Program 1

```
#include <GL/glut.h>
#include <stdio.h>   // For printf and scanf
#include <stdlib.h>  // For abs function

// Bresenham's line drawing algorithm
void drawLine(int x0, int y0, int x1, int y1) {
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = (x0 < x1) ? 1 : -1;
    int sy = (y0 < y1) ? 1 : -1;
    int err = dx - dy;

    while (1) {
        glBegin(GL_POINTS);
        glVertex2i(x0, y0);
        glEnd();

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y0 += sy;
        }
    }
}

// OpenGL display callback
void display() {
    int x1, x2, y1, y2;

    printf("Enter coordinates for x1 and y1: ");
    scanf("%d %d", &x1, &y1);
    printf("Enter coordinates for x2 and y2: ");
    scanf("%d %d", &x2, &y2);

    glClear(GL_COLOR_BUFFER_BIT);

    // Draw line using Bresenham's algorithm
    glColor3f(1.0f, 1.0f, 1.0f);
    drawLine(x1, y1, x2, y2);

    glFlush();
```

```
}

// OpenGL initialization
void initializeOpenGL(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Bresenham's Line Algorithm");

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 600);

    glutDisplayFunc(display);
}

// Main function
int main(int argc, char** argv) {
    initializeOpenGL(argc, argv);
    glutMainLoop();
    return 0;
}
```

Program 2

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

float squareX = 0.0f;
float squareY = 0.0f;
float squareSize = 0.2f;

void init()
{
   glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // Set the clipping area
}

void drawSquare()
{
   glColor3f(0.0, 0.0, 0.0); // Set square color to black
   glBegin(GL_QUADS);
   glVertex2f(squareX, squareY);
   glVertex2f(squareX + squareSize, squareY);
   glVertex2f(squareX + squareSize, squareY + squareSize);
   glVertex2f(squareX, squareY + squareSize);
   glEnd();
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT); // Clear the color buffer
   glLoadIdentity(); // Load the identity matrix

   drawSquare();

   glFlush(); // Flush OpenGL buffer
}

void reshape(int width, int height)
{
   glViewport(0, 0, width, height); // Set the viewport to cover the new window
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key)
   {
   case 'w':
      squareY += 0.05f; // Move square upwards
      break;
   case 's':
```

```
        squareY -= 0.05f; // Move square downwards
        break;
    case 'a':
        squareX -= 0.05f; // Move square to the left
        break;
    case 'd':
        squareX += 0.05f; // Move square to the right
        break;
    case 27:
        exit(0); // Exit program when 'Esc' key is pressed
        break;
    }
    glutPostRedisplay(); // Mark the current window for redisplay
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Set display mode
    glutInitWindowSize(500, 500); // Set window size
    glutInitWindowPosition(100, 100); // Set window position
    glutCreateWindow("vtucode| Basic Geometric Operations"); // Create the window with the
given title
    init(); // Initialize drawing
    glutDisplayFunc(display); // Register display callback function
    glutReshapeFunc(reshape); // Register reshape callback function
    glutKeyboardFunc(keyboard); // Register keyboard callback function
    glutMainLoop(); // Enter the main loop
    return 0;
}
```

Program 3

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

GLfloat angleX = 0.0f; // Angle for rotation around x-axis
GLfloat angleY = 0.0f; // Angle for rotation around y-axis
GLfloat scale = 1.0f;   // Scale factor

void init()
{
   glEnable(GL_DEPTH_TEST); // Enable depth testing for 3D rendering
   glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Set background color to white
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and depth
buffers
   glLoadIdentity(); // Load the identity matrix

   glTranslatef(0.0f, 0.0f, -5.0f); // Translate the cube to a distance of -5 units along z-axis
   glRotatef(angleX, 1.0f, 0.0f, 0.0f); // Rotate the cube around x-axis
   glRotatef(angleY, 0.0f, 1.0f, 0.0f); // Rotate the cube around y-axis
   glScalef(scale, scale, scale); // Scale the cube

   // Draw Cube
   glBegin(GL_QUADS); // Begin drawing the cube using quads
   // Front face
   glColor3f(1.0f, 0.0f, 0.0f); // Red color
   glVertex3f(-1.0f, -1.0f, 1.0f);
   glVertex3f(1.0f, -1.0f, 1.0f);
   glVertex3f(1.0f, 1.0f, 1.0f);
   glVertex3f(-1.0f, 1.0f, 1.0f);
   // Back face
   glColor3f(0.0f, 1.0f, 0.0f); // Green color
   glVertex3f(-1.0f, -1.0f, -1.0f);
   glVertex3f(-1.0f, 1.0f, -1.0f);
   glVertex3f(1.0f, 1.0f, -1.0f);
   glVertex3f(1.0f, -1.0f, -1.0f);
   // Top face
   glColor3f(0.0f, 0.0f, 1.0f); // Blue color
   glVertex3f(-1.0f, 1.0f, -1.0f);
   glVertex3f(-1.0f, 1.0f, 1.0f);
   glVertex3f(1.0f, 1.0f, 1.0f);
   glVertex3f(1.0f, 1.0f, -1.0f);
   // Bottom face
   glColor3f(1.0f, 1.0f, 0.0f); // Yellow color
   glVertex3f(-1.0f, -1.0f, -1.0f);
   glVertex3f(1.0f, -1.0f, -1.0f);
```

```
    glVertex3f(1.0f, -1.0f, 1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    // Right face
    glColor3f(1.0f, 0.0f, 1.0f); // Purple color
    glVertex3f(1.0f, -1.0f, -1.0f);
    glVertex3f(1.0f, 1.0f, -1.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, -1.0f, 1.0f);
    // Left face
    glColor3f(0.0f, 1.0f, 1.0f); // Cyan color
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glEnd(); // End of drawing cube

    glFlush(); // Flush OpenGL buffer
    glutSwapBuffers(); // Swap front and back buffers
}

void reshape(int width, int height)
{
    if (height == 0)
        height = 1; // To prevent divide by 0 error when height is 0
    glViewport(0, 0, width, height); // Set the viewport to cover the new window
    glMatrixMode(GL_PROJECTION); // Switch to projection matrix
    glLoadIdentity(); // Load the identity matrix
    gluPerspective(45.0f, (GLfloat)width / (GLfloat)height, 0.1f, 100.0f); // Set perspective
    glMatrixMode(GL_MODELVIEW); // Switch back to modelview matrix
    glLoadIdentity(); // Load the identity matrix
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'q': // 'q' key or ESC key
        exit(0); // Exit program
        break;
    case '+':
        scale += 0.1f; // Scale up
        break;
    case '-':
        scale -= 0.1f; // Scale down
        if (scale < 0.1f)
            scale = 0.1f; // Prevent scaling down too small
        break;
    }
```

```
    glutPostRedisplay(); // Mark the current window for redisplay
}

void specialKeys(int key, int x, int y)
{
    switch (key)
    {
    case GLUT_KEY_UP:
        angleX += 5.0f; // Rotate cube upward
        break;
    case GLUT_KEY_DOWN:
        angleX -= 5.0f; // Rotate cube downward
        break;
    case GLUT_KEY_LEFT:
        angleY -= 5.0f; // Rotate cube to the left
        break;
    case GLUT_KEY_RIGHT:
        angleY += 5.0f; // Rotate cube to the right
        break;
    }
    glutPostRedisplay(); // Mark the current window for redisplay
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH); // Set display mode
    glutInitWindowSize(800, 600); // Set window size
    glutInitWindowPosition(100, 100); // Set window position
    glutCreateWindow("vtucode | Basic 3D Geometric Operations"); // Create the window with
the given title
    init(); // Initialize drawing
    glutDisplayFunc(display); // Register display callback function
    glutReshapeFunc(reshape); // Register reshape callback function
    glutKeyboardFunc(keyboard); // Register keyboard callback function
    glutSpecialFunc(specialKeys); // Register special keys callback function
    glutMainLoop(); // Enter the main loop
    return 0;
}
```

Program 4

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

float angle = 0.0f;
float scaleX = 1.0f;
float scaleY = 1.0f;
float translateX = 0.0f;
float translateY = 0.0f;

void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();

   // Apply transformations
   glTranslatef(translateX, translateY, 0.0f); // Translation
   glRotatef(angle, 0.0f, 0.0f, 1.0f);         // Rotation
   glScalef(scaleX, scaleY, 1.0f);             // Scaling

   // Set color to green
   glColor3f(0.0f, 1.0f, 0.0f); // Green color

   // Draw your object here (e.g., a square)
   glBegin(GL_QUADS);
   glVertex2f(-0.5f, -0.5f);
   glVertex2f(0.5f, -0.5f);
   glVertex2f(0.5f, 0.5f);
   glVertex2f(-0.5f, 0.5f);
   glEnd();

   glFlush();
}


void reshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-1.0f, 1.0f, -1.0f, 1.0f);
   glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y)
{
   switch(key)
   {
   case 'r':
```

```
        angle += 5.0f;  // Rotate clockwise by 5 degrees
        break;
    case 'd':
        scaleX += 0.1f; // Scale up by 10%
        scaleY += 0.1f;
        break;
    case 's':
        scaleX -= 0.1f; // Scale down by 10%
        scaleY -= 0.1f;
        break;
    case 'f':
        translateX += 0.1f; // Translate 0.1 units to the right
        break;
    case 'a':
        translateX -= 0.1f; // Translate 0.1 units to the left
        break;
    case 'e':
        translateY += 0.1f; // Translate 0.1 units upwards
        break;
    case 'x':
        translateY -= 0.1f; // Translate 0.1 units downwards
        break;
    }
    glutPostRedisplay(); // Redraw the scene
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("vtucode | 2D Transformation");
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Program 5

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

GLfloat angle = 0.0;

void init()
{
   glClearColor(0.0, 0.0, 0.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(45.0, 1.0, 1.0, 100.0);
   glMatrixMode(GL_MODELVIEW);
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   glTranslatef(0.0, 0.0, -6.0);
   glRotatef(angle, 1.0, 1.0, 1.0);

   glBegin(GL_QUADS);
   // Front face
   glColor3f(1.0, 0.0, 0.0);
   glVertex3f(-1.0, -1.0, 1.0);
   glVertex3f(1.0, -1.0, 1.0);
   glVertex3f(1.0, 1.0, 1.0);
   glVertex3f(-1.0, 1.0, 1.0);
   // Back face
   glColor3f(0.0, 1.0, 0.0);
   glVertex3f(-1.0, -1.0, -1.0);
   glVertex3f(-1.0, 1.0, -1.0);
   glVertex3f(1.0, 1.0, -1.0);
   glVertex3f(1.0, -1.0, -1.0);
   // Top face
   glColor3f(0.0, 0.0, 1.0);
   glVertex3f(-1.0, 1.0, -1.0);
   glVertex3f(-1.0, 1.0, 1.0);
   glVertex3f(1.0, 1.0, 1.0);
   glVertex3f(1.0, 1.0, -1.0);
   // Bottom face
   glColor3f(1.0, 1.0, 0.0);
   glVertex3f(-1.0, -1.0, -1.0);
   glVertex3f(1.0, -1.0, -1.0);
   glVertex3f(1.0, -1.0, 1.0);
   glVertex3f(-1.0, -1.0, 1.0);
   // Right face
   glColor3f(1.0, 0.0, 1.0);
```

```
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);
    // Left face
    glColor3f(0.0, 1.0, 1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glEnd();

    glutSwapBuffers();
}

void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
}

void timer(int)
{
    angle += 2.0;
    glutPostRedisplay();
    glutTimerFunc(1000/60, timer, 0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("vtucode | 3D Transformations");

    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutTimerFunc(0, timer, 0);

    glEnable(GL_DEPTH_TEST);

    glutMainLoop();
    return 0;
}
```

Program 6

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

float trianglePosX = -0.5f; // Initial position of the triangle
float triangleSpeed = 0.005f; // Speed of the triangle

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    // Draw the triangle
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f);   // Red color
    glVertex2f(trianglePosX, 0.0f);
    glVertex2f(trianglePosX + 0.1f, 0.2f);
    glVertex2f(trianglePosX + 0.2f, 0.0f);
    glEnd();

    glutSwapBuffers();
}

void update(int value)
{
    // Update the position of the triangle
    trianglePosX += triangleSpeed;

    // If the triangle goes beyond the right edge of the window, reset its position
    if (trianglePosX > 1.1f)
        trianglePosX = -0.5f;

    // Redisplay the scene
    glutPostRedisplay();

    // Call update() again after 16 milliseconds
    glutTimerFunc(16, update, 0);
}

void init()
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // White background color
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600); // Window size
```

```
    glutCreateWindow("vtucode | OpenGL Animation");

    init();

    glutDisplayFunc(display);
    glutTimerFunc(16, update, 0); // Call update() after 16 milliseconds

    glutMainLoop();
    return 0;
}
```