EX-08 - 3D Transformations in C++ using OpenGL

02/10/2021

Venkataraman Nagarajan, CSE - C

18500192

AIM

To implement 3d-Transformations in C++.

SPECIFICATION

Perform the following basic 3D Transformations on any 3D Object.

1. Translation
2. Rotation
3. Scaling

Use only homogeneous coordinate representation and matrix multiplication to perform transformations.

Set the camera to any position on the 3D space. Have $(0, 0, 0)$ at the center of the screen. Draw X , Y and Z axis.

PROGRAM - 01

3D ransformations

```
1  // Perform the following basic 3D Transformations on any 3D Object.
2  //      1) Translation
3  //      2) Rotation
4  //      3) Scaling
5
6  // Use only homogeneous coordinate representation and matrix ↩
       multiplication to
7  // perform transformations.
8
9  // Set the camera to any position on the 3D space. Have (0,0,0) at the ↩
       center of the
10 // screen. Draw X , Y and Z axis.
11
12 #include <GL/glut.h>
13 #include <bits/stdc++.h>
14 #include <algorithm>
15
16 using namespace std;
17 using ld = long double;
18 using ll = long long;
19
20 const int WINDOW_WIDTH = 900;
21 const int WINDOW_HEIGHT = 900;
22
23 const int X_MIN = -600;
24 const int X_MAX = 600;
25 const int Y_MIN = -600;
26 const int Y_MAX = 600;
27 const int Z_MIN = -600;
28 const int Z_MAX = 600;
29
30 const ld PADDING = 0;
31 const ld STEP = 10;
32 const ld SCALE = 1;
33 const ld PI = 3.14159265358979323846264338327950288419716939937510582;
34
35 struct Point {
36     ld X, Y, Z;
37     Point(ld x, ld y, ld z) : X(x), Y(y), Z(z) {}
38     Point() : X(0), Y(0), Z(0) {}
39 };
40
41 struct Face {
```

```
42      vector<Point> points;
43      Face(vector<Point> &points) : points(points) {}
44  };
45
46  struct Object {
47      vector<Face> faces;
48      Object(vector<Face> &faces) : faces(faces) {}
49  };
50
51  enum RotationType {X_AXIS, Y_AXIS, Z_AXIS, INVALID};
52
53  void init();
54  void disp();
55  void display();
56
57  void printAxes();
58  ld getRadian(ld degree);
59  ld multiply(vector<ld> a, vector<ll> b);
60  vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b);
61  vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b);
62
63  Point getPoint(vector<ld> point_matrix);
64  vector<ll> getHomogeneousPointCoords(Point point, ll h=1);
65  vector<vector<ld>> getTransformMatrix();
66
67  vector<vector<ld>> translate(ld tx=0, ld ty=0, ld tz=0);
68  vector<vector<ld>> scale(ld sx=1, ld sy=1, ld sz=1, Point pivot=Point↩
        (0,0,0));
69  vector<vector<ld>> rotate(ld angle=0.0, RotationType type=Z_AXIS);
70
71  void transformShape();
72
73  int main(int argc, char *argv[]) {
74      glutInit(&argc, argv);
75      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
76      glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
77      glutCreateWindow("3D - Transformations");
78      init();
79      glutDisplayFunc(display);
80      glutMainLoop();
81
82      return 0;
83  }
84
85  void init() {
86      glClearColor(1.0, 1.0, 1.0, 1.0);
87      glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
88      glEnable( GL_BLEND );
89      glLoadIdentity();
90      glOrtho(X_MIN, X_MAX, Y_MIN, Y_MAX, Z_MIN, Z_MAX);
91      glEnable(GL_DEPTH_TEST);
92  }
93
94  //Transformations - Built-in functions- only for reference
95  void disp() {
96      glRotatef(50, 1, 0, 0);
97      glRotatef(50, 0, 1, 0);
98      glRotatef(50, 0, 0, 1);
99  }
100
101 bool twoEqual(Point a, Point b) {
102     return (a.X == b.X && a.Y == b.Y ) || (a.X == b.X && a.Z == b.Z) || (a↩
            .Y == b.Y && a.Z == b.Z);
103 }
104
105 Face orderSquare(Face face, string commonPlane="X") {
106     vector<Point> points = face.points;
107     vector<Point> newface;
108     if(points.size() != 4) return face;
109
110     Point current = points[0];
111     newface.push_back(current);
112     ll cou = 0;
113
114     points.erase(points.begin());
115
116     while(cou < 4) {
117         for(auto point: points) {
118             if(twoEqual(point, current)) {
119                 current = point;
120                 newface.push_back(current);
121                 points.erase(
122                     remove_if(points.begin(), points.end(), [&](Point ↩
                            const & point) {
123                         return point.X == current.X && point.Y == current.↩
                                Y && point.Z == current.Z;
124                     }), points.end());
125                 break;
126             }
127         }
128         cou++;
129     }
130
131     return Face(newface);
```

```
}
// Returns a cube
Object getObject() {
    vector<Point> front, back, left, right, top, bottom;
    vector<Face> faces;

    for(ll x=-100; x<=100; x+=200) {
        for(ll y=-100; y<=100; y+=200) {
            for(ll z=-100; z<=100; z+=200) {
                if(z > 0) {
                    front.push_back(Point(x, y, z));
                } else {
                    back.push_back(Point(x, y, z));
                }

                if(x > 0) {
                    right.push_back(Point(x, y, z));
                } else {
                    left.push_back(Point(x, y, z));
                }

                if(y > 0) {
                    top.push_back(Point(x, y, z));
                } else {
                    bottom.push_back(Point(x, y, z));
                }
            }
        }
    }

    faces.push_back(orderSquare(Face(front), "Z"));
    faces.push_back(orderSquare(Face(back), "Z"));
    faces.push_back(orderSquare(Face(left), "X"));
    faces.push_back(orderSquare(Face(right), "X"));
    faces.push_back(orderSquare(Face(top), "Y"));
    faces.push_back(orderSquare(Face(bottom), "Y"));

    return Object(faces);
}

vector<vector<ld>> getTransformMatrix() {
    ll no_of_transformations;
    string type;

    vector<vector<ld>> transform_matrix = ↩
       {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};

```

```
178     cout << "\n\t\t Composite Transformations \n\n";
179     cout << "Input Format: \n";
180     cout << "\t'T' tx ty tz (Translation by (tx,ty,tz))\n";
181     cout << "\t'Ro' angle type['X'/'Y'/'Z'] (Rotation by angle about x-↩
            axis,y-axis,z-axis)\n";
182     cout << "\t'Sc' sx sy sz fx fy fz (Scale by sx & sy with ref to (fx,fy↩
            ,fz))\n";
183
184     cout << "\nSample Input: \n";
185     cout << "\t3\n";
186     cout << "\tT 4.5 10 20\n";
187     cout << "\tRo 45 X\n";
188     cout << "\tSc 3 2 0.5 0 0 0\n";
189
190     cout << "\nYour Input: \n";
191     cin >> no_of_transformations;
192
193     for(ll i=0;i<no_of_transformations;i++) {
194         cin>>type;
195         if(type=="T") {
196             ld tx,ty,tz;
197             cin>>tx>>ty>>tz;
198
199             vector<vector<ld>> translation_matrix = translate(tx,ty,tz);
200
201             transform_matrix = multiply(translation_matrix, ↩
                    transform_matrix);
202         } else if(type=="Ro") {
203             ld angle;
204             string type;
205             cin>>angle>>type;
206
207             vector<vector<ld>> rotation_matrix;
208
209             if(type=="X")       rotation_matrix = rotate(angle, X_AXIS);
210             else if(type=="Y")  rotation_matrix = rotate(angle, Y_AXIS);
211             else if(type=="Z")  rotation_matrix = rotate(angle, Z_AXIS);
212             else                rotation_matrix = rotate(angle, INVALID);
213
214             transform_matrix = multiply(rotation_matrix, transform_matrix)↩
                    ;
215         } else if(type=="Sc") {
216             ld sx,sy,sz,fx,fy,fz;
217             cin>>sx>>sy>>sz>>fx>>fy>>fz;
218
219             vector<vector<ld>> scaling_matrix = scale(sx,sy,sz,Point(fx,fy↩
                    ,fz));
```

```
            transform_matrix = multiply(scaling_matrix, transform_matrix);
        }
    }

    cout << "\nOutput is displayed on the window\n\n";

    return transform_matrix;
}

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

    Object obj = getObject();
    disp();
    printAxes();

    // Print Original Object

    ll cou = 0;
    for(Face face : obj.faces) {

        if(cou < 6) glColor3f(0.0, 0.0, 1.0);
        if(cou < 4) glColor3f(0.0, 1.0, 0.0);
        if(cou < 2) glColor3f(1.0, 0.0, 0.0);

        glBegin(GL_QUADS);
        for(Point point : face.points) {
            glVertex3f(point.X, point.Y, point.Z);
        }
        glEnd();
        cou++;
    }

    vector<vector<ld>> transform_matrix = getTransformMatrix();

    // Print Transformed Object

    cou = 0;
    for(Face face : obj.faces) {

        if(cou < 6) glColor4f(0.0, 0.0, 1.0, 0.5);
        if(cou < 4) glColor4f(0.0, 1.0, 0.0, 0.5);
        if(cou < 2) glColor4f(1.0, 0.0, 0.0, 0.5);

        glBegin(GL_QUADS);
        for(Point point : face.points) {
            Point transformed_point = getPoint(multiply(transform_matrix, ↩
```

```
                getHomogeneousPointCoords(point)));
267             glVertex3f(transformed_point.X, transformed_point.Y, ↩
                transformed_point.Z);
268         }
269         glEnd();
270         cou++;
271     }
272
273     glFlush();
274 }
275
276 void printAxes() {
277     glBegin(GL_LINES);
278
279     glColor3f(1.0f,0.0f,0.0f);
280     glVertex3d(2*X_MIN,0,0);
281     glVertex3d(2*X_MAX,0,0);
282
283     glColor3f(0.0f,1.0f,0.0f);
284     glVertex3d(0,2*Y_MIN,0);
285     glVertex3d(0,2*Y_MAX,0);
286
287     glColor3f(0.0f,0.0f,1.0f);
288     glVertex3d(0,0,2*Z_MIN);
289     glVertex3d(0,0,2*Z_MAX);
290
291     glEnd();
292 }
293
294 Point getPoint(vector<ld> point_matrix) {
295     ll h = point_matrix[3];
296     ld x = point_matrix[0];
297     ld y = point_matrix[1];
298     ld z = point_matrix[2];
299
300     return {x/h,y/h,z/h};
301 }
302
303 vector<ll> getHomogeneousPointCoords(Point point, ll h) {
304     vector<ll> point_matrix;
305     point_matrix.push_back(h*point.X);
306     point_matrix.push_back(h*point.Y);
307     point_matrix.push_back(h*point.Z);
308     point_matrix.push_back(h);
309     return point_matrix;
310 }
311
```

```
ld getRadian(ld degree) {
    return degree*PI/180;
}

vector<vector<ld>> translate(ld tx, ld ty, ld tz) {
    vector<vector<ld>> translate_matrix = {
                                            {1,0,0,tx},
                                            {0,1,0,ty},
                                            {0,0,1,tz},
                                            {0,0,0,1}
                                        };
    return translate_matrix;
}

vector<vector<ld>> scale(ld sx, ld sy, ld sz, Point pivot) {

    ll xf = pivot.X;
    ll yf = pivot.Y;
    ll zf = pivot.Z;

    vector<vector<ld>> scale_matrix = {
                                {sx, 0, 0, xf*(1-sx)},
                                {0, sy, 0, yf*(1-sy)},
                                {0,  0, sz, zf*(1-sz)},
                                {0,  0,  0, 1}
                            };
    return scale_matrix;
}

vector<vector<ld>> rotate(ld angle, RotationType type) {
    angle = getRadian(angle);

    vector<vector<ld>> rotate_matrix;

    switch(type) {
        case X_AXIS:
            rotate_matrix = {
                                {1, 0, 0, 0},
                                {0, cos(angle), -sin(angle), 0},
                                {0, sin(angle), cos(angle), 0},
                                {0, 0, 0, 1}
                            };
            break;

        case Y_AXIS:
            rotate_matrix = {
                                {cos(angle), 0, sin(angle), 0},
```

```
                                {0, 1, 0, 0},
                                {-sin(angle), 0, cos(angle), 0},
                                {0, 0, 0, 1}
                            };
            break;

        case Z_AXIS:
            rotate_matrix = {
                                {cos(angle), -sin(angle), 0, 0},
                                {sin(angle), cos(angle), 0, 0},
                                {0, 0, 1, 0},
                                {0, 0, 0, 1}
                            };
            break;
        default:
            rotate_matrix = {
                                {1,0,0,0},
                                {0,1,0,0},
                                {0,0,1,0},
                                {0,0,0,1}
                            };
            break;

    }

    return rotate_matrix;
}


vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b) ↩
    {
    vector<vector<ld>> result;
    for(int i=0; i<a.size(); i++) {
        vector<ld> row;
        for(int j=0; j<b[0].size(); j++) {
            ld sum = 0;
            for(int k=0; k<a[0].size(); k++) {
                sum += a[i][k]*b[k][j];
            }
            row.push_back(sum);
        }
        result.push_back(row);
    }
    return result;
}

vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b) {
```

```
405     vector<ld> result;
406     for(int i=0;i<a.size();i++) {
407         ll temp = multiply(a[i],b);
408         result.push_back(temp);
409     }
410     return result;
411 }
412
413 ld multiply(vector<ld> a, vector<ll> b) {
414     ld result=0;
415     for(int i=0;i<a.size();i++) {
416         result+=(a[i]*b[i]);
417     }
418     return result;
419 }
```

SAMPLE I/0

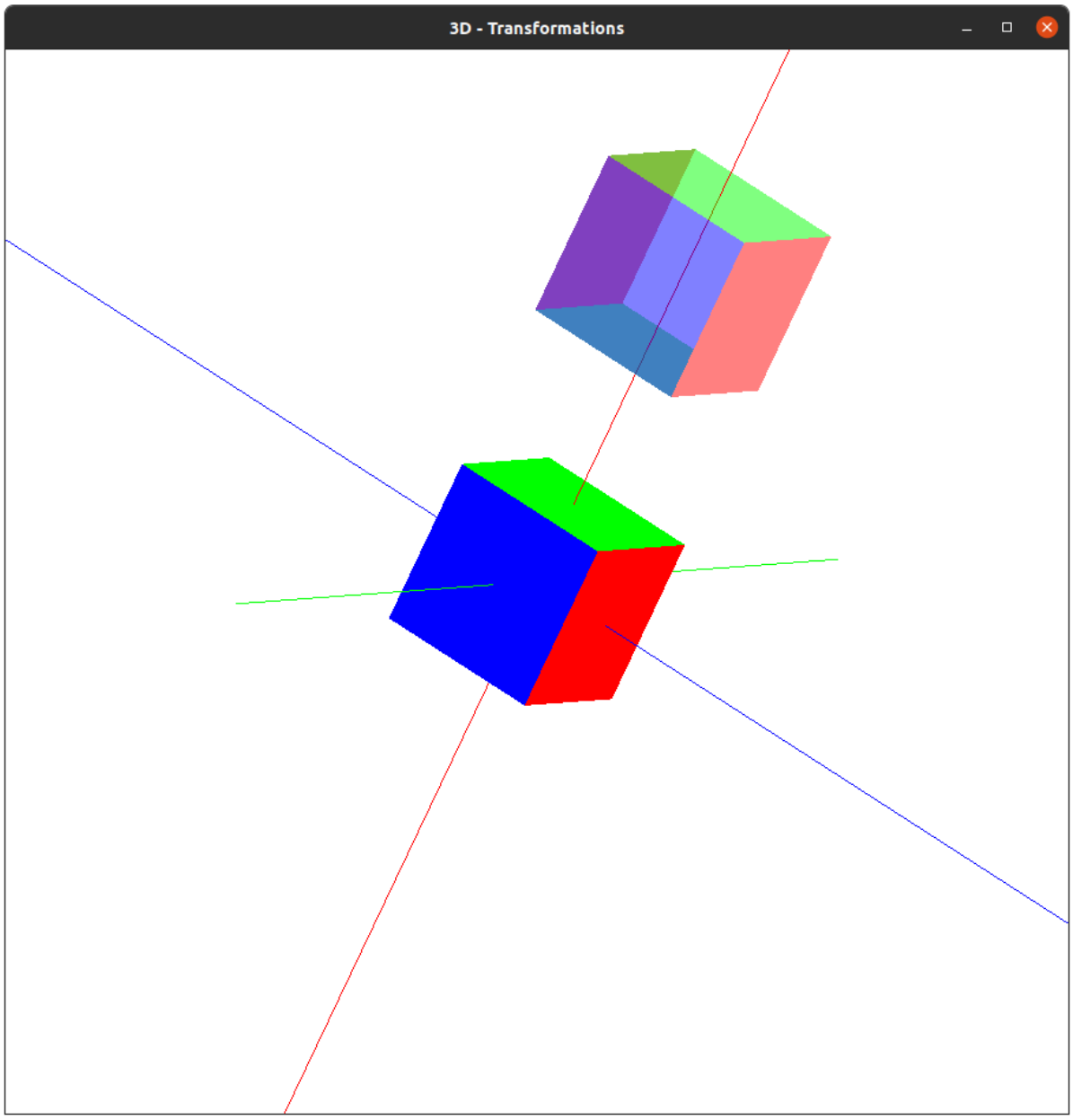

Figure 1: Translating solid cube to translucent cube

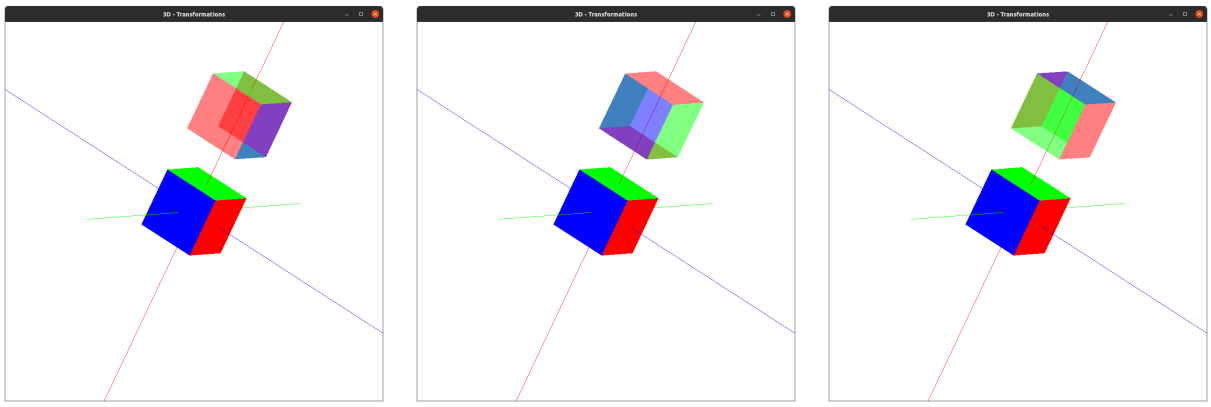

Figure 2: Rotating solid cube by $90°$ with respect to x, y, z axes respectively

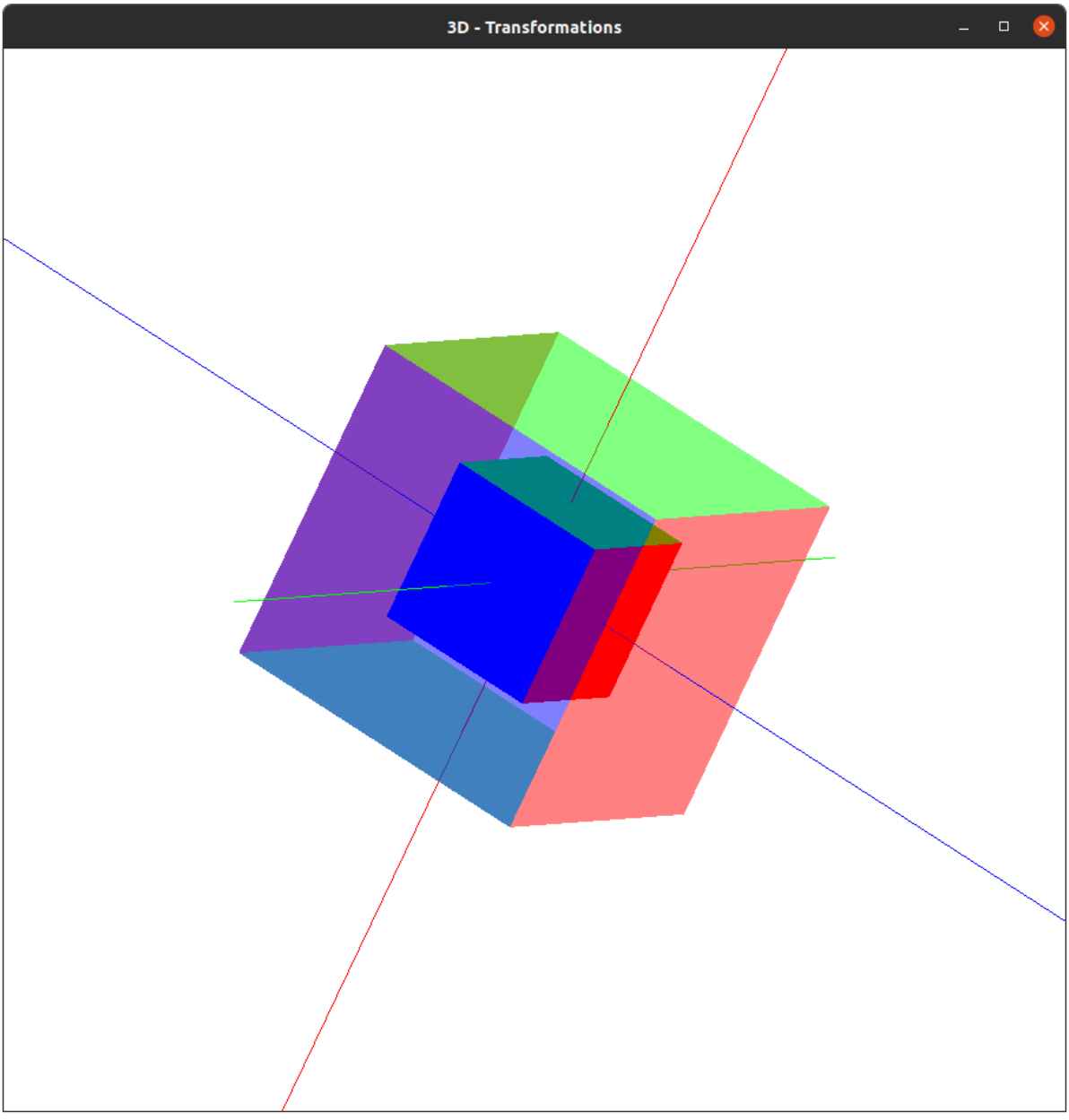

Figure 3: Scaling solid cube to translucent cube w.r.t origin

RESULT

The code to implement 3d transformations are written and output is verified.
