```
#include <iostream>
#include <iomanip>

using namespace std;

#include <Eigen/Core>
#include <Eigen/Geometry>

using namespace Eigen;

#include <pangolin/pangolin.h>

struct RotationMatrix {
    Matrix3d matrix = Matrix3d::Identity();
};

ostream &operator<<(ostream &out, const RotationMatrix &r) {
    out.setf(ios::fixed);
    Matrix3d matrix = r.matrix;
    out << '=';
    out << "[" << setprecision(2) << matrix(0, 0) << "," << matrix(0, 1) << "," << matrix(0, 2) << "],"
        << "[" << matrix(1, 0) << "," << matrix(1, 1) << "," << matrix(1, 2) << "],"
        << "[" << matrix(2, 0) << "," << matrix(2, 1) << "," << matrix(2, 2) << "]";
    return out;
}

istream &operator>>(istream &in, RotationMatrix &r) {
    return in;
}

struct TranslationVector {
    Vector3d trans = Vector3d(0, 0, 0);
};

ostream &operator<<(ostream &out, const TranslationVector &t) {
    out << "=[" << t.trans(0) << ',' << t.trans(1) << ',' << t.trans(2) << "]";
    return out;
}

istream &operator>>(istream &in, TranslationVector &t) {
    return in;
```

```
}

struct QuaternionDraw {
   Quaterniond q;
};

ostream &operator<<(ostream &out, const QuaternionDraw quat) {
   auto c = quat.q.coeffs();
   out << "=[" << c[0] << "," << c[1] << "," << c[2] << "," << c[3] << "]";
   return out;
}

istream &operator>>(istream &in, const QuaternionDraw quat) {
   return in;
}

inline void glDrawColouredCube2(GLfloat axis_min=-0.5f, GLfloat axis_max = +0.5f)
{
      const GLfloat l = axis_min;
      const GLfloat h = axis_max;

      const GLfloat verts[] = {
            l,l,h,   h,l,h,   l,h,h,   h,h,h,   // FRONT
            l,l,l,   l,h,l,   h,l,l,   h,h,l,   // BACK
            l,l,h,   l,h,h,   l,l,l,   l,h,l,   // LEFT
            h,l,l,   h,h,l,   h,l,h,   h,h,h,   // RIGHT
            l,h,h,   h,h,h,   l,h,l,   h,h,l,   // TOP
            l,l,h,   l,l,l,   h,l,h,   h,l,l    // BOTTOM
      };

      glVertexPointer(3, GL_FLOAT, 0, verts);
      glEnableClientState(GL_VERTEX_ARRAY);

      glColor4f(1.0f, 0.5f, 0.0f, 0.2);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
      glDrawArrays(GL_TRIANGLE_STRIP, 4, 4);

      glColor4f(0.0f, 0.5f, 0.0f, 1.0f);
      glDrawArrays(GL_TRIANGLE_STRIP, 8, 4);
      glDrawArrays(GL_TRIANGLE_STRIP, 12, 4);
```

```
        glColor4f(0.0f, 0.0f, 1.0f, 1.0f);
        glDrawArrays(GL_TRIANGLE_STRIP, 16, 4);
        glDrawArrays(GL_TRIANGLE_STRIP, 20, 4);

        glDisableClientState(GL_VERTEX_ARRAY);
}

int main(int argc, char **argv) {
    pangolin::CreateWindowAndBind("visualize geometry", 1000, 600);
    glEnable(GL_DEPTH_TEST);
    pangolin::OpenGlRenderState s_cam(
        pangolin::ProjectionMatrix(1000, 600, 420, 420, 500, 300, 0.1, 1000),
        pangolin::ModelViewLookAt(3, 3, 3, 0, 0, 0, pangolin::AxisY)
    );

    const int UI_WIDTH = 500;

    pangolin::View &d_cam = pangolin::CreateDisplay().
        SetBounds(0.0, 1.0, pangolin::Attach::Pix(UI_WIDTH), 1.0, -1000.0f / 600.0f).
        SetHandler(new pangolin::Handler3D(s_cam));

    // ui
    pangolin::Var<RotationMatrix> rotation_matrix("ui.R", RotationMatrix());
    pangolin::Var<TranslationVector> translation_vector("ui.t", TranslationVector());
    pangolin::Var<TranslationVector> euler_angles("ui.rpy", TranslationVector());
    pangolin::Var<QuaternionDraw> quaternion("ui.q", QuaternionDraw());
    pangolin::CreatePanel("ui").SetBounds(0.0, 1.0, 0.0, pangolin::Attach::Pix(UI_WIDTH));

    while (!pangolin::ShouldQuit()) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        d_cam.Activate(s_cam);

        pangolin::OpenGlMatrix matrix = s_cam.GetModelViewMatrix();
        Matrix<double, 4, 4> m = matrix;

        RotationMatrix R;
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                R.matrix(i, j) = m(j, i);
        rotation_matrix = R;
```

```
        TranslationVector t;
        t.trans = Vector3d(m(0, 3), m(1, 3), m(2, 3));
        t.trans = -R.matrix * t.trans;
        translation_vector = t;

        TranslationVector euler;
        euler.trans = R.matrix.eulerAngles(2, 1, 0);
        euler_angles = euler;

        QuaternionDraw quat;
        quat.q = Quaterniond(R.matrix);
        quaternion = quat;

        glColor3f(0.2, 0.2, 1.0);

//      pangolin::glDrawColouredCube();
        glDrawColouredCube2();
        // draw the original axis
        glLineWidth(3);
        glColor3f(1.0f, 0.f, 0.f);
        glBegin(GL_LINES);
        glVertex3f(0, 0, 0);
        glVertex3f(10, 0, 0);
        glColor3f(0.f, 1.0f, 0.f);
        glVertex3f(0, 0, 0);
        glVertex3f(0, 10, 0);
        glColor3f(0.0f, 0.0f, 1.f);
        glVertex3f(0, 0, 0);
        glVertex3f(0, 0, 10);
        glEnd();

        pangolin::FinishFrame();
    }
}
```