CPSC 484 – Fundamentals of Computer Graphics:
Implement a working QUATERNION class in C++.

The second programming assignment is to complete the following C++, templated versions of the quaternion class, printed on this pdf, and also included as a source file on Slack #general and on Titanium.

Your code should entirely be implemented in the quaternion_T.h header file, which includes an extensive set of tests in its static void run_tests() method.

I have updated the main.cpp file from the previous assignment to include the quaternion tests. In particular, the run_tests method of quaternion includes many tests of an airplane rotating in space about its x, y, and z axes from the following website: https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/examples/index.htm.

Why do we care about Quaternions? Couldn't we just use Euler angles to do this? We could, but they are susceptible to losing one degree of freedom if there are small computational errors (meaning we can only rotate about two directions then), and they are not as simple, powerful, and fast as quaternions.

To help with this project, a version of this code is also provided in Python. The Python code is similar, but not identical to the C++ code, of course, due to the many differences between the languages. But, I believe you will find it helpful.

Your code, however, must be implemented in C++, must compile and run correctly, and must pass all of the assertions in the provided C++ header file.

You can submit your code in a single header file, and include a file showing the output of your program running to Titanium. You can also submit the code to your github portfolio, if you wish, and make sure that the link you submit to Titanium is live, and that your portfolio is public.

This project is due by 2359 on Sunday, 28 February 2021. See the syllabus for precise information on late submission penalties, which are significant.

Good luck.

```
// =====================================================================================================
// quaternion_T.h
// =====================================================================================================

#ifndef quaternion_T_h
#define quaternion_T_h

#include <cmath>
#include "vector3d_T.h"
#include "matrix3d_T.h"

template <typename T> class quaternion;
template <typename T> using quat = class quaternion<T>;
typedef quat<double> quatD;

template <typename T>
class quaternion {
public:
  quaternion(T w_=T(), T x_=T(), T y_=T(), T z_=T())
  : w(w_), x(x_), y(y_), z(z_) { }

  static quaternion i();
  static quaternion j();
  static quaternion k();
```

```
  static double ii();
  static double jj();
  static double kk();
  static double ijk();

  static quaternion ij();
  static quaternion jk();
  static quaternion ki();

  static quaternion ji();
  static quaternion kj();
  static quaternion ik();

  friend quaternion operator+(const quaternion& a, const quaternion& b);
  friend quaternion operator-(const quaternion& a, const quaternion& b);
  friend quaternion operator*(const quaternion& a, const quaternion& b);

  friend quaternion operator+(const quaternion& q, T k);
  friend quaternion operator+(T k, const quaternion& q);


  friend quaternion operator-(const quaternion& q, T k);
  friend quaternion operator-(T k, const quaternion& q);

  friend quaternion operator*(const quaternion& q, T k);
  friend quaternion operator*(T k, const quaternion& q);
  friend quaternion operator/(const quaternion& q, T k);


  quaternion operator-() const;

  friend bool operator==(const quaternion& q, const quaternion& r);
  friend bool operator!=(const quaternion& q, const quaternion& r);
  vector3d<T> vector() const;
  T scalar() const;

  quaternion unit_scalar() const;

  quaternion conjugate() const;

  quaternion inverse() const;

  quaternion unit() const;

  double norm() const;
  double magnitude();

  double dot(const quaternion& v) const;

  double angle(const quaternion& v) const;

  matrix3d<T> rot_matrix() const;

  // rotates point pt (pt.x, pt.y, pt.z) about (axis.x, axis.y, axis.z) by theta
  static vec3 rotate(const vector3D& pt, const vector3D& axis, double theta);

  friend std::ostream& operator<<(std::ostream& os, const quaternion& q) {
    os << "Quat(";
    if (q ==  quaternion::i())  { return os <<  "i)"; }
    if (q == -quaternion::i())  { return os << "-i)"; }
    if (q ==  quaternion::j())  { return os <<  "j)"; }
    if (q == -quaternion::j())  { return os << "-j)"; }
    if (q ==  quaternion::k())  { return os <<  "k)"; }
    if (q == -quaternion::k())  { return os << "-k)"; }

    if (q.magnitude() == 0.0 && q.w == 0)   { return os << "0)"; }
    if (q.magnitude() == 0.0 && q.w == 0)   { return os << "0)"; }
    if (q.magnitude() == 1.0 && q.w == 1)   { return os << "1)"; }
    if (q.vector().magnitude() == 0.0)      { return os << q.w << ")"; }
    else { return os << q.w << q.vector() << ")"; }
  }

  static void run_tests();

private:
  T w, x, y, z;
};

void plane_rotation(const std::string& msg, const quatD& plane, const std::initializer_list<double>& li) {
  matrix3dD rotate = matrix3dD("rot_matrix", 3, li);
  assert(plane.rot_matrix() == rotate);
```

```
  std::cout << msg << " is: " << plane << plane.rot_matrix() << "\n";
}


template <typename T>
void quaternion<T>::run_tests() {
  quatD a(1, 2, 3, 4), b(4, 0, 0, 7), c(0, 1, 1, 0), d(0, 0, 1, 0);
  quatD e(0, 0, 0, 1), f(0, 0, 0, 0), g(1, 0, 0, 0), h(3, 0, 0, 0);
  std::cout << "a = " << a << ")\nb = " << b << ")\nc = " << c << ")\nd = " << d
            << ")\ne = " << e << ")\nf = " << f << ")\ng = " << g << ")\nh = " <<  h << "\n";

  std::cout << "c + d = " <<  c + d << "\nc + d + e = " << c + d + e;
  std::cout << "5 * h = " << 5 * h << "\nh * 5 = " << h * 5 << "\nh / 3.0 = " << h / 3.0 << "\n\n";

  std::cout << "h.magnitude() is " << h.magnitude() << "\nh.unit() is " << h.unit();
  std::cout << "g.unit() is " << g.unit() << "\na.unit() is " << a.unit() << ")\n\n";

  std::cout << "a.vector() is " << a.vector() << "\na.scalar() is " << a.scalar() << "\n";
  std::cout << "a.conjugate() is " << a.conjugate() << "\na.inverse() is " << a.inverse()
            << "\na * a.inverse() is " << a * a.inverse() << "\n\n";

  std::cout << "c == d is " << (c == d) << "\nc != d is " << (c != d);
  std::cout << "\ne == e is " << (e == e) << "\ne != e is " << (e != e) << "\n";

  std::cout << "\n\nquat.ij is: " << quatD::ij() << "\nquat.jk is: " << quatD::jk()
            << "\nquat.ki is: " << quatD::ki() << "\n";
  assert(quatD::ij() == quatD::k());
  assert(quatD::jk() == quatD::i());
  assert(quatD::ki() == quatD::j());

  std::cout << "\nquat.ji is: " << quatD::ji() << "\nquat.kj is: " << quatD::kj()
            << "\nquat.ik is: " << quatD::ik() << "\nquat.ijk is: " << quatD::ijk() << "\n";
  assert(quatD::ji() == -quatD::k());
  assert(quatD::kj() == -quatD::i());
  assert(quatD::ik() == -quatD::j());

  std::cout << "\nquat.ii is: " << quatD::ii() << "\nquat.jj is: " << quatD::jj()
            << "\nquat.kk is: " << quatD::kk() << "\n";
  assert(quatD::ii()  == -1);
  assert(quatD::jj()  == -1);
  assert(quatD::kk()  == -1);
  assert(quatD::ijk() == -1);

  std::cout << "\nangle (deg) between c and d is: " << c.angle(d) << "\n";
  quatD c_minus_d = c - d;
  std::cout << "c_minus_d is: " << c_minus_d;
  matrix3dD rot_matrix = c_minus_d.rot_matrix();
  std::cout << "rot_matrix of c_minus_d is: " << c_minus_d.rot_matrix() << "\n";

  double rad2_2 = sqrt(2)/2.0;
  std::cout << "// -------------- LEVEL FLIGHT -------------------')\n";
  plane_rotation("levelFlight(E)", quatD(1),                          {  1,  0,  0,   0,  1,  0,   0,  0,  1 });
  plane_rotation("levelFlight(N)", quatD(rad2_2, 0, rad2_2,  0), {  0,  0,  1,   0,  1,  0,  -1,  0,  0 });
  plane_rotation("levelFlight(W)", quatD(0,      0,  1,      0), { -1,  0,  0,   0,  1,  0,   0,  0, -1 });
  plane_rotation("levelFlight(S)", quatD(rad2_2, 0, -rad2_2, 0), {  0,  0, -1,   0,  1,  0,   1,  0,  0} );
  std::cout << "LEVEL FLIGHT assertions passed .............................................\n";
  std::cout << "// --------- end LEVEL FLIGHT ------------------------)\n";

  std::cout << "// -------------- STRAIGHT UP -------------------')\n";
  plane_rotation("straightUp(E)", quatD(rad2_2, 0, 0, rad2_2),   {  0, -1,  0,   1,  0,  0,   0,  0,  1 } );
  plane_rotation("straightUp(N)", quatD(0.5, 0.5, 0.5, 0.5),     {  0,  0,  1,   1,  0,  0,   0,  1,  0 } );
  plane_rotation("straightUp(W)", quatD(0, rad2_2, rad2_2, 0),   {  0,  1,  0,   1,  0,  0,   0,  0, -1 } );
  plane_rotation("straightUp(S)", quatD(0.5, -0.5, -0.5, 0.5),   {  0,  0, -1,   1,  0,  0,   0, -1,  0 } );
  std::cout << "STRAIGHT UP assertions passed.............................................\n";
  std::cout << "// --------- end STRAIGHT UP ------------------------)\n\n";


  std::cout << "// -------------- STRAIGHT DOWN ------------------')\n";
  plane_rotation("straightDown(E)", quatD(rad2_2, 0, 0, -rad2_2), {  0,  1,  0,  -1,  0,  0,   0,  0,  1 } );
  plane_rotation("straightDown(E)", quatD(0.5, -0.5, 0.5, -0.5),  {  0,  0,  1,  -1,  0,  0,   0, -1,  0 });
  plane_rotation("straightDown(E)", quatD(0, -rad2_2, rad2_2, 0), {  0, -1,  0,  -1,  0,  0,   0,  0, -1 } );
  plane_rotation("straightDown(E)", quatD(0.5, 0.5, -0.5, -0.5),  {  0,  0, -1,  -1,  0,  0,   0,  1,  0 });
  std::cout << "STRAIGHT DOWN assertions passed.............................................\n";
  std::cout << "// --------- end STRAIGHT DOWN ----------------------)\n\n";


  std::cout << "\n\n -------- BANK/ROLL ----------------\n";
  std::cout << "\nBanking/Rolling 90 degrees left...\n";
  plane_rotation("plane_E_bankLeft90", quatD(rad2_2, rad2_2, 0, 0),  {  1,  0,  0,   0,  0, -1,   0,  1,  0 } );
  plane_rotation("plane_N_bankLeft90", quatD(0.5, 0.5, 0.5, -0.5),   {  0,  1,  0,   0,  0, -1,  -1,  0,  0 } );
  plane_rotation("plane_W_bankLeft90", quatD(0, 0, rad2_2, -rad2_2), { -1,  0,  0,   0,  0, -1,   0, -1,  0 }  );
```

```
  plane_rotation("plane_W_bankLeft90", quatD(0.5, 0.5, -0.5, 0.5),   {  0, -1,  0,   0,  0, -1,   1,  0,  0 } );
  std::cout << "ROLL 90 deg left assertions passed..............................................\n";

  std::cout << "\n\nBanking/Rolling 180 degrees...\n";
  plane_rotation("plane_E_bankLeft180", quatD(0, 1, 0, 0),            {  1,  0,  0,   0, -1,  0,   0,  0, -1 });
  plane_rotation("plane_N_bankLeft180", quatD(0, rad2_2, 0, -rad2_2), {  0,  0, -1,   0, -1,  0,  -1,  0,  0 });
  plane_rotation("plane_W_bankLeft180", quatD(0, 0, 0, 1),            { -1,  0,  0,   0, -1,  0,   0,  0,  1 });
  plane_rotation("plane_S_bankLeft180", quatD(0, rad2_2, 0, rad2_2),  {  0,  0,  1,   0, -1,  0,   1,  0,  0 });
  std::cout << "ROLL 180 degrees assertions passed..............................................\n";

  std::cout << "\n\nBanking/Rolling 90 degrees right...\n";
  plane_rotation("plane_E_bankRight90", quatD(rad2_2, -rad2_2, 0, 0), {  1,  0,  0,   0,  0,  1,   0, -1,  0 });
  plane_rotation("plane_N_bankRight90", quatD(0.5, -0.5, 0.5, 0.5),   {  0, -1,  0,   0,  0,  1,  -1,  0,  0 });
  plane_rotation("plane_W_bankRight90", quatD(0, 0, rad2_2, rad2_2),  { -1,  0,  0,   0,  0,  1,   0,  1,  0 });
  plane_rotation("plane_S_bankRight90", quatD(0.5, -0.5, -0.5, -0.5), {  0,  1,  0,   0,  0,  1,   1,  0,  0 });
  std::cout << "ROLL 90 deg right assertions passed.............................................\n";
  std::cout << "\n -------- end BANK/ROLL ----------------\n";

  std::cout << "\nALL PLANE ROTATION ASSERTIONS PASSED ............................................\n\n";

  std::cout << "SEE THIS WEBSITE for DETAILED DIAGRAMS on the TESTS of the PLANE's rotations\n";
  std::cout << "https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/examples/
index.htm\n";
}


#endif /* quaternion_T_h */
```

```
# ==========================================================================================================
# main.cpp
# ==========================================================================================================
//  main.cpp
//  vectors
//
//  Created by William McCarthy on 27/Jan/21.
//


//=============================================================
// file: main.cpp
//=============================================================
#include <iostream>
#include <cstring>
#include <initializer_list>
#include <cassert>

//MATRIX and VECTOR classes assignment
#include "vector3d_T.h"
#include "matrix3d_T.h"
#include "quaternion_T.h"


template <typename T>
void print(T v) { std::cout << v << "\n"; }

template <typename T>
void show_vect(T v) { std::cout << v.name() << " is: " << v << "\n"; }

template <typename T>
void show_mat(T m) { std::cout << m.name() << " is: " << m << "\n"; }

void test_vectors() {
  print("\n====================  TESTING VECTORS  ========================");
  vector3d<double> u("u", 3, {1,  2,  4});
  std::cout << u.name() << "\n";
  std::cout << u << "\n";
  u.zero();
  show_vect(u);
  vector3D v("v", 3, {8, 16, 32});
  vector3D i("i", 3, {1, 0, 0}), j("j", 3, {0, 1, 0}), k("k", 3, {0, 0, 1});
  vector3D w(3 * i + 4 * j - 2 * k);

   show_vect(u);
  show_vect(v);
  show_vect(i);
```

```
  show_vect(j);
  show_vect(k);
  j + k;
  show_vect(w);

  assert(u == u);
  assert(u != v);
  assert(u + v == v + u);
  assert(u - v == -(v - u));
  assert(-(-u) == u);
  assert(3.0 + u == u + 3.0);
  assert(3.0 * u == u * 3.0);
  assert((u - 3.0) == -(3.0 - u));
  assert((5.0 * u) / 5.0 == u);

  assert(u + vector3D::zero() == u);

  assert(i.dot(j) == j.dot(k) == k.dot(i) == 0);
  assert(i.cross(j) == k);
  assert(j.cross(k) == i);
  assert(k.cross(i) == j);
  assert(u.cross(v) == -v.cross(u));
  assert(u.cross(v + w) == u.cross(v) + u.cross(w));
  assert((u.cross(v)).dot(u) == 0);

  print(i.angle(j));
  print(M_PI/2);

  assert(i.angle(j) == M_PI_2);
  assert(j.angle(k) == M_PI_2);
  assert(k.angle(i) == M_PI_2);
  vector3D uhat = u / u.magnitude();
  show_vect(u);
  show_vect(uhat);
  print(uhat.magnitude());
  assert(uhat.magnitude() - 1.0 < 1.0e-10);

  print("...test vectors assertions passed");
  print("====================  FINISHED testing vectors  ========================");
}

void test_matrices() {
  print("\n====================  TESTING MATRICES  ========================");
  matrix3dD a("a", 3, {3, 2, 0,   0, 0, 1,   2, -2, 1});
  matrix3dD b("b", 3, {1, 0, 5,   2, 1, 6,   3,  4, 0});
  matrix3dD id = matrix3dD::identity(3);
  assert(a * id == a);
  assert(a * b != -b * a);
  assert((a * b).transpose() == b.transpose() * a.transpose());

  matrix3dD acopy(a);    // copy constructor
  matrix3dD a2copy = a;  // copy constructor

  matrix3dD bcopy;
  bcopy = b;        // assignment operator

  matrix3dD ainv = a.inverse();
  matrix3dD binv = b.inverse();

  show_mat(a);
  show_mat(b);
  show_mat(-a);
  show_mat(-b);
  show_mat(a * b);
  printf("|a| = %.2f\n", a.determinant());
  printf("|b| = %.2f\n", b.determinant());
  show_mat(a.transpose());
  show_mat(b.transpose());

  show_mat(a.minors());
  show_mat(b.minors());

  show_mat(a.cofactor());
  show_mat(b.cofactor());

  show_mat(a.adjugate());
  show_mat(b.adjugate());


  show_mat(ainv);
  show_mat(binv);
```

```
  show_mat(a * ainv);
  show_mat(b * binv);
  show_mat(matrix3dD::identity(3));

  assert(a * ainv == matrix3dD::identity(3));
  assert(a * ainv == ainv * a);
  assert(b * binv == matrix3dD::identity(3));
  assert(b * binv == binv * b);
  assert(a.transpose().transpose() == a);
  assert(a.determinant() == a.transpose().determinant());

  assert(a + b == b + a);
  assert(a - b == -(b - a));
  assert(3.0 + a == a + 3.0);
  assert(3.0 * a == a * 3.0);
  assert((a + 3.0) - 3.0 == a);
  assert((3.0 * a) / 3.0 == a);
  assert(-(-a) == a);

  matrix3dD zerod("zerod", 3, {1, 2, 3,   4, 5, 6,   7, 8, 9});
  assert(zerod.determinant() == 0);

  print("...test matrices assertions passed");
  print("====================  FINISHED testing matrices  ========================");
}

void test_matrices_and_vectors() {
  print("\n====================  TESTING MATRICES and VECTORS  ========================");
  vector3D p("p", 2, {1, 2});
  matrix3dD m("m", 2, {1, 2,   3, 4});
  show_vect(p);
  show_mat(m);
  assert(p * m == m * p);

  vector3D q("q", 3, {1, 2, 3});
  matrix3dD n("n", 3, {1, 2, 3,   4, 5, 6,   7, 8, 9});
  show_vect(q);
  show_mat(n);
  assert(q * n == n * q);
  print("...test_matrices_and_vectors assertions passed");
  print("====================  FINISHED testing matrices and vectors  ========================");
}

void test_quaternions() {
  print("\n====================  TESTING QUATERNIONS  ========================");
  quaternion<double>::run_tests();
  print("...test_matrices_and_vectors assertions passed");
  print("====================  FINISHED testing quaternions  ========================");

}

int main(int argc, const char * argv[]) {
//  test_vectors();
//  test_matrices();
//  test_matrices_and_vectors();
  test_quaternions();
  print("... program completed...\n");
  return 0;
}
```

```
# ==========================================================================================================
# quaternion.py
# ==========================================================================================================

from math import sqrt, acos
from math import pi
from math import cos, sin, atan2
PI_2 = pi / 2.0
from vector import Vector
from matrix import Matrix


class Quaternion:
    def __init__(self, w, x=0, y=0, z=0):
        self.w = float(w)
```

```
        self.x = float(x)
        self.y = float(y)
        self.z = float(z)

    @classmethod
    def i(cls): return Quaternion(0.0, 1.0, 0.0, 0.0)
    @classmethod
    def j(cls): return Quaternion(0.0, 0.0, 1.0, 0.0)
    @classmethod
    def k(cls): return Quaternion(0.0, 0.0, 0.0, 1.0)

    @classmethod
    def ii(cls): return -1
    @classmethod
    def jj(cls): return -1
    @classmethod
    def kk(cls): return -1

    @classmethod
    def ij(cls): return Quaternion.k()
    @classmethod
    def ji(cls): return -Quaternion.k()

    @classmethod
    def jk(cls): return Quaternion.i()
    @classmethod
    def kj(cls): return -Quaternion.i()

    @classmethod
    def ki(cls): return Quaternion.j()
    @classmethod
    def ik(cls): return -Quaternion.j()

    @classmethod
    def ijk(cls): return -1

    def __str__(self):
        s = 'Quat('
        if self == Quaternion.i(): return s + 'i)'
        if self == -Quaternion.i(): return s + '-i)'
        if self == Quaternion.j(): return s + 'j)'
        if self == -Quaternion.j(): return s + '-j)'
        if self == Quaternion.k(): return s + 'k)'
        if self == -Quaternion.k(): return s + '-k)'
        if self.magnitude() == 0.0 and self.w == 0: return s + '0)'
        if self.magnitude() == 1.0 and self.w == 1: return s + '1)'
        if self.vector().magnitude() == 0.0: return f'{s}{self.w})'
        else: return s + f'{self.w:.1f} + {self.vector()})'

    def __add__(self, o):
        if isinstance(o, float):
            return Quaternion(o, self.x + o, self.y + o, self.z)
        return Quaternion(self.w + o.w, self.x + o.x, self.y + o.y, self.z + o.z)

    def __radd__(self, o): return self + o

    def __sub__(self, v): return self + -v

    def __rsub__(self, o): return -(o - self)

    def __rmul__(self, o): return self * o

    def __mul__(self, o):
        if isinstance(o, Quaternion):
            w1, x1, y1, z1 = self.w, self.x, self.y, self.z
            w2, x2, y2, z2 = o.w, o.x, o.y, o.z
            return Quaternion((w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2),
                              (w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2),
                              (w1 * y2 + y1 * w2 + z1 * x2 - x1 * z2),
                              (w1 * z2 + z1 * w2 + x1 * y2 - y1 * x2))
        val = o
        return Quaternion(val * self.w, val * self.x, val * self.y, val * self.z)

    def  __truediv__(self, val): return self * (1.0 / val)

    def __neg__(self): return Quaternion(-self.w, -self.x, -self.y, -self.z)

    def __eq__(self, v): return self.w == v.w and self.x == v.x and self.y == v.y and self.z == v.z

    def __ne__(self, v): return not (self == v)
```

```
def vector(self): return Vector(self.x, self.y, self.z)

def scalar(self): return self.w

def unit_scalar(self): return Quaternion(1.0, Vector())

def conjugate(self): return Quaternion(self.w, -self.x, -self.y, -self.z)

def inverse(self): return self.conjugate() / self.magnitude() ** 2

def unit(self): return self / self.magnitude()

def norm(self): return sqrt(self.w ** 2 + self.x ** 2 + self.y ** 2 + self.z ** 2)

def magnitude(self): return self.norm()

def dot(self, v): self.w * v.w + self.vector().dot(v.vector())

def angle(self, v):
    if not isinstance(v, Quaternion): raise TypeError
    z = self.conjugate() * v
    zvnorm = z.vector().norm()
    zscalar = z.scalar()
    angle = atan2(zvnorm, zscalar)
    return angle * 180.0 / 3.1415

def rot_matrix(self):
    w, x, y, z = self.w, self.x, self.y, self.z
    # print(w, x, y, z)
    return Matrix(3, 3, -2*(y**2 + z**2) + 1,  2*(x*y  - w*z),       2*(x*z  +  w*y),
                        2*(x*y  + w*z),      -2*(x**2 + z**2) + 1,  2*(y*z  -  w*x),
                        2*(x*z  - w*y),       2*(y*z  + w*x),      -2*(x**2 + y**2)+1)
    # return Matrix(3, 3, 2*(w**2 + x**2) - 1,  2*(x*y  - w*z),       2*(x*z  +  w*y),
    #                      2*(x*y  + w*z),       2*(w**2 + y**2) - 1,  2*(y*z  -  w*x),
    #                      2*(x*z  - w*y),       2*(y*z  + w*x),       2*(w**2 + z**2)-1)

@staticmethod
def rotate(pt, axis, theta):     # rotates a point pt (pt.x, pt.y, pt.z) about (axis.x, axis.y, axis.z) by theta
    costheta2 = cos(theta / 2.0)
    sintheta2 = sin(theta / 2.0)
    q = Quaternion(costheta2, axis.x * sintheta2, axis.y * sintheta2, axis.z * sintheta2)
    q_star = Quaternion(q.w, -q.x, -q.y, -q.z)
    p = Quaternion(0, pt.x, pt.y, pt.z)
    p_rot = q * p * q_star
    return Vector(p_rot.x, p_rot.y, p_rot.z)

@staticmethod
def run_tests():
    a = Quaternion(1, 2, 3, 4)
    b = Quaternion(4, 0, 0, 7)

    c = Quaternion(0, 1, 1, 0)
    d = Quaternion(0, 0, 1, 0)

    e = Quaternion(0, 0, 0, 1)
    f = Quaternion(0, 0, 0, 0)
    g = Quaternion(1, 0, 0, 0)
    h = Quaternion(3, 0, 0, 0)

    print('a = ' + str(a))
    print('b = ' + str(b))
    print('c = ' + str(c))
    print('d = ' + str(d))
    print('e = ' + str(e))
    print('f = ' + str(f))
    print('g = ' + str(g))
    print('h = ' + str(h), '\n')

    print('c + d = ', str(c + d))
    print('c + d + e = ', c + d + e, '\n')

    print(f'5 * h is: {5.0 * h}')
    print(f'h * 5 is: {h * 5.0}')
    print(f'h / 3.0 is: {h / 3.0}')
    print(f'h.magnitude() is: {h.magnitude()}')
    print(f'h.unit() is: {h.unit()}')
    print(f'g.unit() is: {g.unit()}')
    print(f'a.unit() is: {a.unit()}\n')

    print(f'a.vector() is: {a.vector()}')
    print(f'a.scalar() is: {a.scalar()}')
```

```
        print(f'a.conjugate() is: {a.conjugate()}')
        print(f'a.inverse() is: {a.inverse()}')
        print(f'a * a.inverse() is: {a * a.inverse()}\n')

        print(f'c == d is: {c == d}')
        print(f'c != d is: {c != d}')

        print(f'e == e is: {e == e}')
        print(f'e != e is: {e != e}\n')

        print(f'Quaternion.ij is: {Quaternion.ij()}')
        print(f'Quaternion.jk is: {Quaternion.jk()}')
        print(f'Quaternion.ki is: {Quaternion.ki()}\n')

        print(f'Quaternion.ji is: {Quaternion.ji()}')
        print(f'Quaternion.kj is: {Quaternion.kj()}')
        print(f'Quaternion.ik is: {Quaternion.ik()}\n')

        print(f'Quaternion.ijk is: {Quaternion.ijk()}')

        print(f'Quaternion.ii is: {Quaternion.ii()}')
        print(f'Quaternion.jj is: {Quaternion.jj()}')
        print(f'Quaternion.kk is: {Quaternion.kk()}\n')


        print(f'angle between c and d is: {c.angle(d):.3f} degrees')
        c_minus_d = c - d
        print(f'c_minus_d is: {c_minus_d}')
        rot_matrix = c_minus_d.rot_matrix()
        print(f'rot_matrix of c_minus_d is: {rot_matrix}')

        rad2_2 = sqrt(2)/2.0

        print("SEE THIS WEBSITE for DETAILED DIAGRAMS on the TESTS of the PLANE's rotations")
        print('https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/examples/
index.htm')

        print('# -------------- LEVEL FLIGHT -------------------')
        plane = Quaternion(1)
        print(f'levelflight(E) is {plane}{plane.rot_matrix()}')

        plane = Quaternion(rad2_2, 0, rad2_2, 0)
        print(f'levelflight(N) is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, 0, 1, 0)
        print(f'levelflight(W) is {plane}{plane.rot_matrix()}')

        plane = Quaternion(rad2_2, 0, -rad2_2, 0)
        print(f'levelflight(S) is {plane}{plane.rot_matrix()}')
        print('# --------------------------------------------------')


        print('\n\n# -------- STRAIGHT UP ---------------------')
        plane = Quaternion(rad2_2, 0, 0, rad2_2)
        print(f'plane_straightupE is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, 0.5, 0.5, 0.5)
        print(f'plane_straightupN is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, rad2_2, rad2_2, 0)
        print(f'plane_straightupW is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, -0.5, -0.5, 0.5)
        print(f'plane_straightupS is {plane}{plane.rot_matrix()}')
        print('# -------- end STRAIGHT UP ---------------------')


        print('\n\n# -------- STRAIGHT DOWN ---------------------')
        plane = Quaternion(rad2_2, 0, 0, -rad2_2)
        print(f'plane_straightdownE is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, -0.5, 0.5, -0.5)
        print(f'plane_straightdownN is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, -rad2_2, rad2_2, 0)
        print(f'plane_straightdownW is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, 0.5, -0.5, -0.5)
        print(f'plane_straightdownS is {plane}{plane.rot_matrix()}')
        print('# -------- end STRAIGHT UP ---------------------')
```

```
        print('\n\n# --------  BANK/ROLL ---------------------')
        plane = Quaternion(rad2_2, rad2_2, 0, 0)
        print(f'plane_E_bankLeft90 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, 0.5, 0.5, -0.5)
        print(f'plane_N_bankLeft90 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, 0, rad2_2, -rad2_2)
        print(f'plane_W_bankLeft90 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, 0.5, -0.5, 0.5)
        print(f'plane_S_bankLeft90 is {plane}{plane.rot_matrix()}')

        print('\nBanking/Rolling 180 degrees')
        plane = Quaternion(0, 1, 0, 0)
        print(f'plane_E_bankLeft180 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, rad2_2, 0, -rad2_2)
        print(f'plane_N_bankLeft180 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, 0, 0, 1)
        print(f'plane_W_bankLeft180 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, rad2_2, 0, rad2_2)
        print(f'plane_S_bankLeft180 is {plane}{plane.rot_matrix()}')


        print('\nBanking/Rolling Right 90 degrees')
        plane = Quaternion(rad2_2, -rad2_2, 0, 0)
        print(f'plane_E_bankRight180 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, -0.5, 0.5, 0.5)
        print(f'plane_N_bankRight180 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0, 0, rad2_2, rad2_2)
        print(f'plane_W_bankRight80 is {plane}{plane.rot_matrix()}')

        plane = Quaternion(0.5, -0.5, -0.5, -0.5)
        print(f'plane_S_bankRight80 is {plane}{plane.rot_matrix()}')
        print('# -------- end BANK/ROLL ---------------------')

        print("SEE THIS WEBSITE for DETAILED DIAGRAMS on the TESTS of the PLANE's rotations")
        print('https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/examples/
index.htm')


def main():
    # Vector.run_tests()
    Quaternion.run_tests()
    # Matrix.run_tests()


if __name__ == '__main__':
    main()
```

```
//====================  TESTING QUATERNIONS  ========================  EXPECTED OUTPUT ========================
a = Quat(1<quat,   2   3   4   0>))
b = Quat(4<quat,   0   0   7   0>))
c = Quat(0<quat,   1   1   0   0>))
d = Quat(j))
e = Quat(k))
f = Quat(0))
g = Quat(1))
h = Quat(3)
c + d = Quat(0<quat,   1   2   0   0>)
c + d + e = Quat(0<quat,   1   2   1   0>)5 * h = Quat(15)
h * 5 = Quat(15)
h / 3.0 = Quat(1)

h.magnitude() is 3
h.unit() is Quat(1)g.unit() is Quat(1)
a.unit() is Quat(0.183<quat, 0.365 0.548 0.73   0>))

a.vector() is <quat,   2   3   4   0>
```

```
a.scalar() is 1
a.conjugate() is Quat(1<quat,  -2  -3  -4   0>)
a.inverse() is Quat(0.183<quat, -0.365 -0.548 -0.73   0>)
a * a.inverse() is Quat(5.48)

c == d is 0
c != d is 1
e == e is 1
e != e is 0


quat.ij is: Quat(k)
quat.jk is: Quat(i)
quat.ki is: Quat(j)

quat.ji is: Quat(-k)
quat.kj is: Quat(-i)
quat.ik is: Quat(-j)
quat.ijk is: -1

quat.ii is: -1
quat.jj is: -1
quat.kk is: -1

angle (deg) between c and d is: 45
c_minus_d is: Quat(i)rot_matrix of c_minus_d is: <'rot_matrix', <col0,   1   0   0   0><col1,   0  -1   0   0><col2,
0   0  -1   0>> OR by rows...
  1   0   0
  0  -1   0
  0   0  -1
>
// -------------- LEVEL FLIGHT -------------------')
levelFlight(E) is: Quat(1)<'rot_matrix', <col0,   1   0   0   0><col1,   0   1   0   0><col2,   0   0   1   0>> OR by
rows...
  1   0   0
  0   1   0
  0   0   1
>
levelFlight(N) is: Quat(0.707<quat,   0 0.707   0   0>)<'rot_matrix', <col0,   0   0  -1   0><col1,   0   1   0
0><col2,   1   0   0   0>> OR by rows...
  0   0   1
  0   1   0
 -1   0   0
>
levelFlight(W) is: Quat(j)<'rot_matrix', <col0,  -1   0   0   0><col1,   0   1   0   0><col2,   0   0  -1   0>> OR by
rows...
 -1   0   0
  0   1   0
  0   0  -1
>
levelFlight(S) is: Quat(0.707<quat,   0 -0.707   0   0>)<'rot_matrix', <col0,   0   0   1   0><col1,   0   1   0
0><col2,  -1   0   0   0>> OR by rows...
  0   0  -1
  0   1   0
  1   0   0
>
LEVEL FLIGHT assertions passed .............................................
// --------- end LEVEL FLIGHT ------------------------)
// -------------- STRAIGHT UP -------------------')
straightUp(E) is: Quat(0.707<quat,   0   0 0.707   0>)<'rot_matrix', <col0,   0   1   0   0><col1,  -1   0   0
0><col2,   0   0   1   0>> OR by rows...
  0  -1   0
  1   0   0
  0   0   1
>
straightUp(N) is: Quat(0.5<quat, 0.5 0.5 0.5   0>)<'rot_matrix', <col0,   0   1   0   0><col1,   0   0   1   0><col2,
1   0   0   0>> OR by rows...
  0   0   1
  1   0   0
  0   1   0
>
straightUp(W) is: Quat(0<quat, 0.707 0.707   0   0>)<'rot_matrix', <col0,   0   1   0   0><col1,   1   0   0
0><col2,   0   0  -1   0>> OR by rows...
  0   1   0
  1   0   0
  0   0  -1
>
straightUp(S) is: Quat(0.5<quat, -0.5 -0.5 0.5   0>)<'rot_matrix', <col0,   0   1   0   0><col1,   0   0  -1
0><col2,  -1   0   0   0>> OR by rows...
  0   0  -1
  1   0   0
```

```
  0  -1   0
>
STRAIGHT UP assertions passed.............................................
// --------- end STRAIGHT UP ------------------------)

// -------------- STRAIGHT DOWN ------------------')
straightDown(E) is: Quat(0.707<quat,   0   0 -0.707   0>)<'rot_matrix', <col0,   0  -1   0   0><col1,   1   0   0
0><col2,   0   0   1   0>> OR by rows...
  0   1   0
 -1   0   0
  0   0   1
>
straightDown(E) is: Quat(0.5<quat, -0.5 0.5 -0.5   0>)<'rot_matrix', <col0,   0  -1   0   0><col1,   0   0  -1
0><col2,   1   0   0   0>> OR by rows...
  0   0   1
 -1   0   0
  0  -1   0
>
straightDown(E) is: Quat(0<quat, -0.707 0.707   0   0>)<'rot_matrix', <col0,   0  -1   0   0><col1,  -1   0   0
0><col2,   0   0  -1   0>> OR by rows...
  0  -1   0
 -1   0   0
  0   0  -1
>
straightDown(E) is: Quat(0.5<quat, 0.5 -0.5 -0.5   0>)<'rot_matrix', <col0,   0  -1   0   0><col1,   0   0   1
0><col2,  -1   0   0   0>> OR by rows...
  0   0  -1
 -1   0   0
  0   1   0
>
STRAIGHT DOWN assertions passed.............................................
// --------- end STRAIGHT DOWN ----------------------)



 -------- BANK/ROLL ----------------

Banking/Rolling 90 degrees left...
plane_E_bankLeft90 is: Quat(0.707<quat, 0.707   0   0   0>)<'rot_matrix', <col0,   1   0   0   0><col1,   0   0   1
0><col2,   0  -1   0   0>> OR by rows...
  1   0   0
  0   0  -1
  0   1   0
>
plane_N_bankLeft90 is: Quat(0.5<quat, 0.5 0.5 -0.5   0>)<'rot_matrix', <col0,   0   0  -1   0><col1,   1   0   0
0><col2,   0  -1   0   0>> OR by rows...
  0   1   0
  0   0  -1
 -1   0   0
>
plane_W_bankLeft90 is: Quat(0<quat,   0 0.707 -0.707   0>)<'rot_matrix', <col0,  -1   0   0   0><col1,   0   0  -1
0><col2,   0  -1   0   0>> OR by rows...
 -1   0   0
  0   0  -1
  0  -1   0
>
plane_W_bankLeft90 is: Quat(0.5<quat, 0.5 -0.5 0.5   0>)<'rot_matrix', <col0,   0   0   1   0><col1,  -1   0   0
0><col2,   0  -1   0   0>> OR by rows...
  0  -1   0
  0   0  -1
  1   0   0
>
ROLL 90 deg left assertions passed...........................................


Banking/Rolling 180 degrees...
plane_E_bankLeft180 is: Quat(i)<'rot_matrix', <col0,   1   0   0   0><col1,   0  -1   0   0><col2,   0   0  -1   0>>
OR by rows...
  1   0   0
  0  -1   0
  0   0  -1
>
plane_N_bankLeft180 is: Quat(0<quat, 0.707   0 -0.707   0>)<'rot_matrix', <col0,   0   0  -1   0><col1,   0  -1   0
0><col2,  -1   0   0   0>> OR by rows...
  0   0  -1
  0  -1   0
 -1   0   0
>
plane_W_bankLeft180 is: Quat(k)<'rot_matrix', <col0,  -1   0   0   0><col1,   0  -1   0   0><col2,   0   0   1   0>>
OR by rows...
 -1   0   0
```

```
  0  -1   0
  0   0   1
>
plane_S_bankLeft180 is: Quat(0<quat, 0.707   0 0.707   0>)<'rot_matrix', <col0,   0   0   1   0><col1,   0  -1   0
0><col2,   1   0   0   0>> OR by rows...
  0   0   1
  0  -1   0
  1   0   0
>
ROLL 180 degrees assertions passed...........................................


Banking/Rolling 90 degrees right...
plane_E_bankRight90 is: Quat(0.707<quat, -0.707   0   0   0>)<'rot_matrix', <col0,   1   0   0   0><col1,   0   0  -1
0><col2,   0   1   0   0>> OR by rows...
  1   0   0
  0   0   1
  0  -1   0
>
plane_N_bankRight90 is: Quat(0.5<quat, -0.5 0.5 0.5   0>)<'rot_matrix', <col0,   0   0  -1   0><col1,  -1   0   0
0><col2,   0   1   0   0>> OR by rows...
  0  -1   0
  0   0   1
 -1   0   0
>
plane_W_bankRight90 is: Quat(0<quat,   0 0.707 0.707   0>)<'rot_matrix', <col0,  -1   0   0   0><col1,   0   0   1
0><col2,   0   1   0   0>> OR by rows...
 -1   0   0
  0   0   1
  0   1   0
>
plane_S_bankRight90 is: Quat(0.5<quat, -0.5 -0.5 -0.5   0>)<'rot_matrix', <col0,   0   0   1   0><col1,   1   0   0
0><col2,   0   1   0   0>> OR by rows...
  0   1   0
  0   0   1
  1   0   0
>
ROLL 90 deg right assertions passed...........................................

 -------- end BANK/ROLL ----------------

ALL PLANE ROTATION ASSERTIONS PASSED ..........................................

SEE THIS WEBSITE for DETAILED DIAGRAMS on the TESTS of the PLANE's rotations
https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/examples/index.htm
...test_matrices_and_vectors assertions passed
====================  FINISHED testing quaternions  ========================
... program completed...

Program ended with exit code: 0
```