Develop a program to draw a line using Bresenham’s line drawing technique

```
//Header File inclusion
#include<stdio.h>
#include<math.h>
#include<glut.h>
int  X1, Y1, X2, Y2;
// Function to draw a pixel
void draw_pixel(int x, int y)
{
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
}
//algorithm to find next pixels
void LineBres( )
{
glClear(GL_COLOR_BUFFER_BIT);
int dx = abs(X2 - X1), dy = abs(Y2 - Y1);
int p = 2 * dy - dx;
int twoDy = 2 * dy, twoDyDx = 2 * (dy - dx);
int x, y;
if (X1 > X2)
{
x = X2;
y = Y2;
X2 = X1;
}
else
{
x = X1;
y = Y1;
X2 = X2;
}
draw_pixel(x, y);
while (x < X2)
{
x++;
if (p < 0)
p += twoDy;
else
{
y++;
p += twoDyDx;
}
draw_pixel(x, y);
}
glFlush();
}
```

```
// Init Function
void Init()
{
glClearColor(1,1,1,1);// White Background
glColor3f(0,0,0);//Black writing Color
glPointSize(2.0);//pointsize=2
glViewport(0, 0, 500, 500);
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
gluOrtho2D(0, 500, 0, 500);// min max of x and y is 0 and 500
glMatrixMode(GL_MODELVIEW);
}
void main()
{
printf("enter two points for draw line Bresenham:\n");
printf("\n enter point1(X1 Y1):");
scanf_s("%d%d", &X1, &Y1);
printf("\n enter point2(X2 Y2):");
scanf_s("%d%d", &X2, &Y2);
glutInitWindowSize(300, 400);
glutInitWindowPosition(0, 0);
glutCreateWindow("LineBresenham");
Init( );
glutDisplayFunc(LineBres);
glutMainLoop();
}
```

Develop a program to demonstrate basic geometric operations on the 2D object

```
#include<glut.h>
#include<stdio.h>

/* initial triangle */

float v[3][2] = { {0, 1},
        {-0.5, -0.5}, {0.5,-0.5} };

int n; /* number of subdivisions */
void triangle(float* a, float* b, float* c)
{
  /* display one triangle  */
  glBegin(GL_TRIANGLES);
  glVertex2fv(a);
  glVertex2fv(b);
  glVertex2fv(c);
  glEnd();
}
void divide_triangle(float* a, float* b, float* c, int m)
{
  /* triangle subdivision using vertex numbers */
  float v1[2], v2[2], v3[2];
  int j;
  if (m > 0)
  {
    for (j = 0; j < 2; j++) v1[j] = (a[j] + b[j]) / 2;
    for (j = 0; j < 2; j++) v2[j] = (a[j] + c[j]) / 2;
    for (j = 0; j < 2; j++) v3[j] = (b[j] + c[j]) / 2;
    divide_triangle(a, v1, v2, m - 1);
    divide_triangle(v1, b, v3, m - 1);
    divide_triangle(v2, v3, c, m - 1);
  }
  else
    triangle(a, b, c);
  /* draw triangle at end of recursion */
}
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  divide_triangle(v[0], v[1], v[2], n);
  glFlush();
}

void myinit()
{
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-2, 2, -2, 2);
```

```
    glMatrixMode(GL_MODELVIEW);
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
}
void main()
{
    printf("How many subdivisions ? : ");
    scanf_s("%d", &n);
    glutInitWindowSize(500, 500);
    glutCreateWindow("2D Gasket");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

Develop a program to demonstrate basic geometric operations on the 3D object

3d gasket - prgm3

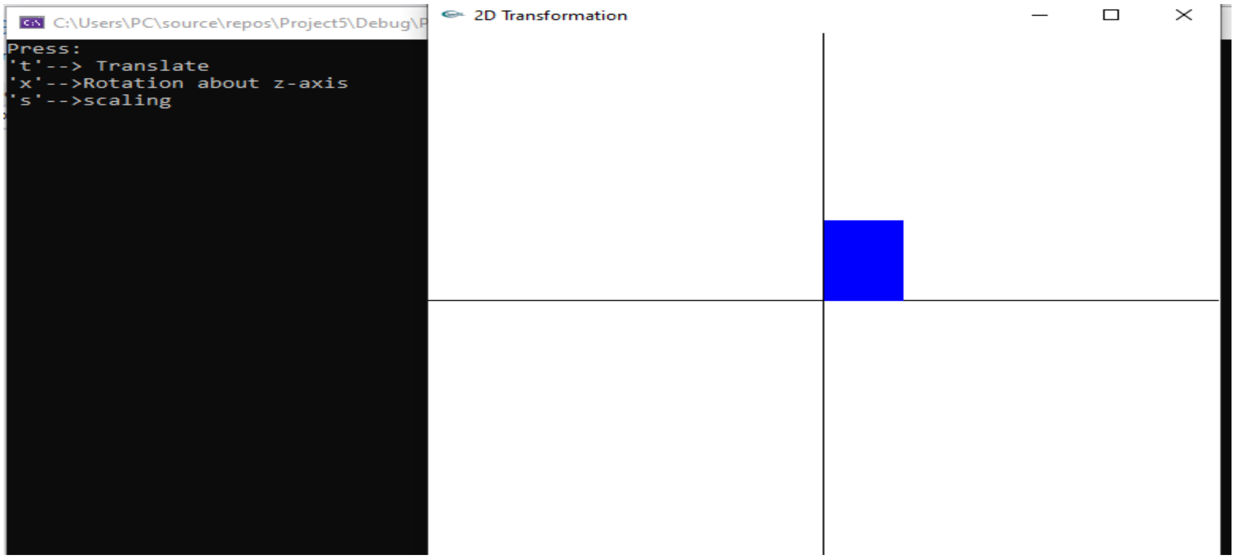
```
#include<stdio.h>
#include<glut.h>
float v[4][3] = { {0,0,1},{0,1,0},{-1,-0.5,0},{1,-0.5,0} };
float colors[4][3] = { {1,0,0},{0,1,0}, { 0,0,1 }, { 0,0,0 } };
int n;
void triangle(float* va, float* vb, float* vc)
{
	glBegin(GL_TRIANGLES);
	glVertex3fv(va);
	glVertex3fv(vb);
	glVertex3fv(vc);
	glEnd();
}
void tetra(float* a, float* b, float* c, float* d)
{
	glColor3fv(colors[0]);
	triangle(a, b, c);
	glColor3fv(colors[1]);
	triangle(a, c, d);
	glColor3fv(colors[2]);
	triangle(a, d, b);
	glColor3fv(colors[3]);
	triangle(b, d, c);

}
void divide_tetra(float* a, float* b, float* c, float* d, int m)
{
	float mid[6][3];
	int j;
	if (m > 0)
	{
		for (j = 0; j < 3; j++)mid[0][j] = (a[j] + b[j]) / 2;
		for (j = 0; j < 3; j++)mid[1][j] = (a[j] + c[j]) / 2;
		for (j = 0; j < 3; j++)mid[2][j] = (a[j] + d[j]) / 2;
		for (j = 0; j < 3; j++)mid[3][j] = (b[j] + c[j]) / 2;
		for (j = 0; j < 3; j++)mid[4][j] = (c[j] + d[j]) / 2;
		for (j = 0; j < 3; j++)mid[5][j] = (b[j] + d[j]) / 2;
		divide_tetra(a, mid[0], mid[1], mid[2], m - 1);
		divide_tetra(mid[0], b, mid[3], mid[5], m - 1);
		divide_tetra(mid[1], mid[3], c, mid[4], m - 1);
```

```
        divide_tetra(mid[2], mid[5], mid[4], d, m - 1);

    }
    else
        tetra(a, b, c, d);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    divide_tetra(v[0], v[1], v[2], v[3], n);
    glFlush();
}
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-2, 2, -2 * (float)h / (float)w, 2 * (float)h / (float)w, -10, 10);
    else
        glOrtho(-2 * (float)w / (float)h, 2 * (float)w / (float)h, -2, 2, -10, 10);
    glMatrixMode(GL_MODELVIEW);
}
int main()
{
    printf("no. of divisions");
    scanf_s("%d", &n);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D gasket");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glutMainLoop();

}
```

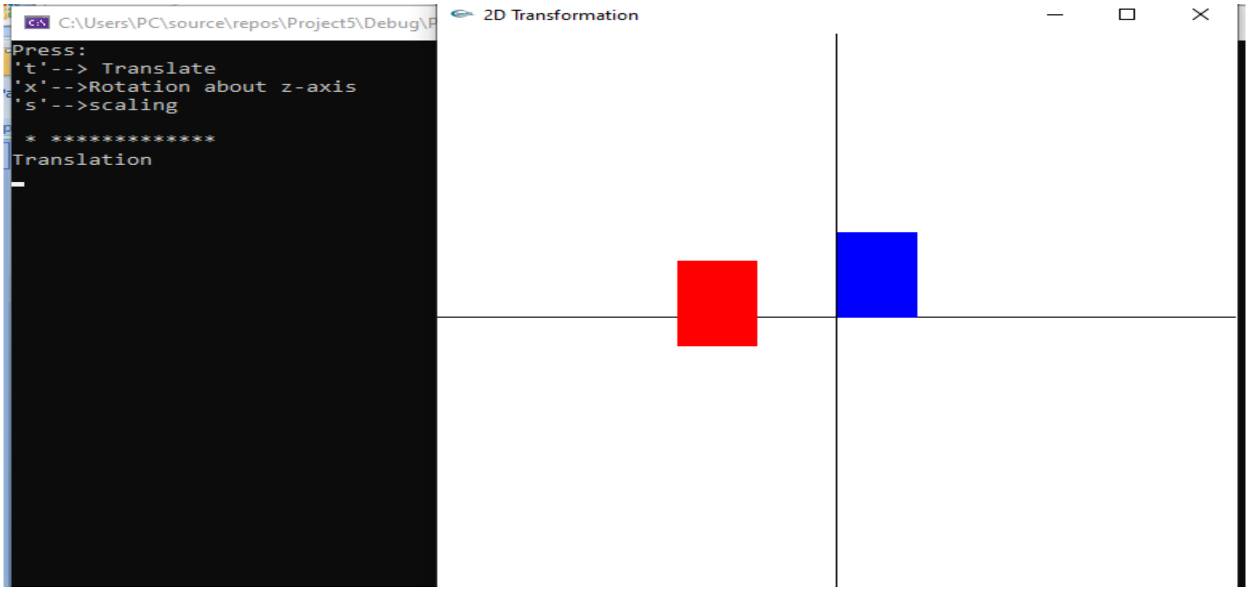
Develop a program to demonstrate 2D transformation on basic objects

```
#include<glut.h>
#include<stdio.h>
char T;
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0, 0, 0);
        glLoadIdentity();
        glBegin(GL_LINES);
        glVertex2f(-499, 0);
        glVertex2f(499, 0);
        glVertex2f(0, -499);
        glVertex2f(0, 499);
        glEnd();
        glColor3f(0, 0, 1);
        glLoadIdentity(); // Reset current matrix to identity.
        glRecti(0, 0, 100, 150); // Display blue rectangle.
        if (T == 't')
        {       printf("\n * *************\nTranslation\n");
                glLoadIdentity(); // Reset current matrix to identity.
                glColor3f(1, 0, 0);
                glTranslatef(-200.0, -50.0, 0.0); // Set translation parameters.
                glRecti(0, 0, 100, 150); // Display red, translated rectangle.
        }
        if (T == 'r')
        {       printf("\n * *************\nRotation about z-axis\n");
                glLoadIdentity(); // Reset current matrix to identity.
                glColor3f(0, 1, 0);
                glRotatef(45, 0.0, 0.0, 1.0); // Set 90-deg. rotation about z axis.
                glRecti(0, 0, 100, 150); // Display red, rotated rectangle.
        }
        if (T == 's')
        {       printf("\n * *************\nScaling\n");
                glLoadIdentity(); // Reset current matrix to identity.
                glColor3f(0, 1, 1);
                glScalef(0.5, 0.5, 0); // Set scale-reflection parameters.
                glRecti(0, 0, 100, 150); // Display red, transformed rectangle.
        }
        glFlush();
}
```

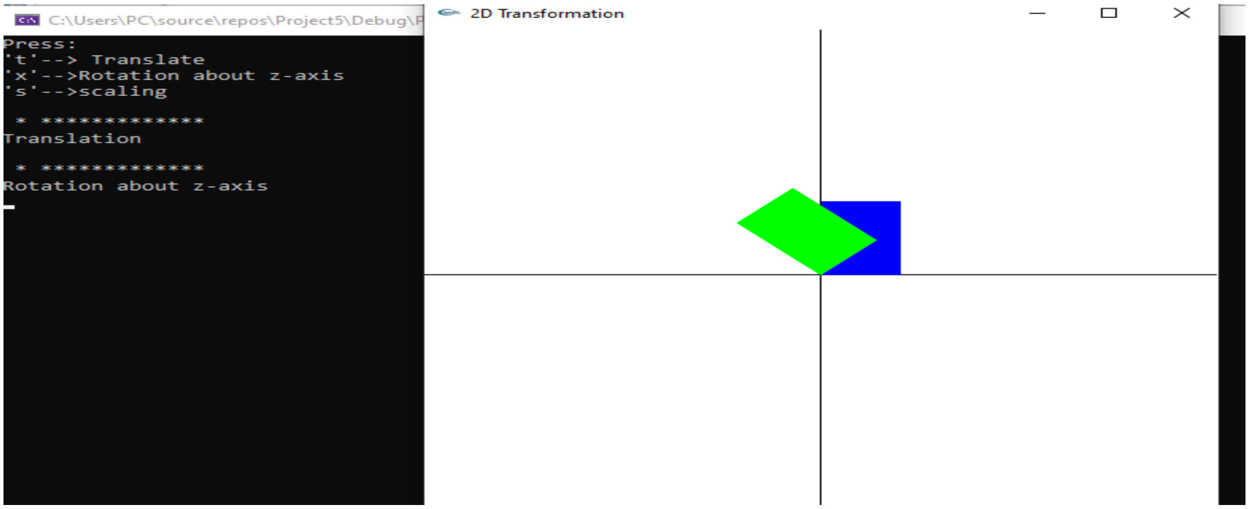
```
void keys(unsigned char k, int x, int y)
{
        T = k;
        display();
}
// Init Function
void Init()
{
        glClearColor(1, 1, 1, 1);// White Background
        glColor3f(0, 0, 0);//Black writing Color
        glPointSize(2.0);//pointsize=2
        glViewport(0, 0, 500, 500);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-500, 500, -500, 500);// min max of x and y is -500 and 500
        glMatrixMode(GL_MODELVIEW);
}
void main()
{
        printf("Press:\n't'--> Translate\n'r'-->Rotation about z-axis\n's'-->scaling\n");
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(300, 50);
        glutCreateWindow("2D Transformation");
        Init();
        glutDisplayFunc(display);
        glutKeyboardFunc(keys);
        glutMainLoop();
}
```



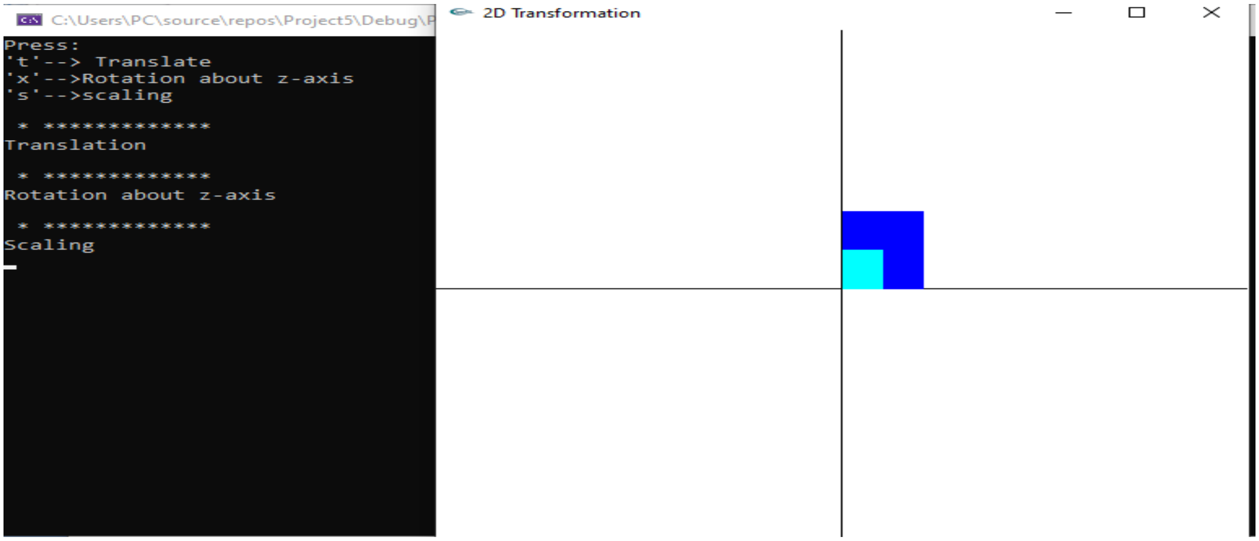
```
Press:
't'--> Translate
'x'-->Rotation about z-axis
's'-->scaling
```



```
Press:
't'--> Translate
'x'-->Rotation about z-axis
's'-->scaling

 * *************
Translation
```



```
Press:
't'--> Translate
'x'-->Rotation about z-axis
's'-->scaling

 * *************
Translation

 * *************
Rotation about z-axis
```



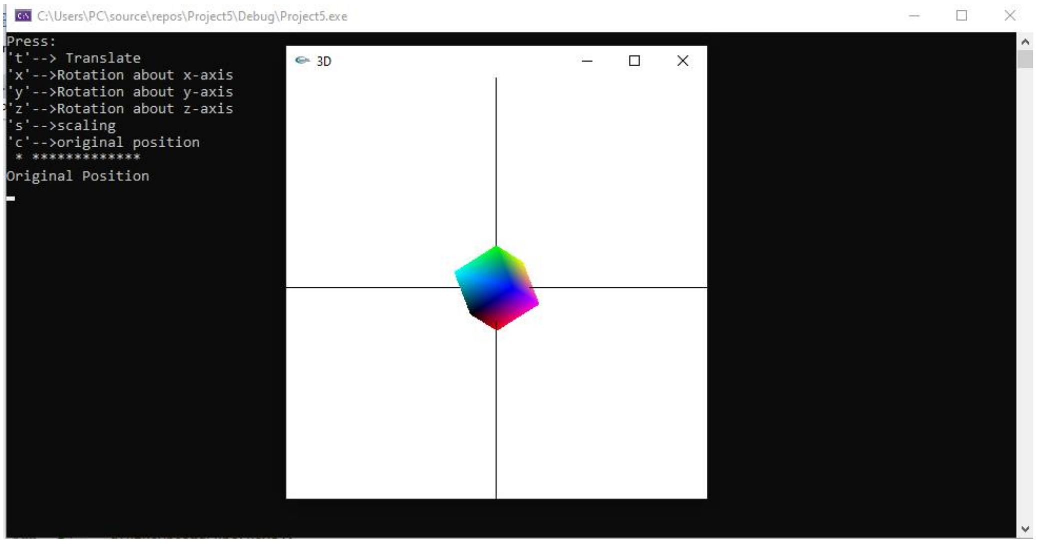
```
Press:
't'--> Translate
'x'-->Rotation about z-axis
's'-->scaling

 * *************
Translation

 * *************
Rotation about z-axis

 * *************
Scaling
```

Develop a program to demonstrate 3D transformation on 3D objects

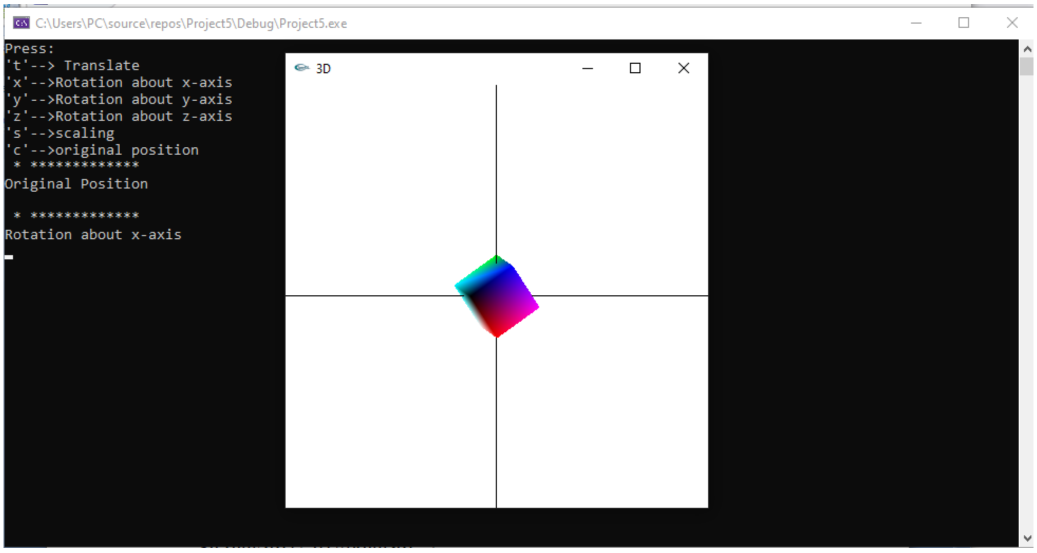
```
#include<glut.h>
#include<stdio.h>
char T='c';
float vertices[] = { -1,-1,-1,1,-1,-1,1,1,-1, -1,1,-1, -1,-1, 1, 1,-1,1, 1,1,1, -1,1,1 };
float colors[] = { 0,0,0,0,0,1,0,1,0,0,1,1,1,0,0,1,0,1,1,1,0,1,1,1 };
unsigned char cubeIndices[] ={ 0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4 };
void cube()
{
glScalef(0.25, 0.25, 0.25);
glRotatef(45, 1,1,1);
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices); // Display cube
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(0, 0, 0);
        glLoadIdentity();
        glBegin(GL_LINES);
        glVertex3f(-1.9, 0,0);
        glVertex3f(1.9, 0,0);
        glVertex3f(0, -1.9,0);
        glVertex3f(0, 1.9,0);
        glEnd();
        if (T == 'c')
        {       printf("\n * *************\nOriginal Position\n");
                glLoadIdentity(); // Reset current matrix to identity.
                cube();
        }
        if (T == 't')
        {       printf("\n * *************\nTranslation\n");
                glLoadIdentity(); // Reset current matrix to identity.
                glTranslatef(-0.5, -0.2, 0.0); // Set translation parameters.
                cube();
        }
```

```
    if (T == 'x')
    {   printf("\n * *************\nRotation about x-axis\n");
        glLoadIdentity(); // Reset current matrix to identity.
        glRotatef(45, 1,0,0); // Set 90-deg. rotation about z axis.
        cube();
    }
    if (T == 'y')
    {   printf("\n * *************\nRotation about y-axis\n");
        glLoadIdentity(); // Reset current matrix to identity.
        glRotatef(45,0,1,0); // Set 90-deg. rotation about z axis.
        cube();
    }
    if (T == 'z')
    {   printf("\n * *************\nRotation about z-axis\n");
        glLoadIdentity(); // Reset current matrix to identity.
        glRotatef(45, 0,0,1); // Set 90-deg. rotation about z axis.
        cube();
    }
    if (T == 's')
    {   printf("\n * *************\nScaling\n");
        glLoadIdentity(); // Reset current matrix to identity.
        glScalef(0.5, 0.5, 0.5); // Set scale-reflection parameters.
        cube();
    }
    glFlush();
}
void keys(unsigned char k, int x, int y)
{

    if (k == 'x' || k == 'y' || k == 'z' || k == 't' || k == 's')
        T = k;
    else
        T = 'c';
    display();
}
```

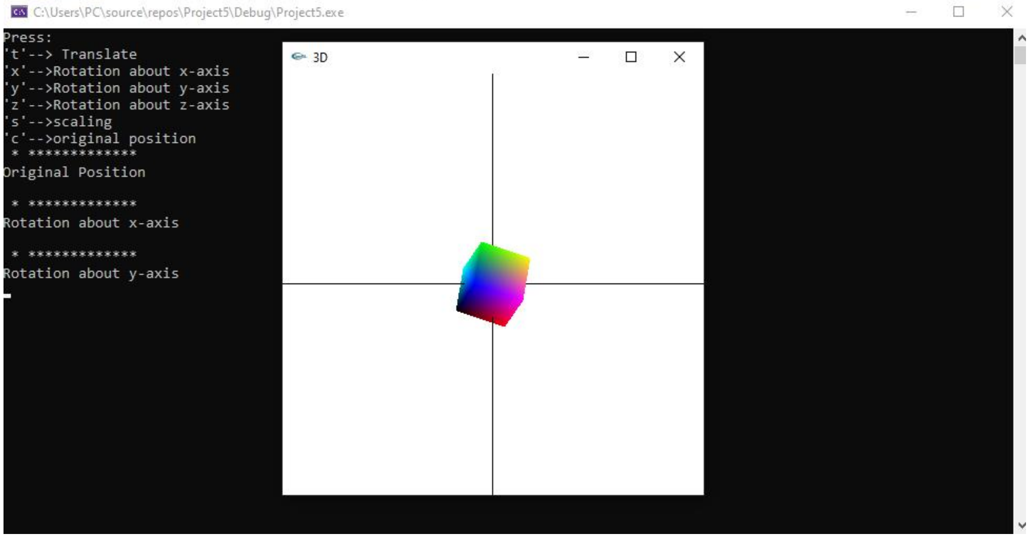
```
void myReshape(int w, int h)
{
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		glOrtho(-2, 2, -2 * (float)h / (float)w, 2 * (float)h / (float)w, -2, 2);
	else
		glOrtho(-2 * (float)w / (float)h, 2 * (float)w / (float)h, -2, 2, -2, 2);
	glMatrixMode(GL_MODELVIEW);
}

void main()
{
	printf("Press:\n't'--> Translate\n'x'-->Rotation about x-axis\n'y'-->Rotation about y-axis\n'z'-->Rotation about z-axis\n's'-->scaling\n'c'-->original position");
	glutInitWindowSize(500, 500);
	glutInitWindowPosition(300, 50);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutCreateWindow("3D");
	glutDisplayFunc(display);
	glutKeyboardFunc(keys);
	glEnable(GL_DEPTH_TEST);
	glEnableClientState(GL_COLOR_ARRAY);
	glEnableClientState(GL_VERTEX_ARRAY);
	glVertexPointer(3, GL_FLOAT, 0, vertices);
	glColorPointer(3, GL_FLOAT, 0, colors);
	glClearColor(1, 1, 1, 1);// White Background
	glColor3f(0, 0, 0);//Black writing Color
	glPointSize(2.0);//pointsize=2
	glutMainLoop();
}
```



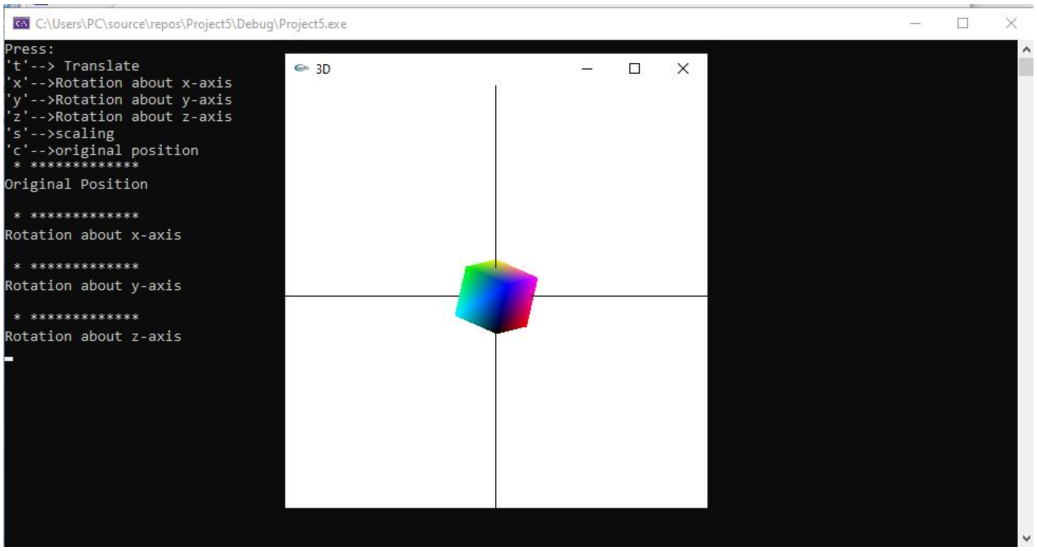
```
Press:
't'--> Translate
'x'-->Rotation about x-axis
'y'-->Rotation about y-axis
'z'-->Rotation about z-axis
's'-->scaling
'c'-->original position
 * *************
Original Position
```



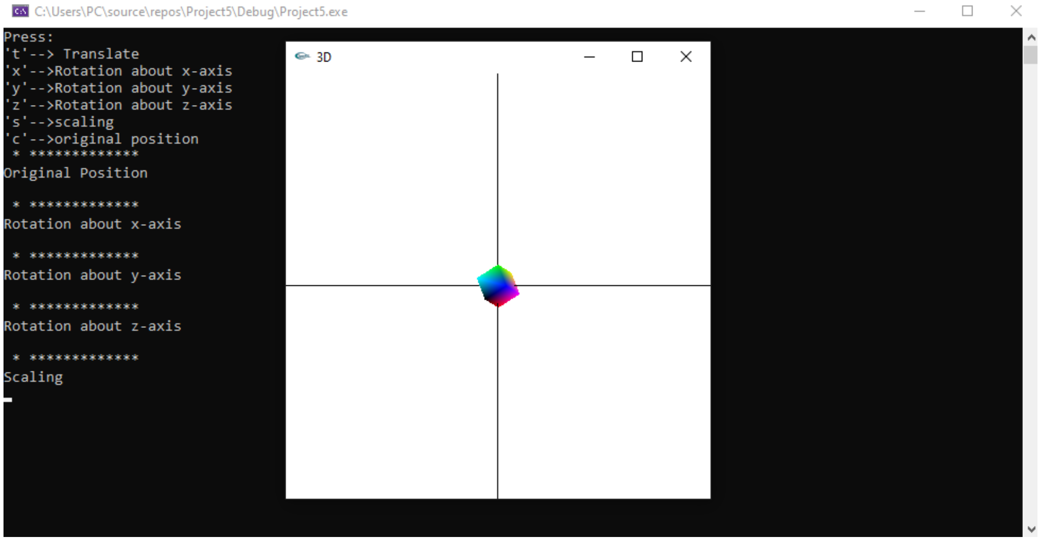
```
Press:
't'--> Translate
'x'-->Rotation about x-axis
'y'-->Rotation about y-axis
'z'-->Rotation about z-axis
's'-->scaling
'c'-->original position
 * *************
Original Position

 * *************
Rotation about x-axis
```



```
Press:
't'--> Translate
'x'-->Rotation about x-axis
'y'-->Rotation about y-axis
'z'-->Rotation about z-axis
's'-->scaling
'c'-->original position
 * *************
Original Position

 * *************
Rotation about x-axis

 * *************
Rotation about y-axis
```



```
Press:
't'--> Translate
'x'-->Rotation about x-axis
'y'-->Rotation about y-axis
'z'-->Rotation about z-axis
's'-->scaling
'c'-->original position
 * *************
Original Position

 * *************
Rotation about x-axis

 * *************
Rotation about y-axis

 * *************
Rotation about z-axis
```



```
Press:
't'--> Translate
'x'-->Rotation about x-axis
'y'-->Rotation about y-axis
'z'-->Rotation about z-axis
's'-->scaling
'c'-->original position
 * *************
Original Position

 * *************
Rotation about x-axis

 * *************
Rotation about y-axis

 * *************
Rotation about z-axis

 * *************
Scaling
```

6. Develop a program to demonstrate Animation effects on simple objects.

```
#include<glut.h>
#include<math.h>
float t, r = 0.5, x, y, t1 = 360, i = -0.5;
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1, 0, 0, 0);
        glColor3f(1, 1, 1);
        for (t = 0; t < 360; t +=0.1)
        {
                x = i + r * cos(t);
                y = r * sin(t);
                glBegin(GL_POINTS);
                glVertex2f(x, y);
                glEnd();
        }
        glBegin(GL_LINES);
        glVertex2f(r * cos(t1) + i, r * sin(t1));
        glVertex2f(r * -cos(t1) + i, r * -sin(t1));
        glVertex2f(r * -sin(t1) + i, r * cos(t1));
        glVertex2f(r * sin(t1) + i, r * -cos(t1));
        glEnd();
        glFlush();
}
void idle()
{
        if (i < 1)
                i = i + 0.001;
        else
                i = -0.5;
        t1 -= 0.01;
        display();
}
void mouse(int b, int s, int x, int y)
```

```
{
        if (b == GLUT_LEFT_BUTTON && s == GLUT_DOWN)
                glutIdleFunc(idle);
        if (b == GLUT_RIGHT_BUTTON && s == GLUT_DOWN)
                glutIdleFunc(NULL);
        if (b == GLUT_MIDDLE_BUTTON && s == GLUT_DOWN)
                exit(0);

}
void main()
{
        glutInitWindowSize(1200, 1200);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("Rotation of wheel");
        glutDisplayFunc(display);
        glutIdleFunc(idle);
        glutMouseFunc(mouse);
        glPointSize(2);
        glutMainLoop();
}
```

```
#Program-7 Write a Program to read a digital image. Split and display image into 4
#quadrants, up, down, right and left.
import cv2
from matplotlib import pyplot as plt
# Load image
image = cv2.imread('D:\\puppy.jpg')
# create figure
plt.figure(figsize=(7, 8))
# setting values to rows and column variables
rows = 3; columns = 2
#Get the height and width of the image
(h, w) = image.shape[:2]
#Converting BGR to RGB
image=image[:,:,::-1]
# get center of image and store in (cX,cY)
 (cX, cY) = (w // 2, h // 2)
# crop the image into four parts

topLeft = image[0:cY, 0:cX] # top left
topRight = image[0:cY, cX:w] #top right
bottomLeft = image[cY:h, 0:cX] # bottom left.
bottomRight = image[cY:h, cX:w] # bottom right.

# Adds a subplot at the 1st position and Display original image
plt.subplot(rows, columns, 1)
plt.imshow(image)
plt.axis('off')
plt.title("Original")
# Adds a subplot at the 2nd position and Display top left image
plt.subplot(rows, columns, 3)
plt.imshow(topLeft)
plt.axis('off')
plt.title("topLeft")
# Adds a subplot at the 3rd position and Display top right image
plt.subplot(rows, columns, 4)
plt.imshow(topRight)
plt.axis('off')
plt.title("topRight")
# Adds a subplot at the 4th position and Display bottom left image
plt.subplot(rows, columns, 5)
plt.imshow(bottomLeft)
plt.axis('off')
plt.title("bottomLeft")
# Adds a subplot at the 4th position and Display bottom Right image
plt.subplot(rows, columns, 6)
plt.imshow(bottomRight)
plt.axis('off')
plt.title("bottomRight")
```

```
# Program-8 program to show rotation, scaling, and translation on an image
#Python program to explain cv2.rotate() method, cv2.resize(),translate

import cv2
from matplotlib import pyplot as plt
import numpy as np
plt.figure(figsize=(20, 10))
rows = 2
columns = 2
# Reading an image and convert to rgb
src = cv2.imread('D:\\puppy.jpg')
src=src[:,:,::-1]
# Adds a subplot at the 1st position and display original image
plt.subplot(rows, columns, 1)
plt.imshow(src)
plt.axis('off')
plt.title("Original")

# Using cv2.rotate() method
# Using cv2.ROTATE_90_CLOCKWISE rotate  by 90 degrees clockwise
rot = cv2.rotate(src, cv2.ROTATE_90_CLOCKWISE)
# Adds a subplot at the 2nd position and display rotated image
plt.subplot(rows, columns, 2)
plt.imshow(rot)
plt.axis('off')
plt.title("Rotated")

#to resize create new dimension and use cv2.resize()
(h,w)=src.shape[:2]
newdim=(100,h)
img_shrinked = cv2.resize(src, newdim, interpolation=cv2.INTER_AREA)
# Adds a subplot at the 3rd position and display rotated image
plt.subplot(rows, columns, 3)
plt.imshow(img_shrinked)
plt.axis('off')
plt.title("scaled")

# shift the image (dx=25)25 pixels to the right and (dy=50)50 pixels down
M = np.float32([[1, 0, 25], [0, 1, 50]])
shifted = cv2.warpAffine(src, M, (w,h))
# Adds a subplot at the 4th position and display translated image
plt.subplot(rows, columns, 4)
plt.imshow(shifted)
plt.axis('off')
plt.title("Translated")
```

#9. Read an image and extract and display low-level features such as edges, textures using #filtering techniques.

```
import cv2
from matplotlib import pyplot as plt
import numpy as np

# Load the image

img = cv2.imread("D:\\puppy.jpg")
plt.figure(figsize=(7, 8))
plt.subplot(2, 2, 1)

# Display the original image
plt.imshow(img[:,:,::-1)
plt.axis('off')
plt.title("Original Image")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Edge detection
edges = cv2.Canny(gray, 100, 200)  # Use Canny edge detector

# Texture extraction
kernel = np.ones((5, 5), np.float32) / 25  # Define a 5x5 averaging kernel
texture = cv2.filter2D(gray, -1, kernel)  # Apply the averaging filter for texture extraction

# Display the edges, and texture

plt.subplot(2, 2, 2)
plt.imshow(edges)
plt.axis('off')
plt.title("Edges")

plt.subplot(2, 2,3)
plt.imshow(texture)
plt.axis('off')
plt.title("Texture")
```

```
#Program-10 Write a program to blur and smoothing an image.
#Smoothing an image using an average blur.
#Notice as how the kernel size increases, the image becomes progressively more
blurred.

import cv2
from matplotlib import pyplot as plt
import numpy as np

plt.figure(figsize=(7, 8))

# Reading an image and coverting to RGB
src = cv2.imread('D:\\puppy.jpg')
src=src[:,:,::-1]

#Display Original Image
plt.subplot(2, 2, 1)
plt.imshow(src)
plt.axis('off')
plt.title("Original")

i=2
kernelSizes = [(3, 3), (9, 9), (15, 15)]
# loop over the kernel sizes
for (kX, kY) in kernelSizes:
    # apply an "average" blur to the image using the current kernel size and display
    blurred = cv2.blur(src, (kX, kY))
    plt.subplot(2, 2, i)
    plt.imshow(blurred)
    plt.axis('off')
    plt.title("blurred ")
    i+=1
```

```
#Program-11 Write a program to contour an image.
# Grayscale
import cv2
from matplotlib import pyplot as plt
import numpy as np
plt.figure(figsize=(7, 8))

# Reading an image
src = cv2.imread('D:\\puppy.jpg')

gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

# Find Canny edges
edged = cv2.Canny(gray, 30, 200)

contours, hierarchy = cv2.findContours(edged,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

plt.subplot(1, 2, 1)

plt.imshow(edged)
plt.axis('off')
plt.title("Canny Edges After Contouring")

print("Number of Contours found = " + str(len(contours)))

# Draw all contours
# -1 signifies drawing all contours,(0,255,0) represents color,1 represents thickness
cv2.drawContours(src, contours, -1, (0, 255, 0), 1)


plt.subplot(1, 2, 2)
imgwithcontour=src[:,:,::-1]
# showing image
plt.imshow(imgwithcontour)
plt.axis('off')
plt.title("Contours")
```

#Program-12 Write a program to detect a face/s in an image

```
import cv2
from matplotlib import pyplot as plt
import numpy as np

# Reading an image
src = cv2.imread(D:\\a1.jpg')

gray_image = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

face_classifier = cv2.CascadeClassifier(
   cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)

face = face_classifier.detectMultiScale(
   gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(40, 40)
)

for (x, y, w, h) in face:
   cv2.rectangle(src, (x, y), (x + w, y + h), (0, 255, 0), 4)

img_rgb = src[:,:,::-1]

plt.figure(figsize=(7,8))
plt.imshow(img_rgb)
plt.axis('off')
```