Bresenhem’s line drawing algorithm:

```
#include <GL/glut.h>
#include <stdio.h>
GLint x0, y0, x1, y1;
void setPixel(int x, int y) {
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
glFlush();
}
void bresenhamLine(int x0, int y0, int x1, int y1) {
int dx = x1 - x0;
int dy = y1 - y0;
int D = 2 * dy - dx;
int y = y0;
for (int x = x0; x <= x1; x++) {
setPixel(x, y);
if (D > 0) {
y++;
D = D + (2 * (dy - dx));
} else {
D = D + 2*dy;
}
}
}
void display() {
glClear(GL_COLOR_BUFFER_BIT);
bresenhamLine(x0, y0, x1, y1);
glFlush();
}
// Initialize OpenGL settings
void init() {
glClearColor(1.0, 1.0, 1.0, 1.0);
glColor3f(0.0, 0.0, 0.0)
gluOrtho2D(0.0, 200.0, 0.0, 200.0);
}
int main(int argc, char** argv) {
printf("*********Bresenham's Line Drawing ");
printf("\nEnter starting vertex (x0, y0):");
scanf("%d%d",&x0, &y0);

printf("\nEnter ending vertex (x1, y1):");
scanf("%d%d",&x0, &y0);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(100, 100);
glutCreateWindow("Bresenham's Line Drawing");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Geometric operations on 2D:

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

GLfloat vertices[] = { -50.0f, 50.0f, 50.0f, 50.0f, 50.0f, -50.0f, -50.0f, -50.0f }; // Square coordinates
GLfloat scale = 1.0f; // Initial scale factor
GLfloat angle = 0.0f; // Initial rotation angle
GLfloat color[] = {1.0f, 0.0f, 0.0f}; // Initial color (red)

void drawSquare() {
  glEnableClientState(GL_VERTEX_ARRAY);
  glVertexPointer(2, GL_FLOAT, 0, vertices);
  glColor3fv(color); // Set object color before drawing
  glDrawArrays(GL_QUADS, 0, 4);
  glDisableClientState(GL_VERTEX_ARRAY);
}

void transformObject() {
  glPushMatrix(); // Push the current transformation matrix
  glTranslatef(0.0f, 0.0f, 0.0f); // Translate (modify these values for translation)
  glRotatef(angle, 0.0f, 0.0f, 1.0f); // Rotate (modify angle for rotation)
  glScalef(scale, scale, 1.0f); // Scale (modify scale for scaling)
  drawSquare();
  glPopMatrix(); // Pop the transformation matrix
}

void myInit() {
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(-100.0f, 100.0f, -100.0f, 100.0f);
  glMatrixMode(GL_MODELVIEW);
}

void myDisplay() {
  glClear(GL_COLOR_BUFFER_BIT);
  transformObject();
  glFlush();
}

void handleKeypress(unsigned char key, int x, int y) {
  switch (key) {
    case 'w': // Increase scale
      scale += 0.1f;
      break;
    case 's': // Decrease scale
      if (scale > 0.1f) {
        scale -= 0.1f;
      }
      break;
    case 'a': // Rotate left
      angle -= 5.0f;
      break;
    case 'd': // Rotate right
      angle += 5.0f;
      break;
    case 'r': // Change color to red
      color[0] = 1.0f;
      color[1] = 0.0f;
      color[2] = 0.0f;
      break;
    case 'g': // Change color to green
      color[0] = 0.0f;
      color[1] = 1.0f;
      color[2] = 0.0f;
      break;
    case 'b': // Change color to blue
      color[0] = 0.0f;
      color[1] = 0.0f;
    color[2] = 1.0f;
    break;
  }
  glutPostRedisplay();
}

int main(int argc, char* argv[]) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Specify RGB mode for color
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Basic Geometric Operations");
  glutDisplayFunc(myDisplay);
  glutKeyboardFunc(handleKeypress);
  myInit();
  glutMainLoop();
  return 0;
}
```

Geometric 3D transformation:

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

GLfloat vertices[8][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[8][3] = {
    {0.0, 1.0, 0.0}, {0.0, 1.0, 1.0}, {1.0, 1.0, 1.0}, {0.0, 1.0, 0.0},
    {0.0, 0.0, 1.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 0.0}, {0.0, 1.0, 1.0}
};

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

void colorcube() {
    polygon(0, 1, 2, 3);
    polygon(3, 2, 6, 7);
    polygon(4, 5, 6, 7);
    polygon(0, 1, 5, 4);
    polygon(1, 2, 6, 5);
    polygon(0, 3, 7, 4);
}

void display() {
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(5.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 6.0, 0.0);
colorcube();
glutSwapBuffers();
}
```

```
void myReshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Static Color Cube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

2D transformation:

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

// Define initial values for transformations
float tx = 0.0, ty = 0.0;  // Translation
float sx = 1.0, sy = 1.0;  // Scaling
float angle = 0.0;         // Rotation angle

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glPushMatrix();

    // Apply transformations
    glTranslatef(tx, ty, 0.0);
    glScalef(sx, sy, 1.0);
    glRotatef(angle, 0.0, 0.0, 1.0); // Rotation around Z-axis

    // Draw the square
    glBegin(GL_QUADS);
    glColor3f(1.0, 0.0, 0.0);   // Red color
    glVertex2f(-0.5, -0.5);
    glVertex2f(0.5, -0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(-0.5, 0.5);
    glEnd();

    glPopMatrix();

    glutSwapBuffers();
}

void menu(int option)
{
    switch (option)
    {
    case 1: // Translate Right
        tx += 0.1;
        break;
    case 2: // Translate Left
        tx -= 0.1;
        break;
    case 3: // Translate Up
        ty += 0.1;
        break;
    case 4: // Translate Down
        ty -= 0.1;
        break;
    case 5: // Scale Up
        sx *= 1.1;
        sy *= 1.1;
        break;
    case 6: // Scale Down
        sx *= 0.9;
        sy *= 0.9;
        break;
    case 7: // Rotate Clockwise
        angle -= 5.0;
        break;
    case 8: // Rotate Counter-clockwise
        angle += 5.0;
        break;
    case 9: // Reset Transformations
        tx = 0.0;
        ty = 0.0;
        sx = 1.0;
        sy = 1.0;
        angle = 0.0;
        break;
    case 10: // Exit
        exit(0);
        break;
    }

    glutPostRedisplay();
}

void createMenu()
{
    glutCreateMenu(menu);
    glutAddMenuEntry("Translate Right", 1);
    glutAddMenuEntry("Translate Left", 2);
    glutAddMenuEntry("Translate Up", 3);
    glutAddMenuEntry("Translate Down", 4);
    glutAddMenuEntry("Scale Up", 5);
    glutAddMenuEntry("Scale Down", 6);
    glutAddMenuEntry("Rotate Clockwise", 7);
    glutAddMenuEntry("Rotate Counter-clockwise", 8);
    glutAddMenuEntry("Reset Transformations", 9);
    glutAddMenuEntry("Exit", 10);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void init()
{
    glClearColor(1.0, 1.0, 1.0, 1.0); // White background
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // 2D orthogonal projection
    glMatrixMode(GL_MODELVIEW);
}
```

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("2D Transformations");
   init();
   createMenu();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```