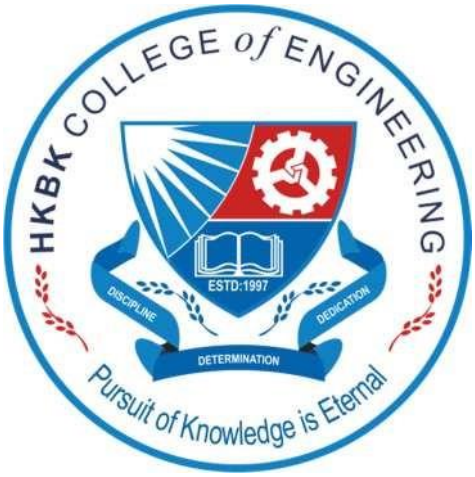
HKBK COLLEGE OF ENGINEERING
(Affiliated to VTU, Belgaum and Approved by AICTE)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



LABORATORY MANUAL

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

17CSL68

[As per choice based credit system (CBCS) scheme]

(Effective from the academic year 2018-2019)

PREPARED BY

Prof. MARY NATHISIYA **Prof. PUSHPA** **Prof. SEEMA SHIVPUR**

HKBK COLLEGE OF ENGINEERING
Nagawara, Bangaluru -560 045
www.hkbkeducation.org

HKBK COLLEGE OF ENGINEERING
(Affiliated to VTU, Belgaum and Approved by AICTE)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Vision and Mission of the Institution

Vision

To empower students through wholesome education and enable the students to develop into highly qualified and trained professionals with ethics and emerge as responsible citizen with broad outlook to build a vibrant nation.

Mission

- To achieve academic excellence in science, engineering and technology through dedication to duty, innovation in teaching and faith in human values.
- To enable our students to develop into outstanding professional with high ethical standards to face the challenges of 21st century.
- To provide educational opportunities to the deprived and weaker section of the society to uplift their socio-economic status.

Vision and Mission of the CSE Department

Vision

To advance the intellectual capacity of the nation and the international community by imparting knowledge to graduates who are globally recognized as innovators, entrepreneur and competent professionals.

Mission

- To provide excellent technical knowledge and computing skills to make the graduates globally competitive with professional ethics.
- To involve in research activities and be committed to lifelong learning to make positive contributions to the society.

HKBK COLLEGE OF ENGINEERING
(Affiliated to VTU, Belgaum and Approved by AICTE)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Programme Educational Objectives

PEO-1	To provide students with a strong foundation in engineering fundamentals and in the computer science and engineering to work in the global scenario.
PEO-2	To provide sound knowledge of programming, computing techniques and good communication and interpersonal skills so that they will be capable of analyzing, designing and building innovative software systems.
PEO-3	To equip students in the chosen field of engineering and related fields to enable him to work in multidisciplinary teams.
PEO-4	To inculcate in students professional, personal and ethical attitude to relate engineering issues to broader social context and become responsible citizen.
PEO-5	To provide students with an environment for life-long learning which allow them to successfully adapt to the evolving technologies throughout their professional career and face the global challenges.

Programme Outcomes	
a.	**Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
b.	**Problem Analysis: Identify,** formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
c.	**Design/ Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
d.	**Conduct investigations of complex problems** using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
e.	**Modern Tool Usage**: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an under- standing of the limitations.
f.	**The Engineer and Society**: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.
g.	**Environment and Sustainability**: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.

h.	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
i.	**Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams and in multi disciplinary settings.
j.	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
k.	**Life-long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life- long learning in the broadest context of technological change.
l.	**Project Management and Finance**: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
Programme Specific Outcomes	
m.	**Problem-Solving Skills:** An ability to investigate and solve a problem by analysis, interpretation of data, design and implementation through appropriate techniques,tools and skills.
n.	**Professional Skills**: An ability to apply algorithmic principles, computing skills and computer science theory in the modeling and design of computer-based systems.
o.	**Entrepreneurial Ability:** An ability to apply design, development principles and management skills in the construction of software product of varying complexity to become an entrepreneur

COURSE OUTCOMES: The students should be able to:

1. Apply the concepts of computer graphics
2. Implement computer graphics applications using OpenGL
3. Animate real world problems using OpenGL

Mapping of Course outcome to Programme Outcomes

PO / CO	a	b	c	d	E	f	g	H	i	j	k	L	m	n	o
1	3	3	3	3	2	-	-	-	2	-	-	-	1	1	1
2	3	3	3	3	2	-	-	-	2	-	-	-	1	1	1
3	3	3	3	3	2	-	-	-	2	-	-	3	3	3	3

3	- High correlation
2	- Moderate(medium) correlation
1	- Slight(low) correlation
-	- No correlation

SYLLABUS

Hours/Week: 03 Exam Hours: 03
I.A. Marks: 20 Total Hours: 40
Semester: VI/A and B C Exam Marks:60

PART - A

Design, develop, and implement the following programs in C / C++

Sl No.	Experiment	Page No.
1.	Implement Brenham's line drawing algorithm for all types of slope. **Refer:Text-1: Chapter 3.5, Refer:Text-2: Chapter 8**	1
2.	Create and rotate a triangle about the origin and a fixed point. **Refer:Text-1: Chapter 5-4**	6
3.	Draw a colour cube and spin it using OpenGL transformation matrices. **Refer:Text-2: Modelling a Coloured Cube**	11
4.	Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing. **Refer:Text-2: Topic: Positioning of Camera**	16
5.	Clip a lines using Cohen-Sutherland algorithm **Refer:Text-1: Chapter 6.7, Refer:Text-2: Chapter 8**	22
6.	To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the Surfaces of the solid object used in the scene. **Refer:Text-2: Topic: Lighting and Shading**	29
7.	Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user. **Refer: Text-2: Topic:** sierpinski gasket.	35
8.	Develop a menu driven program to animate a flag using Bezier Curve algorithm **Refer: Text-1: Chapter** 8-10	41
9.	Develop a menu driven program to fill the polygon using scan line algorithm	44
PART – B		
Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project.		

Note: (During the practical exam: the students should demonstrate and answer Viva-Voce)
Sample Topics: Simulation of concepts of OS, Data structures, algorithms etc.

OBJECTIVES:

- Demonstrate simple algorithms using OpenGL Graphics Primitives and attributes.
- Implementation of line drawing and clipping algorithms using OpenGL functions
- Design and implementation of algorithms Geometric transformations on both 2D and 3D objects.

OUTCOMES:

At the end of this lab session, the students will

- Apply the concepts of computer graphics
- Implement computer graphics applications using OpenGL
- Animate real world problems using OpenGL

Conduction of Practical Examination:

1. All laboratory experiments from part A are to be included for practical examination.
2. Mini project has to be evaluated for 30 Marks as per 6(b).
3. Report should be prepared in a standard format prescribed for project work.
4. Students are allowed to pick one experiment from the lot.
5. Strictly follow the instructions as printed on the cover page of answer script.
6. Marks distribution:
    a) Part A: Procedure + Conduction + Viva:10 + 35 +5 =50 Marks
    b) Part B: Demonstration + Report + Viva voce = 15+10+05 = 30 Marks
7. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

Reference books:

1. Donald Hearn & Pauline Baker: Computer Graphics-OpenGL Version,3rd Edition, Pearson Education,2011

2. Edward Angel: Interactive computer graphics- A Top Down approach with OpenGL, 5th edition. Pearson Education, 2011

3. M M Raikar, Computer Graphics using OpenGL, Fillip Learning / Elsevier, Bangalore / New Delhi (2013)

Q No. 1: Implement Brenham's line drawing algorithm for all types of slope.

DESCRIPTION

To display screen where the straight line segments are to be drawn. The vertical axes show scan-line position and the horizontal axes identify pixel columns

OBJECTIVE

- To understand Brenham's line drawing algorithm

OUTCOME

- Students will be able to draw a line for all types of slope.

ALGORITHM

Bresenham's Line-Drawing Algorithm for $|m| < 1.0$

1. Input the two line endpoints and store the left endpoint in $(x0, y0)$.
2. Set the color for frame-buffer position $(x0, y0)$; i.e., plot the first point.
3. Calculate the constants $\_x$, $\_y$, 2_y, and $2\_y - 2\_x$, and obtain the starting value for the decision parameter as

$$p0 = 2\_y - \_x$$

4. At each xk along the line, starting at $k = 0$, perform the following test: If $pk < 0$, the next point to plot is $(xk + 1, yk)$ and

$$pk+1 = pk + 2\_y$$

Otherwise, the next point to plot is $(xk + 1, yk + 1)$ and

$$pk+1 = pk + 2\_y - 2\_x$$

5. Repeat step 4 $\_x - 1$ more times.

PROGRAM 1.

```
#include<GL/glut.h>
#include<GL/gl.h>
#include<stdio.h>
int x1,Y1,x2,Y2;
void myInit()
{
glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);

glPointSize(3.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0, 200,0,200);
}
void draw_pixel(int x,int y)
{
glBegin(GL_POINTS);
glColor3f(1.0,0.0,0.0);
glVertex2i(x,y);
glEnd();
}
void draw_line(int x1, int x2 ,int Y1,int Y2)
{
int dx,dy,i,e;
int incx,incy,inc1,inc2;
int x,y;

dx=x2-x1;
dy=Y2-Y1;

if(dx<0)dx=-dx;
if(dy<0)dy=-dy;
incx=1;
if(x2<x1)incx=-1;
incy=1;
if(Y2<Y1)incy=-1;
x=x1; y=Y1;
if(dx>dy)
{
draw_pixel(x,y);
e=2*dy-dx;
inc1=2*(dy-dx);
inc2=2*dy;
for(i=0;i<dx;i++)
{
```

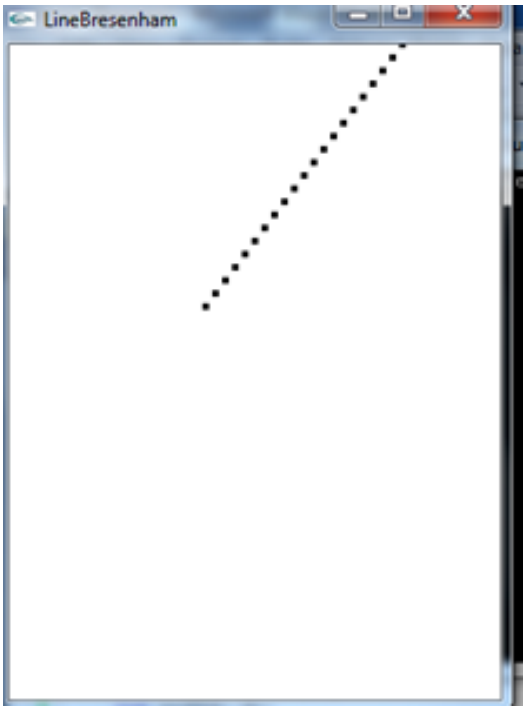
```
if(e>=0)
{
y+=incy;
e+=inc1;
}
else
e+=inc2;
x+=incx;
draw_pixel(x,y);
}
}
else
{
draw_pixel(x,y);
e=2*dx-dy;
inc1=2*(dx-dy);
inc2=2*dx;
for(i=0;i<dy;i++)
{
if(e>=0)
{
x+=incx;
e+=inc1;
}
else
e+=inc2;
y+=incy;
draw_pixel(x,y);
}
}
}
void myDisplay()
{
draw_line(x1,x2,Y1,Y2);
glFlush();
}
void main(int argc,char ** argv)
{
printf("enter(x1,y1,x2,y2)\n");
scanf("%d %d %d %d",&x1,&Y1,&x2,&Y2);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("'.....ABC.....Bresenham's line drawing");
```

```
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
}
```

SAMPLE INPUT AND OUTPUT

gcc -o 1 1.c -lGL -lGLU -lglut

./1



Test Case: Try for both positive and negative slopes

Q No. 2: Create and rotate a triangle about the origin and a fixed point.

DESCRIPTION

A triangle is transformed from its original position to the desired transformation using the composite- matrix calculations in procedure transformVerts2D.

OBJECTIVE

- To understand rotation of a triangle about the origin and a fixed point
- To understand the method of drawing triangle object using simple GL primitives.

OUTCOME

- Students will be able to rotate triangle.

PROGRAM 2.

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

/* Set initial display-window size. */
GLsizei winWidth = 600, winHeight
=
600; /* Set range for world coordinates.
*/ GLfloat xwcMin = 0.0, xwcMax =
225.0;
GLfloat ywcMin = 0.0, ywcMax = 225.0;
class wcPt2D {
public:
GLfloat
x, y; };
typedef GLfloat Matrix3x3
[3][3]; Matrix3x3
matComposite;
const GLdouble pi =
3.14159; void init (void)
{
/* Set color of display window to white. */
glClearColor (1.0, 1.0, 1.0, 0.0);
}
/* Construct the 3 x 3 identity matrix. */
```

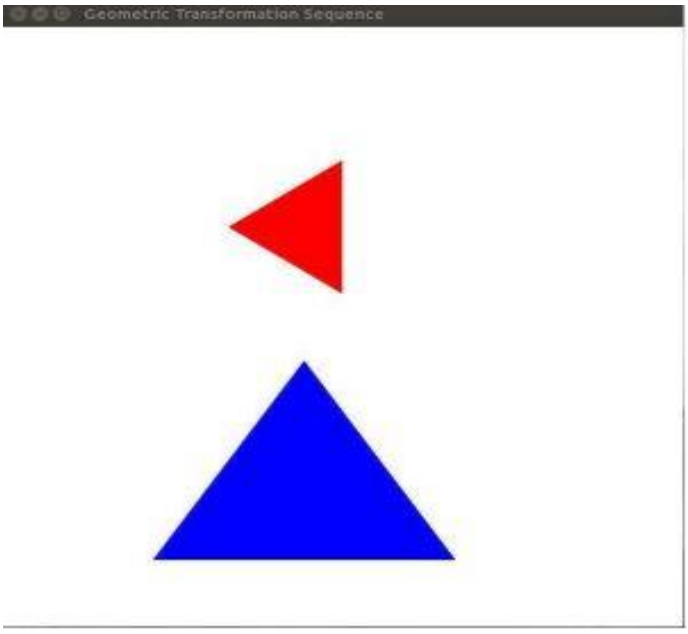
```
void matrix3x3SetIdentity (Matrix3x3 matIdent3x3)
{
GLint row, col;
for (row = 0; row < 3; row++)
for (col = 0; col < 3; col++)

matIdent3x3 [row][col] = (row == col);
}

/* Premultiply matrix m1 times matrix m2, store result in m2.
*/ void matrix3x3PreMultiply (Matrix3x3 m1, Matrix3x3 m2)
{
GLint row, col;
Matrix3x3 matTemp;
for (row = 0; row < 3; row++)
for (col = 0; col < 3 ; col++)
matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] *
m2 [1][col] + m1 [row][2] * m2 [2][col];
for (row = 0; row < 3; row++)
for (col = 0; col < 3; col++)
m2 [row][col] = matTemp [row][col];
}



void translate2D (GLfloat tx, GLfloat ty)
{
Matrix3x3 matTransl;
/* Initialize translation matrix to identity.
*/ matrix3x3SetIdentity (matTransl);
matTransl [0][2] = tx;
matTransl [1][2] = ty;
/* Concatenate matTransl with the composite matrix.
*/ matrix3x3PreMultiply (matTransl, matComposite);
}

void rotate2D (wcPt2D pivotPt, GLfloat theta)
{
Matrix3x3 matRot;
/* Initialize rotation matrix to identity. */
matrix3x3SetIdentity (matRot);
matRot [0][0] = cos (theta);
matRot [0][1] = -sin (theta);
matRot [0][2] = pivotPt.x * (1 - cos (theta)) +
pivotPt.y * sin (theta);
matRot [1][0] = sin (theta);
```

```
matRot [1][1] = cos (theta);
matRot [1][2] = pivotPt.y * (1 - cos (theta)) - pivotPt.x * sin
(theta); /* Concatenate matRot with the composite matrix. */
matrix3x3PreMultiply (matRot, matComposite);
}

void scale2D (GLfloat sx, GLfloat sy, wcPt2D fixedPt)
{


Matrix3x3 matScale;
/* Initialize scaling matrix to identity. */
matrix3x3SetIdentity(matScale);
matScale [0][0] = sx;
matScale [0][2] = (1 - sx) * fixedPt.x;
matScale [1][1] = sy;
matScale [1][2] = (1 - sy) * fixedPt.y;
/* Concatenate matScale with the composite matrix. */
matrix3x3PreMultiply(matScale, matComposite);
}
/* Using the composite matrix, calculate transformed coordinates. */
void transformVerts2D(GLint nVerts, wcPt2D *verts)
{
GLint k;
GLfloat temp;
for (k = 0; k < nVerts; k++) {
temp = matComposite [0][0] * verts [k].x + matComposite [0][1] *
verts [k].y + matComposite [0][2];



verts [k].y = matComposite [1][0] * verts [k].x + matComposite [1][1] *
verts [k].y + matComposite [1][2];
verts [k].x = temp;
}
}

void triangle(wcPt2D *verts)
{
GLint k;
glBegin (GL_TRIANGLES);
for (k = 0; k < 3; k++)
glVertex2f (verts [k].x, verts [k].y);
glEnd ( );
}

void displayFcn (void)
```

```
{
/* Define initial position for triangle. */
GLint nVerts = 3;
wcPt2D verts [3] = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} };
/* Calculate position of triangle centroid. */
wcPt2D centroidPt;
GLint k, xSum = 0, ySum = 0;
for (k = 0; k < nVerts; k++) {
xSum += verts [k].x;
ySum += verts [k].y;
}
centroidPt.x = GLfloat (xSum) / GLfloat (nVerts);
centroidPt.y = GLfloat (ySum) / GLfloat (nVerts);
/* Set geometric transformation parameters. */
wcPt2D pivPt,fixedPt;
pivPt = centroidPt;
fixedPt = centroidPt;
GLfloat tx = 0.0, ty = 100.0;
GLfloat sx = 0.5, sy = 0.5;
GLdouble theta = pi/2.0;
glClear (GL_COLOR_BUFFER_BIT); // Clear display window.
glColor3f (0.0, 0.0, 1.0); // Set initial fill color to blue.
triangle(verts); // Display blue triangle.
/* Initialize composite matrix to identity. */
matrix3x3SetIdentity(matComposite);
/* Construct composite matrix for transformation sequence. */
scale2D (sx, sy, fixedPt); // First transformation: Scale.
rotate2D (pivPt, theta); // Second transformation: Rotate
translate2D (tx, ty); // Final transformation: Translate.
/* Apply composite matrix to triangle vertices.
*/ transformVerts2D(nVerts, verts);
glColor3f (1.0, 0.0, 0.0); // Set color for transformed



triangle. triangle(verts); // Display red transformed.
glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
gluOrtho2D (xwcMin, xwcMax, ywcMin,
ywcMax); glClear (GL_COLOR_BUFFER_BIT);
}
```

```
int main (int argc, char ** argv)
{
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE |
GLUT_RGB); glutInitWindowPosition (50, 50);
glutInitWindowSize (winWidth, winHeight);
glutCreateWindow ("Geometric Transformation Sequence");
init ( );
glutDisplayFunc (displayFcn);
glutReshapeFunc (winReshapeFcn);
glutMainLoop ( );
}
```

SAMPLE INPUT AND OUTPUT



Test Case: Try for different rotation angle

Q No. 3: Draw a colour cube and spin it using OpenGL transformation matrices.

DESCRIPTION

The program draws a cube using GL_POLYGON primitive. For each vertex of the cube different color is given so that the cube becomes colorful. Whenever mouse left, right, or middle buttons are pressed the cube will spin along x axis, y axis or z axis respectively.

OBJECTIVE

- To understand rotation transformation
- To understand the method of drawing 3D objects using simple GL primitives.

OUTCOME

- Students will be able to animate 3D colored objects.

ALGORITHM

1. Initialize vertices, normal's and colors of the color cube. Also initialize theta and axis.
2. Define main()
    a. Enable display mode as double and depth buffer.
    b. Register mouse and idle callback function.
    c. Enable depth test.
3. Define reshape () enabling glOrtho.
4. Define mouse callback mouse(), coding for left, middle and right buttons for x, y and z axis respectively.
5. Define idle callback spincube()
    a. Increment theta for every click.
    b. Reinitialize theta
        If theta > 360 then theta → 0
    c. Call display()
6. Define callback function display()
    a. Load identity matrix.
    b. Define rotation function for x, y & z axis.
    c. Draw the color cube using right hand rule.
    d. Swap buffers for there is animation in the scene.

PROGRAM 3.

/* rotating cube with color interpolation

Demonstration of use of homogeneous coordinate transformations and simple data structure for representing cube from Chapter 4

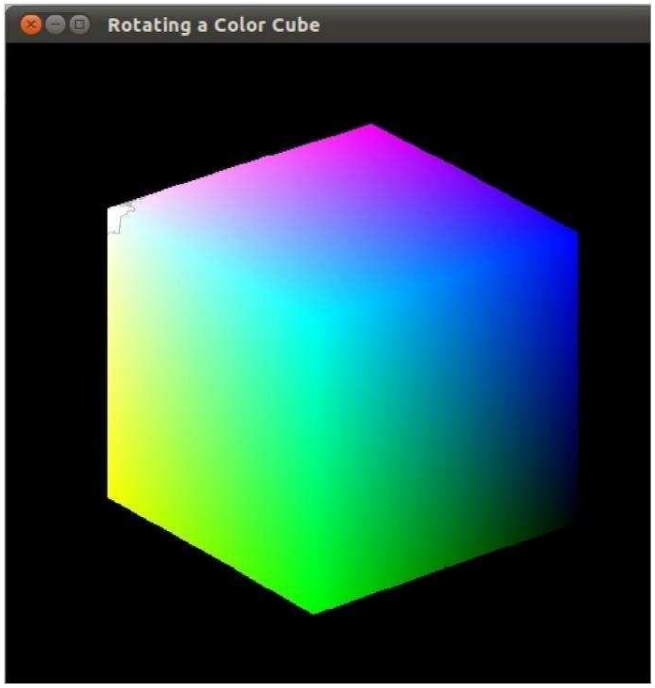
Both normals and colors are assigned to the vertices. Cube is centered at origin so (unnormalized) normals are the same as the vertex values */

```
#include <stdlib.h>
#include <GL/glut.h>

GLfloat vertices[][3] = {{-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[][3] = {{-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[][3] = {{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}};

void polygon(int a, int b, int c , int d)
{
/* draw a polygon via list of vertices */
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glNormal3fv(normals[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glNormal3fv(normals[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glNormal3fv(normals[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glNormal3fv(normals[d]);
                glVertex3fv(vertices[d]);
        glEnd();
}
void colorcube(void)
{
/* map vertices to faces */
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
```

```
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

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

void display(void)
{
/* display callback, clear frame buffer and z buffer, rotate cube and draw, swap buffers */

        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);

        colorcube();
        glFlush();
        glutSwapBuffers();
}

void spinCube()
{
/* Idle callback, spin cube 1 degrees about selected axis */

        theta[axis] += 1.0;
        if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
        glutPostRedisplay();
}

void mouse(int btn, int state, int x, int y)
{
/* mouse callback, selects an axis about which to rotate */
        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;
}
```

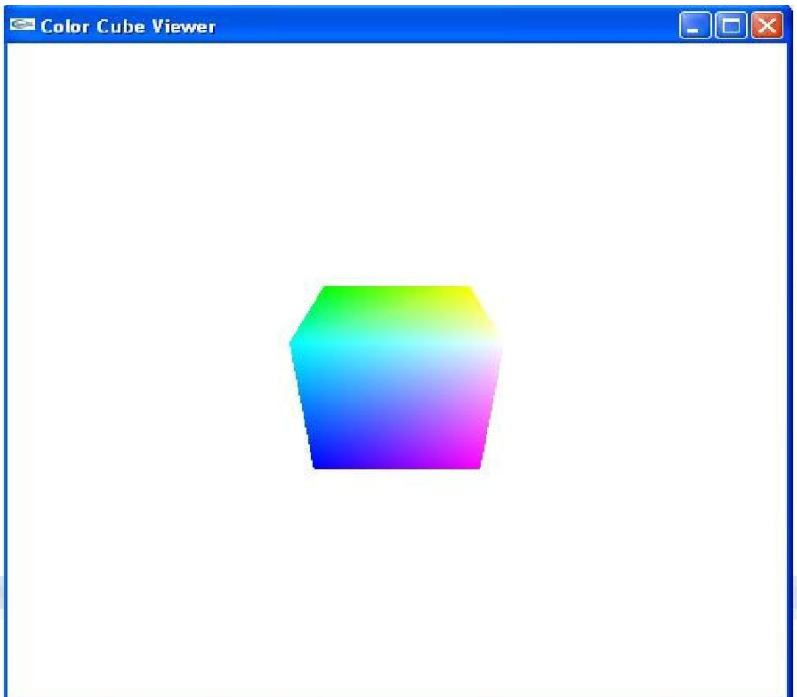
```
void myReshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w, 2.0 * (GLfloat) h / (GLfloat) w, -10.0, 10.0);
   else
      glOrtho(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 * (GLfloat)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

Void main(int argc, char **argv)
{
   glutInit(&argc, argv);

/* need both double buffering and z buffer */
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(500, 500);
   glutCreateWindow("Rotating a Color Cube");
   glutReshapeFunc(myReshape);
   glutDisplayFunc(display);
         glutIdleFunc(spinCube);
         glutMouseFunc(mouse);
         glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */
   glutMainLoop();
}
```

SAMPLE INPUT AND OUTPUT
gcc -o 3 3.c -lGL -lGLU -lglut
./3

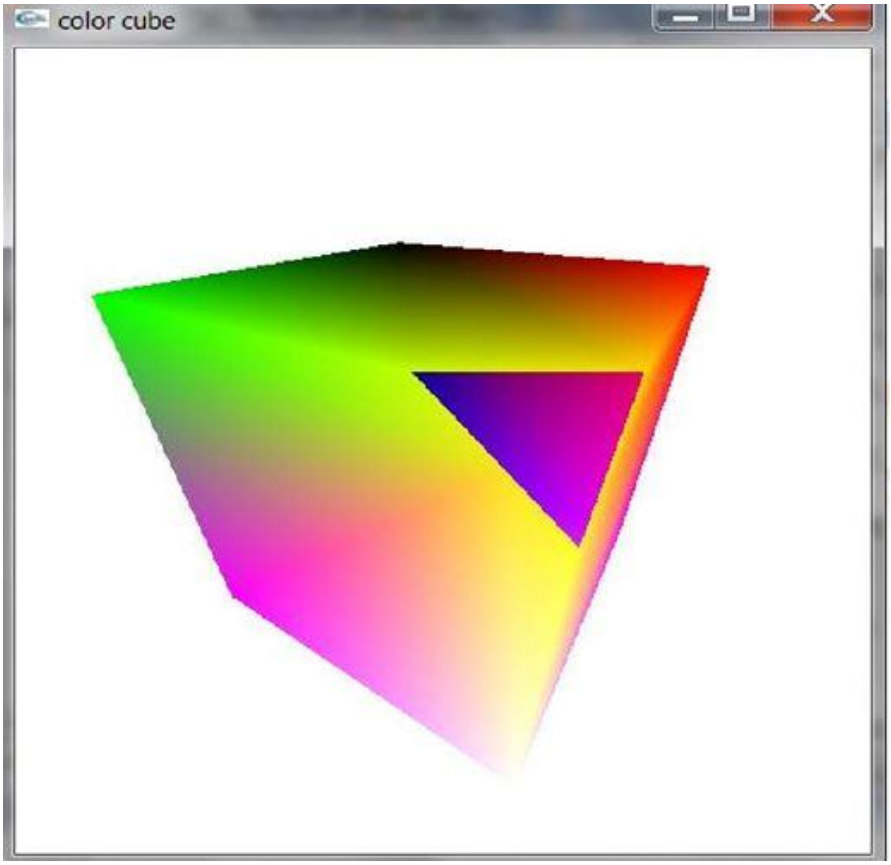


Results:

Run 1:



RUN 2



Q No. 4: Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

DESCRIPTION

The program creates a cube using GL_POLYGON primitive. For each vertex of the cube different color is given so that the cube looks colorful. x, X, y, Y, z and Z keys are used to move camera for perspective viewing. Whenever mouse left, right, or middle buttons are pressed the cube will rotate along x axis, y axis or z axis respectively.

OBJECTIVE

- Understand perspective viewing model with respect to camera movements.

OUTCOME

- Students will be able to implement perspective viewing model with respect to camera movements.

ALGORITHM

1. Initialize vertices, normals and colors of the color cube. Also initialize theta, axis and viewer position.
2. Define main()
    a. Enable display mode as double and depth buffer.
    b. Register mouse and keyboard callback function.
    c. Enable depth test.
3. Define reshape(). Enable either glFrustum or gluPerspective for perspective viewing.
4. Define mouse callback mouse()
    a. Program for left, middle and right buttons for x, y and z axis respectively.
    b. Increment theta for every click.
    c. Reinitialize theta
        If theta > 360 then theta → 0
    d. Call display()
5. Define keyboard callback keys()

    Program for moving the viewer towards or away from the object by incrementing or decrementing by 1, for x, y & z keys for respective directions accordingly.
6. Define callback function display()
    a. Load identity matrix and gluLookAt for viewer. b. Define rotation function for x, y & z axis.
    c. Draw the colorcube using righthand rule.
    d. Swap buffers for there is animation in the scene.

PROGRAM 4.

```
/* Rotating cube with viewer movement */

/* Use Lookat function in the display callback to point the viewer, whose position can be altered by
the x,X,y,Y,z, and Z keys. The perspective view is set in the reshape callback */

#include <stdlib.h>
#include <GL/glut.h>

GLfloat vertices[][3] = {{-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[][3] = {{-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[][3] = {{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}};

void polygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glNormal3fv(normals[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glNormal3fv(normals[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glNormal3fv(normals[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glNormal3fv(normals[d]);
                glVertex3fv(vertices[d]);
        glEnd();
}

void colorcube()
{
```

```
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

static GLfloat theta[] =
{0.0,0.0,0.0}; static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */

void display(void)
{

 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

/* Update viewer position in modelview matrix */

glLoadIdentity();
gluLookAt(viewer[0],viewer[1],viewer[2],  0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

/* rotate cube */

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); colorcube();
glFlush();
glutSwapBuffers();
}

void mouse(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;

        theta[axis] += 2.0;
        if( theta[axis] > 360.0 ) theta[axis] -=
        360.0; display();
}
```

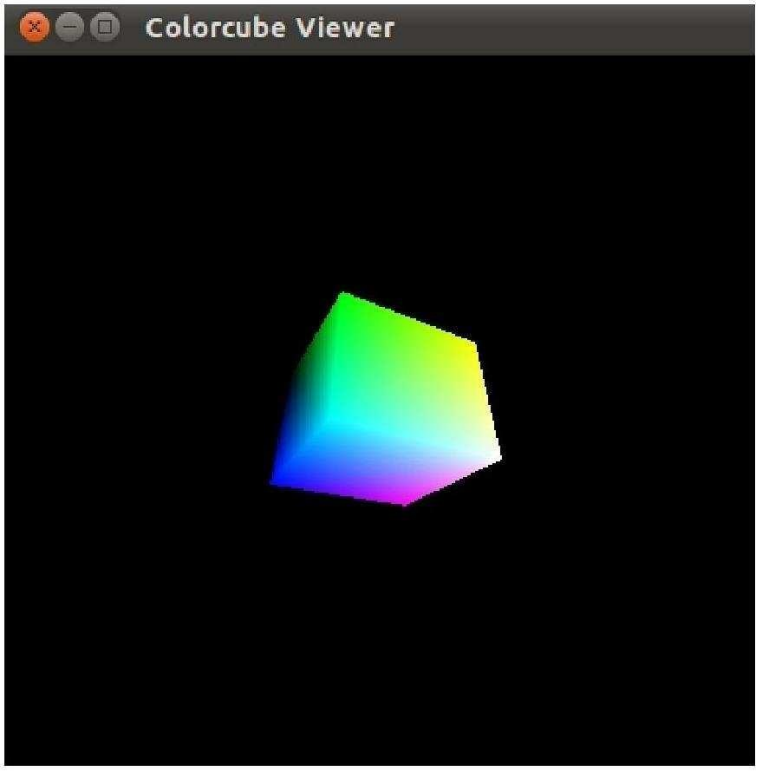
```
void keys(unsigned char key, int x, int y)
{

        /* Use x, X, y, Y, z, and Z keys to move viewer */
  if(key == 'x') viewer[0]-= 1.0;
  if(key == 'X') viewer[0]+=
  1.0; if(key == 'y') viewer[1]-=
  1.0; if(key == 'Y')
  viewer[1]+= 1.0; if(key == 'z')
  viewer[2]-= 1.0; if(key == 'Z')
  viewer[2]+= 1.0; display();
}

void myReshape(int w, int h)
{
 glViewport(0, 0, w, h);

/* Use a perspective view */

 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();

if(w<=h)
        glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,  2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
else
        glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,  2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);

/* Or we can use gluPerspective */
 /* gluPerspective(45.0, w/h, -10.0, 10.0); */

 glMatrixMode(GL_MODELVIEW);
}

void  main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
 GLUT_DEPTH); glutInitWindowSize(500, 500);
```

```
glutCreateWindow("Colorcube Viewer");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keys);
        glEnable(GL_DEPTH_TEST);
glutMainLoop();}
```

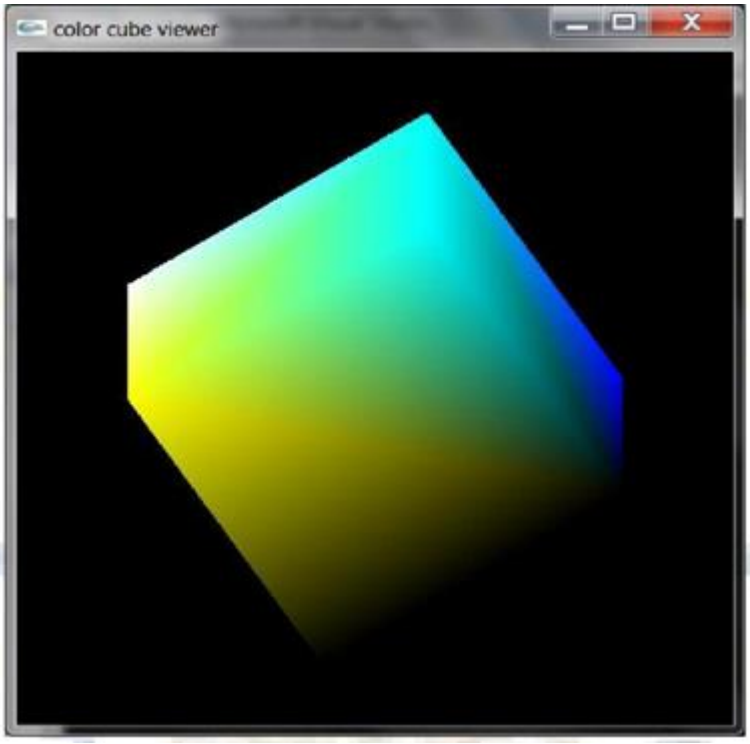
SAMPLE INPUT AND OUTPUT

cc -o 4 4.c -lGL -lGLU -lglut

RUN 1



RUN 2



Q No. 5: Clip a lines using Cohen-Sutherland algorithm

DESCRIPTION

The program draws a clipping window with the line which has to be clipped and calls Cohen- Sutherland line-clipping algorithm. The algorithm divides a two-dimensional space into 9 regions, and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport). The clipped line and clipping window will be scaled and displayed as output.

OBJECTIVE

- To understand the concept of clipping
- Learning Cohen-Sutherland line-clipping algorithm

OUTCOME

- Students will be able to implement Cohen Sutherland line clipping algorithm and compare the same with Liang barsky algorithm.

ALGORITHM

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.

   If both codes are **0000**,(bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.

   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1's 4-bit code in order: **Left**-to-**Right**, **Bottom**-to-**Top**.
5. When a set bit (1) is found, compute the **intersection I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with **I** and repeat the algorithm.

PROGRAM 5.

```
// Cohen-Suderland Line Clipping Algorithm with Window to viewport Mapping
*/ #include <stdio.h>
#include <GL/glut.h>
```

```
#include<stdbool.h>

#define outcode int;
double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries double
xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries //bit
codes for the right, left, top, & bottom
const int RIGHT = 8;
const int LEFT = 2;
const int TOP = 4; const
int BOTTOM = 1;

//used to compute bit codes of a point
outcode ComputeOutCode (double x, double y);


/*Cohen-Sutherland clipping algorithm clips a line from P0 = (x0, y0) to P1 = (x1, y1) against a
rectangle with diagonal from (xmin, ymin) to (xmax, ymax). */

void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)
{
        //Outcodes for P0, P1, and whatever point lies outside the clip
        rectangle outcode outcode0, outcode1, outcodeOut;
        bool accept = false, done = false;


        //compute outcodes
        outcode0 = ComputeOutCode (x0, y0);
        outcode1 = ComputeOutCode (x1, y1);
        do{
                if (!(outcode0 | outcode1))     //logical or is 0 Trivially accept & exit
                {
                        accept = true;
                        done = true;
                }
                else if (outcode0 & outcode1) //logical and is not 0. Trivially reject and exit
                        done = true;
                else
                {
//failed both tests, so calculate the line segment to clip from an outside point to an intersection with clip
                        edge double x, y;


                        //At least one endpoint is outside the clip rectangle; pick it.
```

```
                outcodeOut = outcode0? outcode0: outcode1;

//Now find the intersection point, use formulas y = y0 + slope * (x - x0), x = x0 + (1/slope)* (y -
                y0) if (outcodeOut & TOP) //point is above the clip rectangle
                {
                        x = x0 + (x1 - x0) * (ymax - y0)/(y1 -
                        y0); y = ymax;
                }
                else if (outcodeOut & BOTTOM)  //point is below the clip rectangle
                {
                        x = x0 + (x1 - x0) * (ymin - y0)/(y1 -
                        y0); y = ymin;
                }
                else if (outcodeOut & RIGHT)  //point is to the right of clip rectangle
                {
                        y = y0 + (y1 - y0) * (xmax - x0)/(x1 -
                        x0); x = xmax;
                }
                else                    //point is to the left of clip rectangle
                {
                        y = y0 + (y1 - y0) * (xmin - x0)/(x1 -
                        x0); x = xmin;
                }

//Now we move outside point to intersection point to clip and get ready for next
                pass. if (outcodeOut == outcode0)
                {
                        x0 = x;
                        y0 = y;
                        outcode0 = ComputeOutCode (x0, y0);
                }
                else
                {
                        x1 = x;
                        y1 = y;
                        outcode1 = ComputeOutCode (x1, y1);
                }
        }

    }while (!done);
```

if (accept)

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

/*Compute the bit code for a point (x, y) using the clip rectangle bounded diagonally by (xmin, ymin),
and (xmax, ymax) */
outcode ComputeOutCode (double x, double y)
{
        outcode code = 0;
        if (y > ymax)           //above the clip window
                code |= TOP;
        else if (y < ymin)      //below the clip window
                code |= BOTTOM;
        if (x > xmax)           //to the right of clip window
                code |= RIGHT;
        else if (x < xmin)      //to the left of clip window
                code |= LEFT;
        return code;
}

void display()
{
```

```
double x0=60,y0=20,x1=80,y1=120;
glClear(GL_COLOR_BUFFER_BIT);
//draw the line with red color
glColor3f(1.0,0.0,0.0);
//bres(120,20,340,250);
glBegin(GL_LINES);
                glVertex2d (x0, y0);
                glVertex2d (x1, y1);
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();
CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
glFlush();
}

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

void main(int argc, char** argv)
{
        //int x1, x2, y1, y2;
        //printf("Enter End points:");
        //scanf("%d%d%d%d", &x1,&x2,&y1,&y2);

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

```
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Cohen Suderland Line Clipping
        Algorithm"); glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

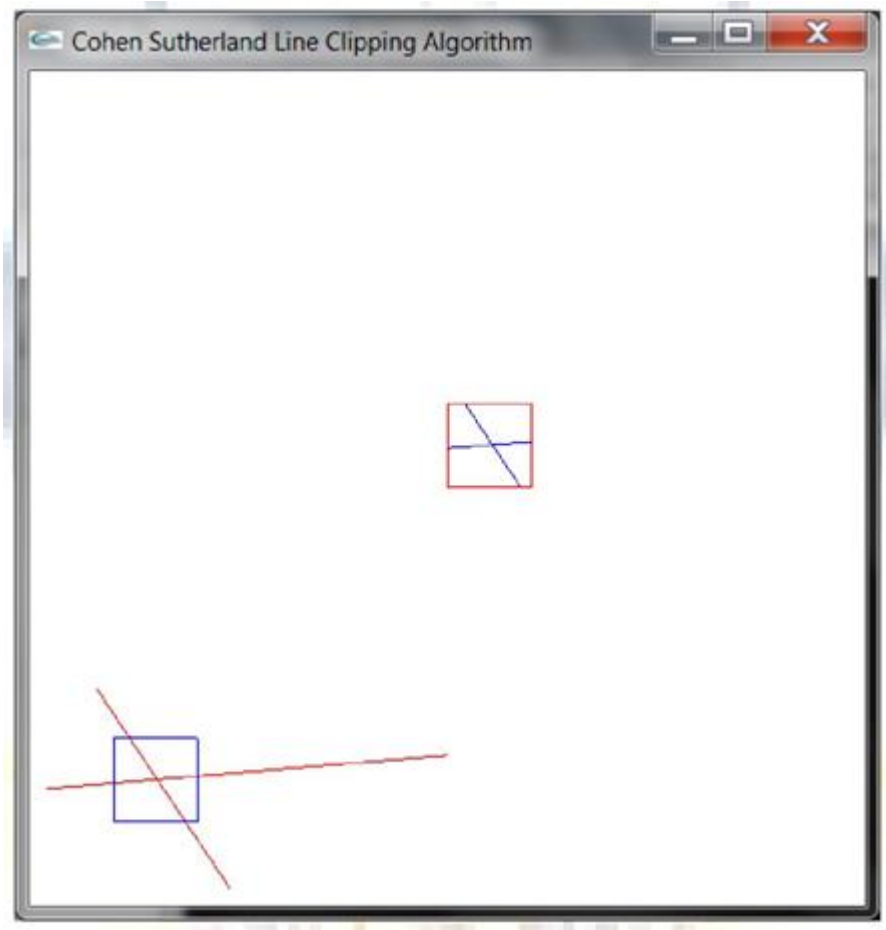
SAMPLE INPUT AND OUTPUT

cc -o 5 5.c -lGL -lGLU -lglut

Run 1:

In display function, the coordinates as follows:

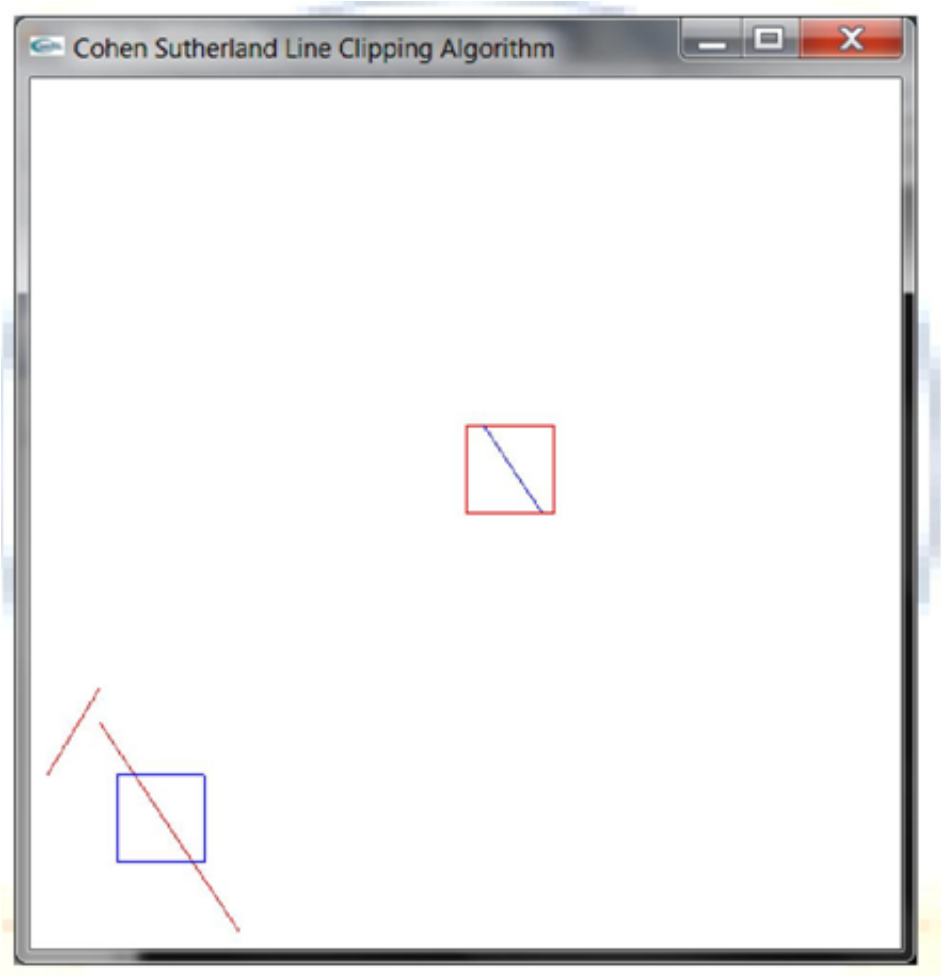
```
glBegin(GL_LINES);
                glVertex2d (x0, y0); glVertex2d
                (x1, y1); glVertex2d (10, 70);
                glVertex2d (250, 90);
glEnd();
CohenSutherlandLineClipAndDraw(10,70,250,90);
```



Run 2:

In display function make changes in the coordinates as follows:

```
glBegin(GL_LINES);
                    glVertex2d (x0, y0); glVertex2d
                    (x1, y1); glVertex2d (10,100);
                    glVertex2d (40,150);
glEnd();
CohenSutherlandLineClipAndDraw(10,100,40,150);
```



Q No. 6: To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

DESCRIPTION

The cube is scaled in different axis's to form walls, table legs and table top. The Tea pot is placed on the table. Color and shading for all the objects and Lightning is provided with the help of openGL functions.

OBJECTIVE

- Understand the complexities of Lighting and shading
- Learn to use openGL function to draw 3D objects.

OUTCOME

- Students will be able to demonstrate how real-world lighting conditions are approximated by

OpenGL

- Able to render illuminated objects by defining the different properties of light and material.

ALGORITHM

7. Initialize ambient, diffuse, specular & shininess material properties. Also initialize light intensity & position.
8. Define main()
    a. Enable display mode as single, RGB and depth buffer.
    b. Register shading with GL_SMOOTH.
    c. Use viewport for (0,0,xmax,ymax)
    d. Enable LIGHTING, LIGHT0, depth test & normalize.
9. Define reshape by gluLookAt() & clear() with color & depth buffer bit.
10. With the help of glutSolidCube () form a table (legs &top) and 3D wall. Use transformation functions translate, rotate and scale prefixed and suffixed by glPushMatrix() and PopMatrix() respectively.
11. Define display()
    a. Set ambient, diffuse, specular & shininess for glMaterials ().
    b. Set light intensity & position for glLightfv()
    c. Call draw_wall() & draw_table().
    d. Use glutSolidTeapot() to place it on the table.

PROGRAM 6.

```
/* simple shaded scene consisting of a tea pot on a table
*/ #include <GL/glut.h>

void wall (double thickness)
{
        //draw thin wall with top = xz-plane, corner at origin
```

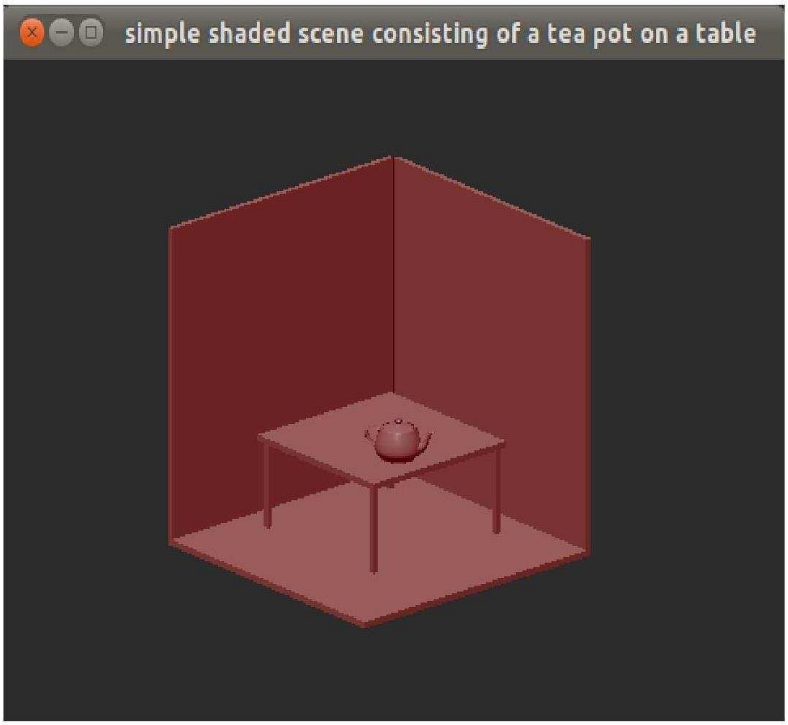
```
        glPushMatrix();
                glTranslated (0.5, 0.5 * thickness, 0.5);
                glScaled (1.0, thickness, 1.0);
                glutSolidCube (1.0);
        glPopMatrix();
}

//draw one table leg
void tableLeg (double thick, double len)
{
        glPushMatrix();
                glTranslated (0, len/2, 0);
                glScaled (thick, len, thick);
                glutSolidCube (1.0);
        glPopMatrix();
}

void table (double topWid, double topThick, double legThick, double legLen)
{
        //draw the table - a top and four
        legs //draw the top first
        glPushMatrix();
                glTranslated (0, legLen, 0);
                glScaled(topWid, topThick,
                topWid); glutSolidCube (1.0);
        glPopMatrix();

        double dist = 0.95 * topWid/2.0 -
        legThick/2.0; glPushMatrix();
                glTranslated (dist, 0, dist);
                tableLeg (legThick, legLen);
                glTranslated (0.0, 0.0, -2 * dist);
                tableLeg (legThick, legLen);
                glTranslated (-2*dist, 0, 2 *dist);
                tableLeg (legThick, legLen);
                glTranslated(0, 0, -2*dist);
                tableLeg (legThick, legLen);
        glPopMatrix();
}

void displaySolid (void)
```

```
{
        //set properties of the surface material
        GLfloat mat_ambient[] = {1.0f, 0.0f, 0.0f, 1.0f}; // red
        GLfloat mat_diffuse[] = {.5f, .5f, .5f, 1.0f}; //white
        GLfloat mat_specular[] = {1.0f, 1.0f, 1.0f, 1.0f}; //white
        GLfloat mat_shininess[] = {75.0f};

        glMaterialfv (GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv (GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv (GL_FRONT, GL_SHININESS, mat_shininess);

        //set the light source properties
        GLfloat lightIntensity[] = {0.7f, 0.7f, 0.7f, 1.0f}; GLfloat
        light_position[] = {2.0f, 6.0f, 3.0f, 0.0f}; glLightfv
        (GL_LIGHT0, GL_POSITION, light_position); glLightfv
        (GL_LIGHT0, GL_DIFFUSE, lightIntensity);

        //set the camera
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity();
        double winHt = 1.0; //half-height of window
        glOrtho (-winHt * 64/48.0, winHt*64/48.0, -winHt, winHt, 0.1, 100.0);
        glMatrixMode (GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt (2.3, 1.3, 2.0, 0.0, 0.25, 0.0, 0.0, 1.0, 0.0);

        //start drawing
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glPushMatrix();
                glTranslated (0.6, 0.38, 0.5);
                glRotated (30, 0, 1, 0);
                glutSolidTeapot (0.08);
        glPopMatrix ();

        glPushMatrix();
                glTranslated (0.4, 0, 0.4);
                table (0.6, 0.02, 0.02, 0.3);
        glPopMatrix();
```

```
        wall (0.02);
        glPushMatrix();
                glRotated (90.0, 0.0, 0.0, 1.0);
                wall (0.02);
        glPopMatrix();

        glPushMatrix();
                glRotated (-90.0, 1.0, 0.0, 0.0);
                wall (0.02);
        glPopMatrix();

        glFlush();
}

void main (int argc, char ** argv)
{
        glutInit (&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize (640, 480);
        glutInitWindowPosition (100, 100);
        glutCreateWindow ("simple shaded scene consisting of a tea pot on a
        table"); glutDisplayFunc (displaySolid);
        glEnable (GL_LIGHTING);
        glEnable (GL_LIGHT0);
        glShadeModel (GL_SMOOTH);
        glEnable (GL_DEPTH_TEST);
        glEnable (GL_NORMALIZE);
        glClearColor (0.1, 0.1, 0.1, 0.0);
        glViewport (0, 0, 640, 480);
        glutMainLoop();
}
```

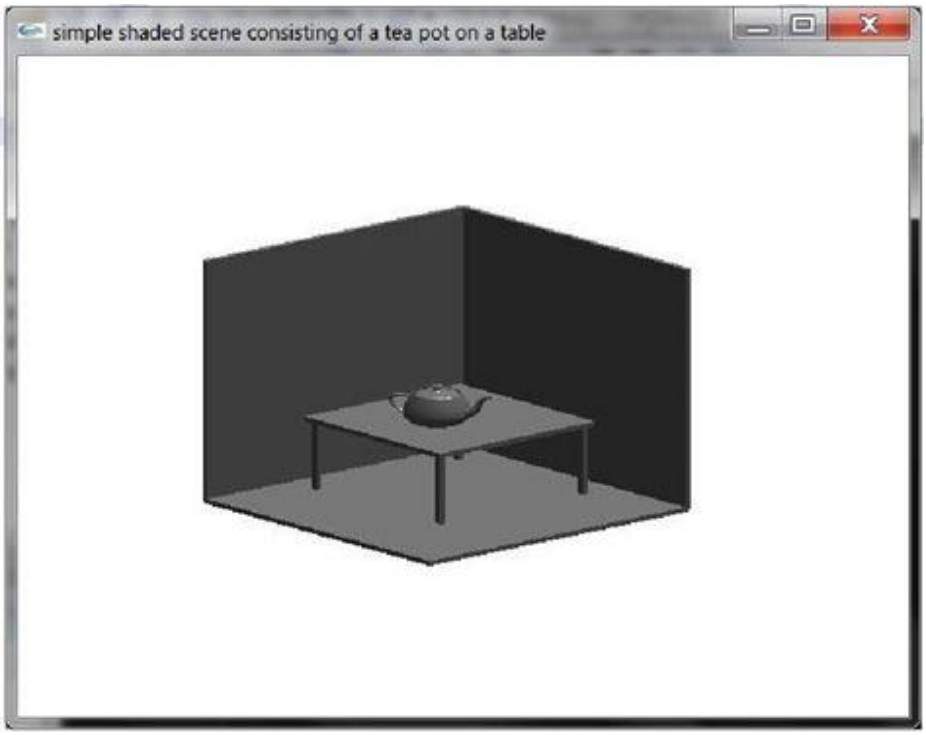
SAMPLE INPUT AND OUTPUT- cc -o 6 6.c -lGL -lGLU -lglut

RUN 1



RUN 2

In displaysolid( ) function, the light_position[]={2.0f,6.0f,3.0f,0.0f };



Q No. 7: Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

DESCRIPTION:

When the user provides a number as input, this number specifies the number of times tetrahedron has to be subdivided to form 3D Sierpinski gasket. This algorithm finds midpoint of each edge of tetrahedron and with the help of one vertex of tetrahedron and midpoints it divides the tetrahedron. The process of finding midpoints and subdividing tetrahedron is a recursive process.

OBJECTIVE:

- To analyze and draw 3D objects
- Learning 3D Sierpinski gasket algorithm

OUTCOME: At the end of this lab session, the student will be able to

- Students will be able create 3D geometric figures using openGL function.

ALGORITHM

1. Initialize 4 vertex points of tetrahedron.
2. Define main()
    a. Enable display mode as single and depth buffer.
    b. Enable depth test.
3. Define Reshape() with glOrtho.
4. Define display()
    a. Call tetrahedron() where 4 different colors are given for each face and call divide_triangle() for each face.
    b. In divide triangle each face is recursively sub-divided into 3 triangles using midpoint of edges, n number of times.
    c. In triangle() a triangle is plotted using polygon primitives.

PROGRAM 7.

```
#include <stdlib.h>
#include <stdio.h>
#include<GL/glut.h>
typedef float point[3]; * initial etrahedron */


point v[]={{0.0, 0.0, 1.0},{0.0, 0.9, -0.3}, {-0.8, -0.5, -0.3}, {0.8, -0.5, -
0.3}; int n;
void triangle( point a, point b, point c)
{
/* display one triangle using a line loop for wire frame, a single normal for constant shading, or
three normals for interpolative shading */
```

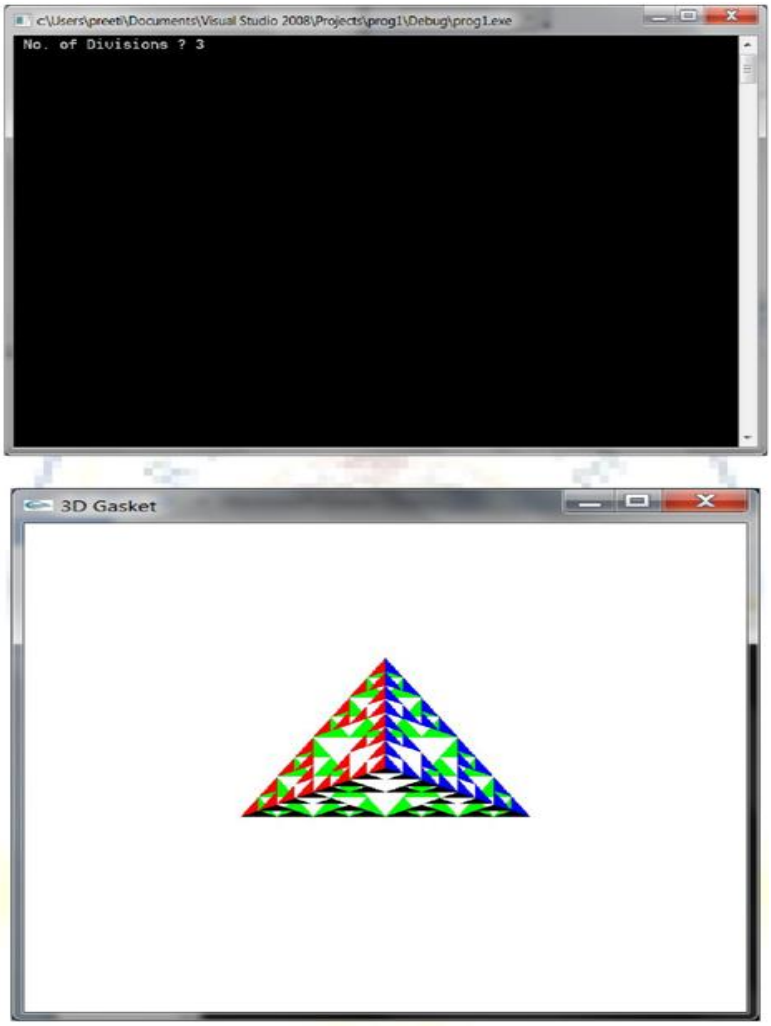
```
glBegin(GL_POLYGON);
        glNormal3fv(a);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
glEnd();
}

void divide_triangle(point a, point b, point c, int m)
{
   // triangle subdivision using vertex numbers right hand rule applied to create outward pointing
   faces 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 triangle(a,b,c); // draw triangle at end of recursion
}

void tetrahedron( int m)
{
/* Apply triangle subdivision to faces of tetrahedron
        */ glColor3f(1.0,0.0,0.0);
         divide_triangle(v[0],  v[1], v[2], m);
         glColor3f(0.0,0.0,0.0);
         divide_triangle(v[3],  v[2], v[1], m);
         glColor3f(0.0,0.0,1.0);
         divide_triangle(v[0],  v[3], v[1], m);
         glColor3f(0.0,1.0,0.0);
         divide_triangle(v[0],  v[2], v[3], m);
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
        tetrahedron(n);
         glFlush();

}

void myReshape(int w, int h)
{
         glClearColor (1.0, 1.0, 1.0, 1.0);
         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);
    glutPostRedisplay();
}


int main(int argc, char **argv)
{
         printf(" No. of Divisions ? ");
         scanf("%d",&n); glutInit(&argc, argv);
         glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
        GLUT_DEPTH); glutInitWindowSize(500, 500);
         glutCreateWindow("3D Gasket");
         glutReshapeFunc(myReshape);
         glutDisplayFunc(display);
         glEnable(GL_DEPTH_TEST);
         glutMainLoop();
         return 1;
}
```

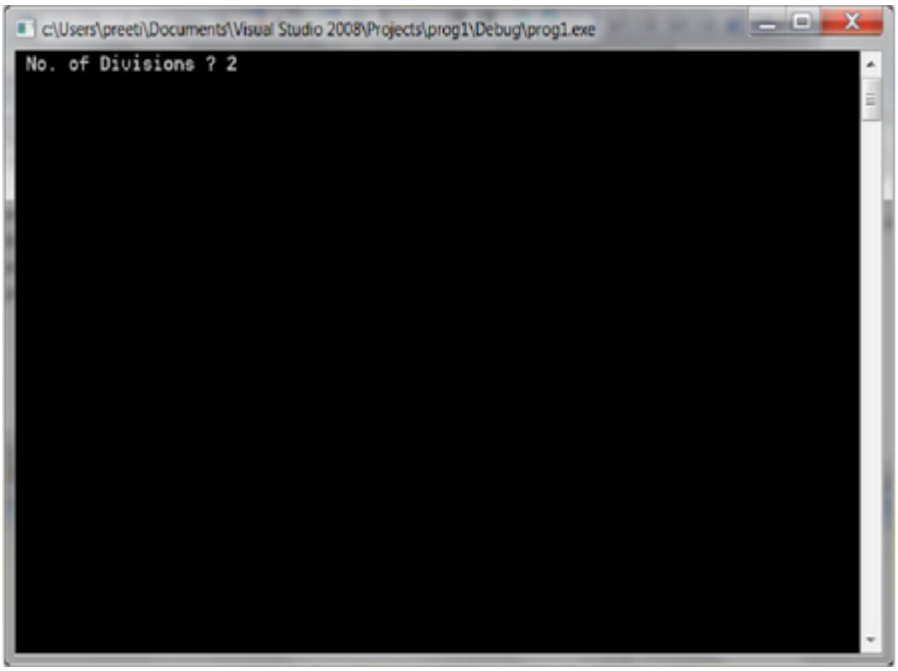
SAMPLE INPUT AND OUTPUT

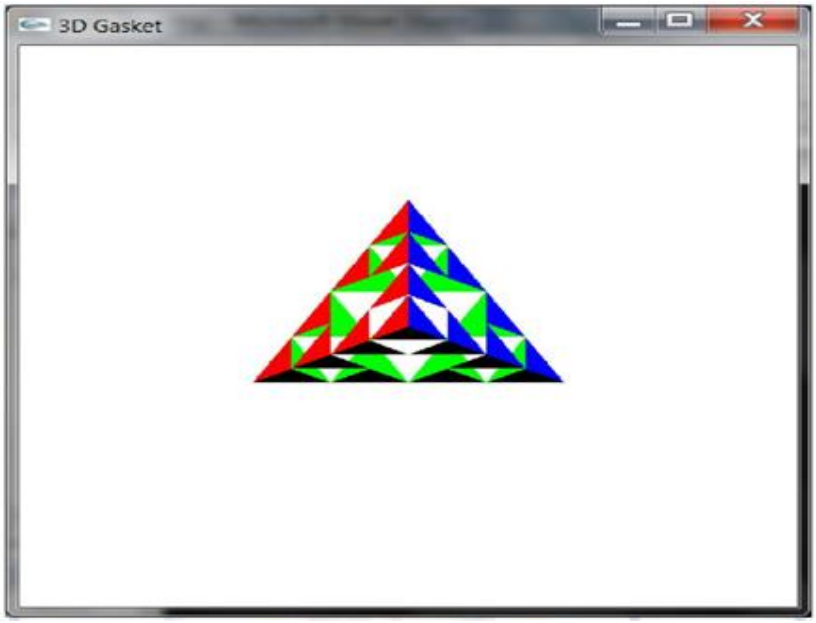
cc -o 7 7.c -lGL -lGLU -lglut

RUN 1


c:\Users\preeti\Documents\Visual Studio 2008\Projects\prog1\Debug\prog1.exe
No. of Divisions ? 3
3D Gasket

RUN 2


c:\Users\preeti\Documents\Visual Studio 2008\Projects\prog1\Debug\prog1.exe
No. of Divisions ? 2


3D Gasket

Q No. 8: Develop a menu driven program to animate a flag using Bezier Curve algorithm

DESCRIPTION

A simple Bezier Curve algorithm.

OBJECTIVE

- To learn Bezier Curve algorithm.

OUTCOME

- Students will be able to implement Bezier Curve algorithm

PROGRAM 8.

```
#include <stdlib.h>
#include <math.h>

/* Set initial size of the display window. */
GLsizei winWidth = 600, winHeight = 600;
/* Set size of world-coordinate clipping window.
*/ GLfloat xwcMin = -50.0, xwcMax = 50.0;
GLfloat ywcMin = -50.0, ywcMax = 50.0;

class wcPt3D
{ public:
GLfloat x, y,
z; };
void init (void)
{
/* Set color of display window to white.
*/ glClearColor (1.0, 1.0, 1.0, 0.0);
}
void plotPoint (wcPt3D bezCurvePt)
{
glBegin (GL_POINTS);
glVertex2f (bezCurvePt.x,
bezCurvePt.y); glEnd ( );
}
/* Compute binomial coefficients C for given value of n.
*/ void binomialCoeffs (GLint n, GLint * C)
{
GLint k, j;
for (k = 0; k <= n; k++) {
/* Compute n!/(k!(n - k)!). */
C [k] = 1;
for (j = n; j >= k + 1; j--)
C [k] *= j;
for (j = n - k; j >= 2; j--)
C [k] /= j;
}
}
void computeBezPt (GLfloat u, wcPt3D * bezPt, GLint
nCtrlPts, wcPt3D * ctrlPts, GLint * C)
{
```

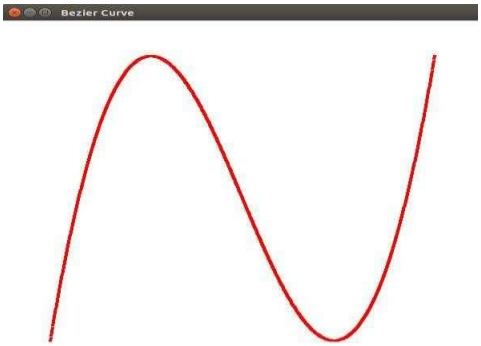
```
GLint k, n = nCtrlPts -
1; GLfloat bezBlendFcn;
bezPt->x = bezPt->y = bezPt->z = 0.0;
/* Compute blending functions and blend control points.
*/ for (k = 0; k < nCtrlPts; k++) {
bezBlendFcn = C [k] * pow (u, k) * pow (1 - u, n -
k); bezPt->x += ctrlPts [k].x * bezBlendFcn;
bezPt->y += ctrlPts [k].y * bezBlendFcn;
bezPt->z += ctrlPts [k].z * bezBlendFcn;
}
}


void bezier (wcPt3D * ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
wcPt3D bezCurvePt;
GLfloat u;
GLint *C, k;
/* Allocate space for binomial coefficients
*/ C = new GLint [nCtrlPts];
binomialCoeffs (nCtrlPts - 1, C);
for (k = 0; k <= nBezCurvePts; k++) {
u = GLfloat (k) / GLfloat (nBezCurvePts);
computeBezPt (u, &bezCurvePt, nCtrlPts, ctrlPts,
C); plotPoint (bezCurvePt);
}
delete [ ] C;
}
void displayFcn (void)
{
/* Set example number of control points and number
of * curve positions to be plotted along the Bezier
curve. */
GLint nCtrlPts = 4, nBezCurvePts = 1000;
wcPt3D ctrlPts [4] = { {-40.0, -40.0, 0.0}, {-10.0, 200.0, 0.0},
{10.0, -200.0, 0.0}, {40.0, 40.0, 0.0} };
glClear (GL_COLOR_BUFFER_BIT); // Clear display window.

glPointSize (4);
glColor3f (1.0, 0.0, 0.0); // Set point color to red.
bezier (ctlPts, nCtrlPts, nBezCurvePts);
 glFlush ( );
 }
 void winReshapeFcn (GLint newWidth, GLint newHeight)
 {
 /* Maintain an aspect ratio of 1.0. */
 glViewport (0, 0, newHeight,
 newHeight); glMatrixMode
 (GL_PROJECTION); glLoadIdentity ( );
 gluOrtho2D (xwcMin, xwcMax, ywcMin,
 ywcMax); glClear (GL_COLOR_BUFFER_BIT);
 }
 int main (int argc, char** argv)
 {
```

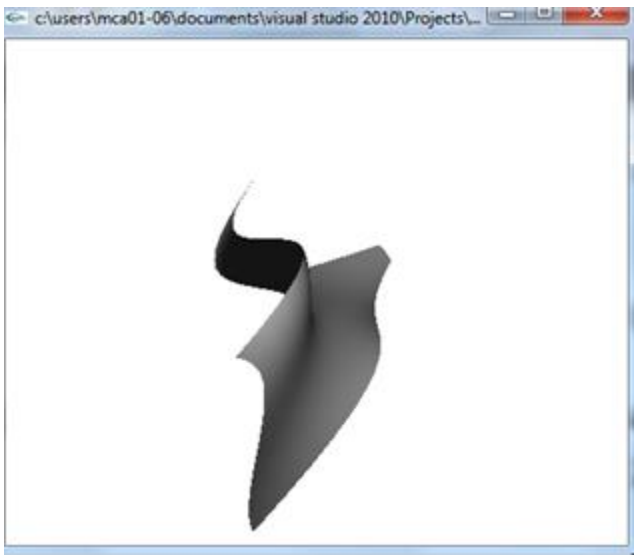
```
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE |
GLUT_RGB); glutInitWindowPosition (50, 50);
glutInitWindowSize (winWidth, winHeight);
glutCreateWindow ("Bezier Curve");
init ( );
glutDisplayFunc (displayFcn);
glutReshapeFunc (winReshapeFcn);
glutMainLoop ( );
}
```

SAMPLE INPUT AND OUTPUT

g++ -o 8 8.c -lGL -lGLU -lglut

./8





Q No. 9: Develop a menu driven program to fill the polygon using scan line algorithm

DESCRIPTION

A simple polygon with four edges is created and color is filled with the help of scan-line area filling algorithm

OBJECTIVE

- To learn scan line polygon fill algorithm.

OUTCOME

- Students will be able to implement scan-line area filling algorithm to fill irregular objects.

ALGORITHM

1. Define main ( ) having display mode as Single Buffer and myinit ( ) having gluOrtho2D.
2. Draw a desired polygon.
3. Initialize left edge to xmax of window and right edge to zero.
4. Conduct edge detect test for all sides of the polygon to compute the left and right edge boundary coordinates for each scan line of the scene.
5. For y ← 0 to ymax
   If le[y] less than re[y] then
   For i ← le[y] to re[y]
   Draw pixel(I,y)

   End for

.

PROGRAM 9

```
// Scan-Line algorithm for filling
a polygon #include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
 float x1,x2,x3,x4,y1,y2,y3,y4;
void edgedetect(float x1,float y1,float x2,float y2,int *le,int *re)
{
float mx,x,temp;
int i;
        if((y2-y1)<0)
        {
                temp=y1;y1=y2;y2=temp;
                temp=x1;x1=x2;x2=temp;
        }



        if((y2-y1)!=0)
                mx=(x2-x1)/(y2-y1);

        else

        x=x1;
mx=x2-x1;
        for(i=y1;i<=y2;i++)
        {
                if(x<(float)le[i])
                        le[i]=(int)x;
                if(x>(float)re[i])
                        re[i]=(int)x;
                x+=mx;
        }
}

void draw_pixel(int x,int y)
{
        glColor3f(1.0,1.0,0.0);
        glBegin(GL_POINTS);
        glVertex2i(x,y);
        glEnd();
}

void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
```

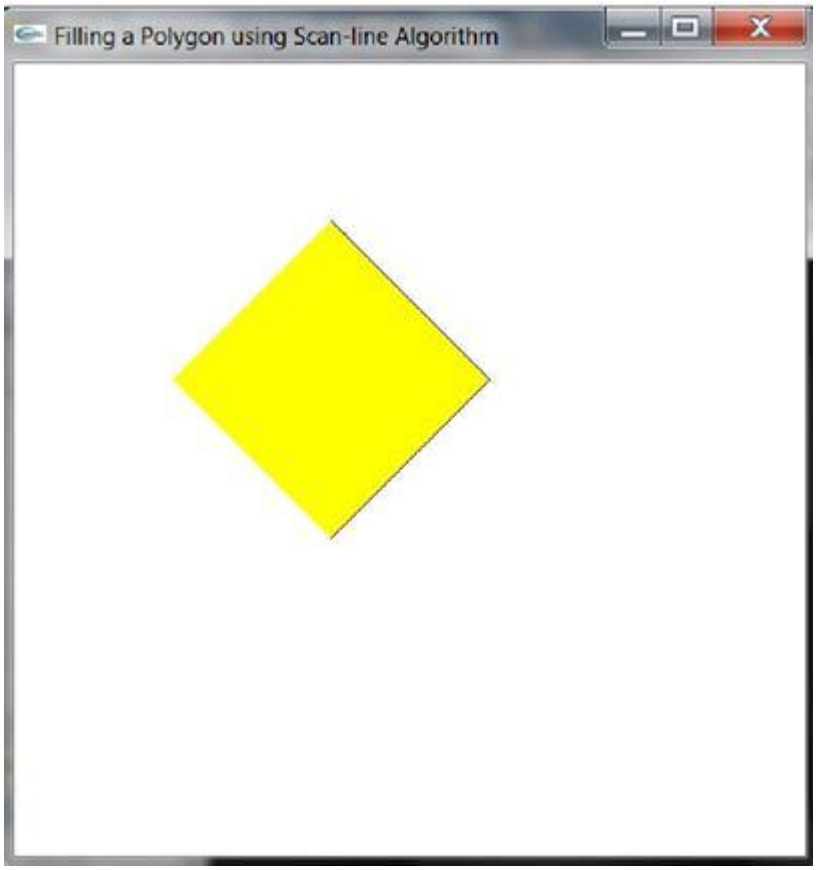
```
{



        int le[500],re[500];
        int i,y;
        for(i=0;i<500;i++)
        {
                le[i]=500;
                re[i]=0;
        }
        edgedetect(x1,y1,x2,y2,le,re);
        edgedetect(x2,y2,x3,y3,le,re);
        edgedetect(x3,y3,x4,y4,le,re);
        edgedetect(x4,y4,x1,y1,le,re);
        for(y=0;y<500;y++)
        {
                if(le[y]<=re[y])
                        for(i=(int)le[y];i<(int)re[y];i++)
                                draw_pixel(i,y);
        }

}

void display()
{
x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;

glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x1,y1);
  glVertex2f(x2,y2);
  glVertex2f(x3,y3);
  glVertex2f(x4,y4);
  glEnd();
  scanfill(x1,y1,x2,y2,x3,y3,x4,y4);

glFlush();
}

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

```
}

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

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Filling a Polygon using Scan-line
        Algorithm"); glutDisplayFunc(display);
        myinit();
        glutMainLoop();
        return 1;
}
```

SAMPLE INPUT AND OUTPUT

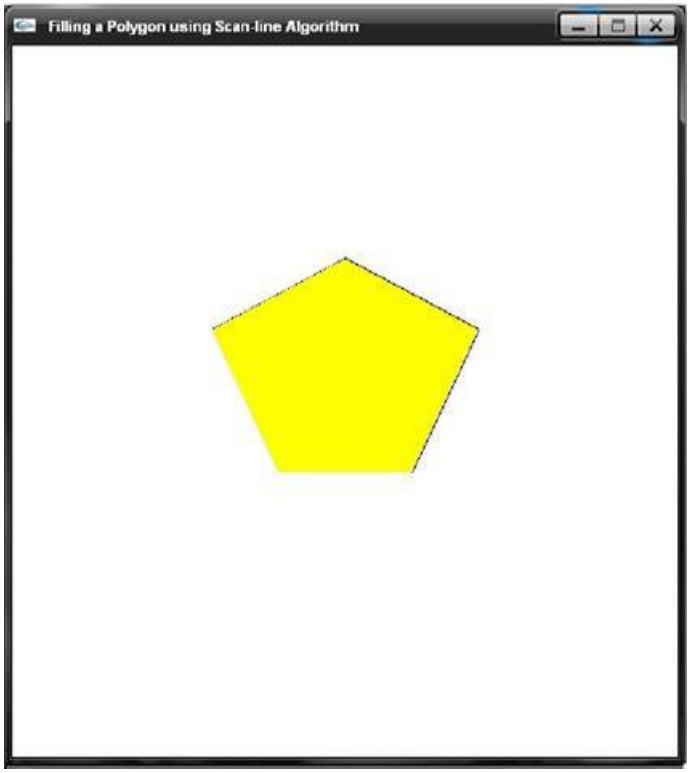
Run 1: In display function,

x1=200.0,y1=200.0;
x2=100.0,y2=300.0;
x3=200.0,y3=400.0;
x4=300.0,y4=300.0;



Run 2: In display function,

x1=200.0,y1=200.0;
x2=300.0,y2=200.0;
x3=350.0,y3=300.0;
x4=250.0,y4=350.0;
x5=150.0,y5=300.0;



VIVA QUESTIONS

1. Explain all the OpenGL functions used in this program?
2. What is the principle of Sierpinski gasket?
3. Difference between additive and subtractive color?
4. What is the Graphics Architecture used in OpenGL?
5. What is Rasterisation?
6. Explain sierpinski Gasket (using points).
7. Explain sierpinski Gasket (using polye)
8. Difference between 2D Tetrohedron and 3D Tetrahedron.
9. What do you mean by clipping?
10. How is Liang-Barsky clipping different from CohenSutherland Clipping Algorithm?
11. How do you set the color attributes in Opengl?
12. What is the command for clearscreen?
13. What is Event callback function?
14. Explain Liang-Barsky line clipping algorithm?
15. Explain window to viewpoint mapping?
16. What are vertex arrays?
17. How are the faces of the color cube modeled?
18. How do you consider the inward and outward pointing of the faces?
19. Explain the OpenGL function used to rotate the color cube?
20. What is the difference between 2D and 3D orthographic projection statements?
21. Explain the Inward and Outward pointing face?
22. Explain the data structure for object representation?
23. Explain the vertex list representation of a cube?
24. What is transformation?
25. Explain the OpenGL functions used for translation, rotation and scaling?

26. What is the order of transformation?
27. State the difference between modelview and projection?
28. What is Homogeneous-coordinate representation?
29. Define the rotation matrix and object matrix.
30. Explain the procedure to obtain the resultant matrix using rotation matrix and object matrix.
31. What is the principle of Cohen-Sutherland Algorithm?
32. State the advantages and disadvantages of Cohen-Sutherland Algorithm?
33. What is an outcode?
34. What is Synthetic Camera Model?
35. What are the Camera Specifications?
36. Explain the cases of outcodes in Cohen-Sutherland algorithm.
37. Explain the cohen-sutherland line clipping algorithm
38. Mention the difference between Liang-Barsky and Cohen-Sutherland line clipping algorithm.
39. Explain about gluLookAt(...), glFrustum(...), gluPerspective?
40. Explain the different types of projections?
41. Explain Z-buffer algorithm?
42. What is antialiasing?
43. What is Center Of Projection(COP), Direction Of Projection(DOP)?

44. What is midpoint circle drawing algorithm
45. How do you get the equation d+=2x+3, d+=2(x-y)+5
46. What is gluOrtho2D function
47. Explain plot pixel function
48. Why do we use GLFlush function in Display
49. Explain Specular, Diffuse and Translucent surfaces.
50. What is ambient light?
51. What is umbra, penumbra?
52. Explain Phong lighting model.
53. Explain glLightfv(...), glMaterialfv(...).
54. What is glutSolidCube function ? what are its Parameters
55. What are the parameters to glScale function
56. Explain Push & Pop matrix Functions
57. What is Materialfv function & its Parameters
58. Explain GLULookAt Function
59. Explain the keyboard and mouse events used in the program?
60. What is Hidden surface Removal? How do you achieve this in OpenGL?
61. What are the functions for creating Menus in OpenGL?
62. Explain about fonts in GLUT?
63. Explain about glutPostRedisplay()?
64. Explain how the cube is constructed
65. Explain rotate function
66. What is GLFrustum function what are its Parameters
67. What is viewport
68. What is glutKeyboard Function what are its Parameters
69. Explain scanline filling algorithm?
70. What is AspectRatio,Viewport?
71. How do you use timer?
72. What are the different frames in OpenGL?
73. What is fragment processing?
74. Explain Polygon Filling algorithm
75. What is slope
76. How the edges of polygon are detected
77. Why you use GL_PROJECTION in MatrixMode Function
78. What is dx & dy
79. What is maxx & maxy
80. What is glutPostRedisplay
81. Why do we use glutMainLoop function
82. What do you mean by GL_LINE_LOOP in GL_Begin function
83. Define Computer Graphics.
84. Explain any 3 uses of computer graphics applications.
85. What are the advantages of DDA algorithm?
86. What are the disadvantages of DDA algorithm?
87. Define Scan-line Polygon fill algorithm.
88. What are Inside-Outside tests?
89. Define Boundary-Fill algorithm.
90. Define Flood-Fill algorithm.
91. Define attribute parameter. Give examples.
92. What is a line width? What is the command used to draw the thickness of lines.
93. What are the three types of thick lines? Define.
94. What are the attribute commands for a line color?

95. What is color table? List the color codes.
96. What is a marker symbol and where it is used?
97. Discuss about inquiry functions.
98. Define translation and translation vector.
99. Define window and view port.
100. Define viewing transformation.
101. Give the equation for window to viewport transformation.
102. Define view up vector.
103. What is meant by clipping? Where it happens?
104. What is point clipping and what are its inequalities?
105. What is line clipping and what are their parametric representations?
106. How is translation applied?
107. What is referred to as rotation?
108. Write down the rotation equation and rotation matrix.
109. Write the matrix representation for scaling, translation and rotation.
110. Draw the block diagram for 2D viewing transformation pipeline.
111. Mention the equation for homogeneous transformation.
112. What is known as composition of matrix?
113. Write the composition transformation matrix for scaling, translation and Rotation.
114. Discuss about the general pivot point rotation?
115. Discuss about the general fixed point scaling.
116. Explain window, view port and window - to - view port transformation
117. Mention the three raster functions available in graphics packages.
118. What is known as region codes?
119. Why Cohen Sutherland line clipping is popular?
120. Mention the point clipping condition for the liang-barsky line clipping.
121. What is called as an exterior clipping?
122. How is the region code bit values determined?
123. Why liang-barsky line clipping is more efficient than Cohen Sutherland line Clipping?
124. Differentiate uniform and differential scaling
125. Explain soft fill procedures.
126. Explain the three primary color used in graphics
127. Explain in detail about color and grey scale levels?
128. Explain color and grey scale levels.
129. Explain the area fill attributes and character attributes.
130. Explain character attributes in detail.
131. Briefly discuss about basic transformations.
132. Explain matrix representations.
133. Discuss about composite transformations.
134. Explain about reflection and shear.
135. Explain the following transformation with the matrix representations.
    Give suitable diagram for illustration translation .ii scaling.iii rotation.
136. How the rotation of an object about the pivot point is performed?
137. How window-to-viewport coordinate transformation happens.
138. Explain clipping with its operation in detail.
139. Explain Cohen- Sutherland line clipping.
140. Discuss the logical classifications of input devices.
141. Explain the details of 2d viewing transformation pipeline.
142. Explain point, line, curve, text, exterior clipping?
143. Discuss the properties of light.
144. Define chromaticity, complementary colors, color gamut and primary colors.

145. What is color model?
146. Define hue, saturation and value.
147. Explain XYZ color model.
148. Explain RGB color model.
149. Explain YIQ color model.
150. Explain CMY color model.
151. Explain HSV color model.
152. Give the procedure to convert HSV & RGB color model.
153. What is the use of chromaticity diagram?
154. What is illumination model?
155. What are the basic illumination models?
156. What is called as lightness?
157. Explain the conversion of CMY to RGB representation.
158. What is animation?
159. Define Morphing.
160. What are the steps involved in designing an animation sequence?
161. How to draw dots using OPENGL?
162. How to draw lines using OPENGL?
163. How to draw convex polygons using OPENGL?
164. What is the command used in OPENGL to clear the screen?
165. What are the various OPENGL data types?