][0] = x_intersect(x1,
                            y1, x2, y2, ix, iy, kx, ky);
                    new_points[new_poly_size][1] = y_intersect(x1,
                            y1, x2, y2, ix, iy, kx, ky);
                    new_poly_size++;

                    new_points[new_poly_size][0] = kx;
                    new_points[new_poly_size][1] = ky;
                    new_poly_size++;
            }

            // Case 3: When only second point is outside
            else if (i_pos >= 0 && k_pos < 0)
            {
                    //Only point of intersection with edge is
    added
                    new_points[new_poly_size][0] = x_intersect(x1,
                            y1, x2, y2, ix, iy, kx, ky);
                    new_points[new_poly_size][1] = y_intersect(x1,
                            y1, x2, y2, ix, iy, kx, ky);
                    new_poly_size++;
            }

            // Case 4: When both points are outside
            else
            {
                    //No points are added
            }
    }

    // Copying new points into original array
    // and changing the no. of vertices
    poly_size = new_poly_size;
    for (int i = 0; i < poly_size; i++)
```

```
        {
                poly_points[i][0] = new_points[i][0];
                poly_points[i][1] = new_points[i][1];
        }
}


void init() {
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);
        glClear(GL_COLOR_BUFFER_BIT);
}

// Implements Sutherland Hodgman algorithm
void display()
{
        init();
        glColor3f(1.0f, 0.0f, 0.0f);
        drawPoly(clipper_points, clipper_size);

        glColor3f(0.0f, 1.0f, 0.0f);
        drawPoly(org_poly_points, org_poly_size);
        //i and k are two consecutive indexes

        for (int i = 0; i < clipper_size; i++)
        {
                int k = (i + 1) % clipper_size;

                // We pass the current array of vertices, it's size
                // and the end points of the selected clipper line
                clip(poly_points, poly_size, clipper_points[i][0],
                        clipper_points[i][1], clipper_points[k][0],
                        clipper_points[k][1]);
        }

        glColor3f(0.0f, 0.0f, 1.0f);
        drawPoly(poly_points, poly_size);
        glFlush();
}

//Driver code
int main(int argc, char* argv[])
{
        printf("Enter no. of vertices: \n");
        scanf("%d", &poly_size);
        org_poly_size = poly_size;
        for (int i = 0; i < poly_size; i++)
        {
                printf("Polygon Vertex:\n");
                scanf("%d%d", &poly_points[i][0], &poly_points[i][1]);
                org_poly_points[i][0] = poly_points[i][0];
                org_poly_points[i][1] = poly_points[i][1];
        }
```

```
        printf("Enter no. of vertices of clipping window:");
        scanf("%d", &clipper_size);
        for (int i = 0; i < clipper_size; i++)
        {
                printf("Clip Vertex:\n");
                scanf("%d%d", &clipper_points[i][0],
&clipper_points[i][1]);
        }

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(400, 400);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Polygon Clipping!");
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Expt. No. _____ Date _____

Page No. _____

Program - 9

Write a program to model a car like figure using display lists.

```
#include <GLUT/glut.h>
#include <math.h>
#include <stdio.h>
#include "Header.h"
#define CAR 1
#define WHEEL 2
float s=1;
void carlist(){
        glNewList(CAR, GL_COMPILE);
        glColor3f(1,1,0);
        glBegin(GL_POLYGON);
        glVertex3f(0,25,0);
        glVertex3f(90,25,0);
        glVertex3f(90,55,0);
        glVertex3f(80,55,0);
        glVertex3f(20,75,0);
        glVertex3f(0,55,0);
        glEnd();
        glEndList();
}
void wheellist(){
        glNewList(WHEEL, GL_COMPILE_AND_EXECUTE
        glColor3f(1,0,0);
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
    glutSolidSphere(10, 25, 25);
    glEndList();
}
void myKeyboard(unsigned char key, int x, int y) {
    switch(key) {
        case 't': glutPostRedisplay();
                  break;
        case 'q': exit(0);
        default : break;
    }
}

void myInit() {
    glClearColor(0, 0, 0, 0);
    glOrtho(0, 600, 0, 600, 0, 600);
}
void drawwheel() {
    glColor3f(1, 0, 0);
    glutSolidSphere(10, 25, 25);
}

void moveCar(float s) {
    glTranslatef(s, 0.0, 0.0);
    glCallList(CAR);
    glPushMatrix();
    glTranslatef(25, 25, 0.0); // move to first wheel
// position
```

Teacher's Signature _____

Expt. No. Date

Page No.

```
    draw_wheel();
    glCallList(WHEEL);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(25, 25, 0.0); // move to second
                               // position
    draw_wheel();
    glCallList(WHEEL);
    glPopMatrix();
    glFlush();
}
void myDisp() {
    glClear(GL_COLOR_BUFFER_BIT);
    carlist();
    movecar(s);
    Wheellist();

}
void mouse(int btn, int state, int x, int y) {
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_
       DOWN) {
        s+=7;
        myDisp();
    }
    else if(btn == GLUT_RIGHT_BUTTON && state ==
            GLUT_DOWN) {
        s+=1;
        myDisp();
    }
```

Teacher's Signature

Expt. No. Date

Page No.

```
}

int main (int argc, char *argv[]) {
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (600, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("car");
    myInit ();
    glutDisplayFunc (myDisp);
    glutMouseFunc (mouse);
    glutKeyboardFunc (myKeyboard);
    glutMainLoop ();
}
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

Program - 10

Write a program to create a color cube and spin it using Open GL transformations.

```
#include <stdlib.h>
#include <GLUT/GLUT.h>
#include <OpenGL/GL.h>
#include <OpenGL/GLU.h>
#include <time.h>
#include "Header.h"

GLfloat vertices[] = {-1.0,-1.0,-1.0, 1.0,-1.0,-1.0,
1.0, 1.0,-1.0,-1.0,1.0,-1.0,-1.0,-1.0,1.0,-1.0,1.0,1.0,
1.0,-1.0, 1.0,1.0,1.0,1.0,-1.0,1.0,1.0};

GLfloat normals[] = {-1.0,-1.0,-1.0, 1.0,-1.0,-1.0,
1.0,1.0,-1.0, -1.0,1.0,-1.0,-1.0,-1.0,1.0,
1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0};

GLfloat colors[] = {0.0,0.0,0.0,1.0,0.0,0.0,
1.0,1.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0,
1.0,0.0,1.0, 1.0,1.0,1.0, 0.0,1.0,1.0};

GLubyte cubeIndices[] = {0,3,2,1,2,3,7,6,0,4,7,3,1,2,
6,5,4,5,6,7,0,1,5,4};

static GLfloat theta[] = {0.0,0.0,0.0};
```

Teacher's Signature _____

```
#include <stdlib.h>
#include <GLUT/GLUT.h>
#include<OpenGL/GL.h>
#include<OpenGL/GLU.h>
#include <time.h>
#include "Header.h"

GLfloat vertices[] = { -1.0,-1.0,-1.0,1.0,-1.0,-1.0,
1.0,1.0,-1.0, -1.0,1.0,-1.0, -1.0,-1.0,1.0,
1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0 };

GLfloat normals[] = { -1.0,-1.0,-1.0,1.0,-1.0,-1.0,
1.0,1.0,-1.0, -1.0,1.0,-1.0, -1.0,-1.0,1.0,
1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0 };

GLfloat colors[] = { 0.0,0.0,0.0,1.0,0.0,0.0,
1.0,1.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0,
1.0,0.0,1.0, 1.0,1.0,1.0, 0.0,1.0,1.0 };

GLubyte cubeIndices[] = {
0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4 };

static GLfloat theta[] = { 0.0,0.0,0.0 };
static GLfloat beta[] = { 0.0,0.0,0.0 };
static GLint axis = 2;

void delay(float secs)
{
    float end = clock() / CLOCKS_PER_SEC + secs;
    while ((clock() / CLOCKS_PER_SEC) < end);
}

void displaySingle(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);

    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);

    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.0, 1.0, 1.0);
    glEnd();

    glFlush();
}

void spinCube()
{
    delay(0.01);
    theta[axis] += 2.0;
    if (theta[axis] > 360.0) theta[axis] -= 360.0;
```

```
    glutPostRedisplay();
}


void mouse10(int btn, int state, int x, int y)
{
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
    if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
    if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w,
            2.0 * (GLfloat)h / (GLfloat)w, -10.0, 10.0);
    else
        glOrtho(-2.0 * (GLfloat)w / (GLfloat)h,
            2.0 * (GLfloat)w / (GLfloat)h, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}

void main_10(int argc, char** argv)
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("colorcube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(displaySingle);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse10);
    glEnable(GL_DEPTH_TEST);
    glEnableClientState(GL_COLOR_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(3, GL_FLOAT, 0, colors);
    glNormalPointer(GL_FLOAT, 0, normals);
    glColor3f(1.0, 1.0, 1.0);
    glutMainLoop();
}
```

```
static GLfloat beta[] = {0.0, 0.0, 0.0};
static GLint axis = 2;

void delay (float secs)
{
  float end = clock()/CLOCKS_PER_SEC + secs;
  while ((clock()) / CLOCKS_PER_SEC < end);
}

void display_single (void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER
          _BIT);
  glLoadIdentity();
  glRotatef (theta[0], 1.0, 0.0, 0.0);
  glRotatef (theta[1], 0.0, 1.0, 0.0);
  glRotatef (theta[2], 0.0, 0.0, 1.0);
  {
  glDrawElements (GL_QUADS, 24, GL_UNSIGNED_BYTE,
                  cubeIndices);
  glBegin (GL_LINES);
  glVertex3f (0.0, 0.0, 0.0);
  glVertex3f (1.0, 1.0, 1.0);
  glEnd();

  glFlush();
}
```

Date

Page No.

Teacher's Signature

Expt. No. Date

Page No.

```
void spinCube()
{
    delay(0.01);
    theta[axis] += 2.0;
    if(theta[axis] > 360.0) theta[axis] -= 360.0;
    glutPostRedisplay();
}

void mouse1O(int btn, int state, int x, int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_
    ~~btn ==~~ DOWN) axis = 0;
    if(btn == GLUT_MIDDLE_BUTTON && state == GLUT_
                DOWN) axis = 1;
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_
                DOWN) axis = 2;
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w <= h)
        glOrtho(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w,
            2.0 * (GLfloat)h / (GLfloat)w, -10.0, 10.0);
    else
        glOrtho(-200 * (GLfloat)w / (GLfloat)h,
```

Teacher's Signature

Expt. No. Date

Page No.

```
        2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode (GL_MODELVIEW);
}

void main_10(int argc, char ** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 100);
    glutInitWindowSize (500, 500);
    glutCreateWindow ("colorcube");
    glutReshapeFunc (myReshape);
    glutDisplayFunc (displaysingle);
    glutIdleFunc (spincube);
    glutMouseFunc (mouse10);
    glEnable (GL_DEPTH_TEST);
    glEnableClientState (GL_COLOR_ARRAY);
    glEnableClientState (GL_NORMAL_ARRAY);
    glEnableClientState (GL_VERTEX_ARRAY);
    glVertexPointer (3, GL_FLOAT, 0, vertices);
    glColorPointer (3, GL_FLOAT, 0, colors);
    glNormalPointer (GL_FLOAT, 0, normals);
    glColor3f (1.0, 1.0, 1.0);
    glutMainLoop ();
}
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

Program - 11

Write a program to generate a limacon, Cardiod, Three-leaf Spiral.

```
#include <GLUT/GLUT.h>
#include <math.h>
#include <stdio.h>
#include "Header.h"

struct screenPt {
        int x;
        int y;
};
typedef enum { limacon = 1, cardioid = 2, threeleaf = 3,
               spiral = 4 } curveName;
int w = 600, h = 500;
int curve = 1;
int red = 0, green = 0, blue = 0;
void myinitFi (void) {
        glClearColor (1.0, 1.0, 1.0, 1.0);
        glMatrixMode (GL_PROJECTION);
        gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}
void lineSegment (screenPt p1, screenPt p2) {

        glBegin (GL_LINES);
        glVertex2i (p1.x, p1.y);
```

Teacher's Signature _____

```
#include<GLUT/GLUT.h>
#include<math.h>
#include<stdio.h>
#include "Header.h"

struct screenPt {
    int x;
    int y;
};
typedef enum { limacon = 1, cardioid = 2, threeLeaf = 3, spiral = 4 }
curveName;
int w = 600, h = 500;
int curve = 1;
int red = 0, green = 0, blue = 0;
void myinit11(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
void lineSegment(screenPt p1, screenPt p2) {

    glBegin(GL_LINES);
    glVertex2i(p1.x, p1.y);
    glVertex2i(p2.x, p2.y);
    glEnd();
    glFlush();
}
void drawCurve(int curveNum) {
    const double twoPi = 6.283185;
    const int a = 175, b = 60;
    float r, theta, dtheta = 1.0 / float(a);
    int x0 = 200, y0 = 250;
    screenPt curvePt[2];
    curve = curveNum;
    glColor3f(red, green, blue);
    curvePt[0].x = x0;
    curvePt[0].y = y0;
    glClear(GL_COLOR_BUFFER_BIT);
    switch (curveNum) {
    case limacon: curvePt[0].x += a + b; break;
    case cardioid: curvePt[0].x += a + a; break;
    case threeLeaf: curvePt[0].x += a; break;
    case spiral: break;
    default: break;
    }
    theta = dtheta;
    while (theta < twoPi) {
        switch (curveNum) {
        case limacon: r = a * cos(theta) + b; break;
        case cardioid: r = a * (1 + cos(theta)); break;
        case threeLeaf: r = a * cos(3 * theta); break;
        case spiral: r = (a / 4.0) * theta; break;
        default: break;
        }
        curvePt[1].x = x0 + r * cos(theta);
        curvePt[1].y = y0 + r * sin(theta);
```

```
            lineSegment(curvePt[0], curvePt[1]);

            curvePt[0].x = curvePt[1].x;
            curvePt[0].y = curvePt[1].y;
            theta += dtheta;
        }
    }

    void colorMenu(int id) {
        switch (id) {

        case 0:
            break;
        case 1:
            red = 0;
            green = 0;
            blue = 1;

            break;
        case 2:
            red = 0;
            green = 1;
            blue = 0;
            break;

        case 4:
            red = 1;
            green = 0;
            blue = 0;

            break;
        case 3:
            red = 0;
            green = 1;
            blue = 1;

            break;
        case 5:
            red = 1;
            green = 0;
            blue = 1;
            break;
        case 6:
            red = 1;
            green = 1;
            blue = 0;
            break;
        case 7:
            red = 1;
            green = 1;
            blue = 1;
            break;
        default:
            break;

        }
        drawCurve(curve);
```

```
}
void main_menu(int id) {

    switch (id) {

    case 3: exit(0);
    default: break;
    }
}
void mydisplay() {
    /*int curveNum=1;
    glClear(GL_COLOR_BUFFER_BIT);
    /*printf("Enter the integer value corresponding to one of the
followinf curve names:\n");
    printf("1 - limacon\n2 - cardioid\n3 - threeleaf\n4 - spiral\n");
    scanf_s("%d", &curveNum);*/

    /*if (curveNum == 1 || curveNum == 2 || curveNum == 3 || curveNum
== 4)
    drawCurve(curveNum);*/

}

void myreshape(int nw, int nh) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (double)nw, 0.0, (double)nh);
    glClear(GL_COLOR_BUFFER_BIT);
}

void main_11(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(w, h);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Drawing curves");
    int curveId = glutCreateMenu(drawCurve);
    glutAddMenuEntry("Limacon", 1);
    glutAddMenuEntry("Cardioid", 2);
    glutAddMenuEntry("Threeleaf", 3);
    glutAddMenuEntry("Spiral", 4);
    glutAttachMenu(GLUT_LEFT_BUTTON);
    int colorId = glutCreateMenu(colorMenu);
    glutAddMenuEntry("Red", 4);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("Blue", 1);
    glutAddMenuEntry("Black", 0);
    glutAddMenuEntry("Yellow", 6);
    glutAddMenuEntry("Cyan", 3);
    glutAddMenuEntry("Magenta", 5);
    glutAddMenuEntry("white", 7);
    glutAttachMenu(GLUT_LEFT_BUTTON);
    glutCreateMenu(main_menu);
    glutAddSubMenu("drawCurve", curveId);
    glutAddSubMenu("colors", colorId);
    glutAddMenuEntry("quit", 3);
    glutAttachMenu(GLUT_LEFT_BUTTON);
```

```
    myinit11();
    glutDisplayFunc(mydisplay);
    glutReshapeFunc(myreshape);

    glutMainLoop();
}
```

Expt. No. Date

Page No.

```
    glEnd();
    glFlush();
}
void drawCurve(int curveNum) {
    const double twoPi = 6.283185;
    const int a = 175, b = 60;
    float r, theta, dtheta = 1.0/float(a);
    int x0 = 200, y0 = 250;
    ScreenPt curvePt[2];
    curve = curveNum;
    glColor3f(red, green, blue);
    curvePt[0].x = x0;
    curvePt[0].y = y0;
    glClear(GL_COLOR_BUFFER_BIT);
    switch(curveNum) {
    case limacon: curvePt[0].x += a+b; break;
    case cardioid: curvePt[0].x += a+a; break;
    case threeleaf: curvePt[0].x += a; break;
    case spiral: break;
    default: break;
    }
    theta = dtheta;
    while (theta < twoPi) {
        switch(curveNum) {
        case limacon: r = a * cos(theta) + b; break;
        case cardioid: r = a * (1 + cos(theta)); break;
        case threeleaf: r = a * cos(3 * theta); break;
        case spiral: r = (a/4.0) * theta; break;
```

Teacher's Signature

Date

Expt. No. Page No.

```
        default : break;
    }
    curvePt[1].x = x0 + r * cos(theta);
    curvePt[1].y = y0 + r * sin(theta);
    lineSegment(curvePt[0], curvePt[1]);

    curvePt[0].x = curvePt[1].x;
    curvePt[0].y = curvePt[1].y;
    theta += dtheta;
  }
}

void colorMenu(int id){
    switch(id){

case 0:
    break;
case 1:
    red = 0
    green = 0
    blue = 1;

    break;
Case 2:
    red = 0
    green = 1;
    blue = 0;
    break;
```

Teacher's Signature

Expt. No. _____ Date _____

Page No. _____

```
Case 4:
    red = 1;
    green = 0;
    blue = 0;

    break;
Case 3:
    red = 0;
    green = 1;
    blue = 1;

    break;
Case 5:
    red = 1;
    green = 0;
    blue = 1;
    break;
Case 6:
    red = 1;
    green = 1;
    blue = 0;
Case 7:
    red = 1;
    green = 1;
    blue = 1;
    break;
default:
    break;
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
    }
    drawCurve(curve);
}
void main_menu(int id) {
    switch(id) {
    case 3: exit(0);
    default: break;
    }
}
void mydisplay() {
    /* int curveNum=1;
    glClear(GL_COLOR_BUFFER_BIT);
    /* printf("Enter the integer value corresponding
        to one of the following curve names
        : \n");
    printf("1-limacon\n2-cardioid\n3-
        threeleaf\n4-Spiral\n");
    scanf_s("%d", &curveNum); */
    /* if (curveNum==1 || curveNum==2 ||
        curveNum==3 || curveNum==4)
    drawCurve(curveNum); */
}
void myreshape(int nw, int nh) {
```

Teacher's Signature _____

Date

Expt. No. _____ Page No. _____

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, (double)nw, 0.0, (double)nh);
glClear(GL_COLOR_BUFFER_BIT);
}

void main_11(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(w, h);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Drawing curves");
    int curveId = glutCreateMenu(drawCurve);
    glutAddMenuEntry("Limacon", 1);
    glutAddMenuEntry("Cardioid", 2);
    glutAddMenuEntry("Threeleaf", 3);
    glutAddMenuEntry("Spiral", 4);
    glutAttachMenu(GLUT_LEFT_BUTTON);
    int colorId = glutCreateMenu(colorMenu);
    glutAddMenuEntry("Red", 4);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("Blue", 1);
    glutAddMenuEntry("Black", 0);
    glutAddMenuEntry("Yellow", 6);
    glutAddMenuEntry("Cyan", 3);
    glutAddMenuEntry("Magenta", 5);
    glutAddMenuEntry("White", 7);
    glutAttachMenu(GLUT_LEFT_BUTTON);
    glutCreateMenu(main_menu);
```

Teacher's Signature _____

Expt. No. _____ Date _____

Page No. _____

```
glutAddSubMenu ("draw Curve", curveId);
glutAddMenuEntry("Colors", colorId);
glutAttachMenu (GLUT_LEFT_BUTTON);
myinit();
glutDisplayFunc(mydisplay);
glutReshapeFunc(myreshape);

glutMainLoop();
}
```

Teacher's Signature _____