DT285 Programming Lab 1

I will provide you with a header file named Affine.h, that contains the following declarations and definitions.

```
struct Hcoords {
   float x, y, z, w;
   Hcoords(void);
   Hcoords(float X, float Y, float Z, float W);
   float& operator[](int i) { return *(&x+i); }
   float operator[](int i) const { return *(&x+i); }
   static bool Near(float x, float y) { return std::abs(x-y) < 1e-5f; }
};
struct Point : Hcoords {
   Point(void);
   Point(float X, float Y, float Z);
   Point(const Hcoords& v) : Hcoords(v) { assert(Near(w,1)); }
};
struct Vector : Hcoords {
   Vector(void);
   Vector(float X, float Y, float Z);
   Vector(const Hcoords& v) : Hcoords(v) { assert(Near(w,0)); }
   bool Normalize(void);
};
struct Matrix {
   Hcoords row[4];
   Matrix(void);
   Hcoords& operator[](int i) { return row[i]; }
   const Hcoords& operator[](int i) const { return row[i]; }
};
struct Affine : Matrix {
   Affine(void);
   Affine(const Vector& Lx, const Vector& Ly, const Vector& Lz, const Point& D);
   Affine(const Matrix& M) : Matrix(M)
      { assert(Hcoords::Near(M[3][0],0) && Hcoords::Near(M[3][1],0)
        && Hcoords::Near(M[3][2],0) && Hcoords::Near(M[3][3],1)); }
};
Hcoords operator+(const Hcoords& u, const Hcoords& v);
Hcoords operator-(const Hcoords& u, const Hcoords& v);
Hcoords operator-(const Hcoords& v);
Hcoords operator*(float r, const Hcoords& v);
Hcoords operator*(const Matrix& A, const Hcoords& v);
Matrix operator*(const Matrix& A, const Matrix& B);
float dot(const Vector& u, const Vector& v);
float abs(const Vector& v);
```

```
Vector cross(const Vector& u, const Vector& v);
Affine Rot(float t, const Vector& v);
Affine Trans(const Vector& v);
Affine Scale(float r);
Affine Scale(float rx, float ry, float rz);
Affine Inverse(const Affine& A);
```

Note that the Hcoords structure represents homogeneous coordinates in general (where the w coordinate can take any value). The Point, and Vector structures, which use homogeneous coordinates, are derived from Hcoords. However, a Point must always have w = 1, and a Vector must always have w = 0. In addition, the Matrix structure represents general 4×4 matrices, whose components may take on any values. In contrast, the Affine structure, which is derived from the Matrix class, is used to represent three–dimensional affine transformation using homogeneous coordinates: 4×4 matrices whose last row is always h0, 0, 0, 1i.

The functions given in the above header file are described as follows.

Hcoords::Hcoords() — default constructor. Returns [0, 0, 0, 0] (the zero vector).

Hcoords::Hcoords(X,Y,Z,W) — constructor with initialization. Returns [X, Y, Z, W].

Hcoords::operator[](i) — subscripting operator. Returns the i–th component of a homogeneous coordinate vector. If i 6= 0, 1, 2, 3, the result is undefined. [Implemented.]

Hcoords::Near(x,y) – convenience function to compare two floating point numbers: returns true if x and y are close enough to be considered equal. [Implemented.]

Point::Point() — default constructor. Returns a point with components (0, 0, 0); i.e., the origin.

Point::Point(X,Y,Z) — constructor to initialize the components of a point. Returns a point with components (X, Y, Z).

Point::Point(v) – conversion operator to attempt to convert to a point. This will fail, and the program will crash, if w 6= 1. [Implemented.]

Vector::Vector() — default constructor. Returns a vector with components h0, 0, 0i.

Vector::Vector(X,Y,Z) — constructor to initialize the components of a vector. Returns a vector with components hX, Y, Zi.

Vector::Vector(v) – conversion operator to attempt to convert to a vector. This will fail, and the program will crash, if w 6= 0. [Implemented.]

Vector::Normalize() — normalize a vector. The components of the Vector structure are changed to yield a unit vector pointing in the same direction. If the original vector is the zero vector, the function should return false; otherwise, it returns true.

Matrix::Matrix() — default constructor. Returns the zero matrix: the 4×4 matrix whose entries are all zeros.

Matrix::operator[](i) — subscripting operator. Returns the i–th row of a 4×4 matrix. [Implemented.]

Affine::Affine — default constructor. Returns the 3D affine transformation corresponding to the trivial affine transformation whose linear part is the 0 transformation, and whose translation part is the 0 vector. Note that the resulting matrix is not the 4 × 4 matrix whose entries are all zeros; rather it is the same as the matrix for uniform scaling by 0 with respect to the origin, H0.

Affine::Affine(Lx,Ly,Lz,D) — constructor to initialize an affine transformation. The quantities Lx, Ly, Lz, and D give the values of the columns of the transformation.

Affine::Affine(M) — conversion operator to attempt to convert the 4 × 4 matrix M to an affine transformation. This will fail, and the program will crash, if the last row of M is not h0, 0, 0, 1i. [Implemented.]

operator+(u,v) — returns the sum u + v of two four–component vectors.

operator-(u,v) — returns the difference u − v of two four–component vectors.

operator-(v) — returns the component–wise negation −v of a four–component vector.

operator*(r,v) — returns the product rv of a scalar and a four–component vector.

operator*(M,v) — returns the result Mv of applying the 4 × 4 matrix M to the four– component vector v.

operator*(A,B) — returns the composition (matrix product) A ◦ B of the 4 × 4 matrices A and B.

dot(u,v) — returns the dot product ~u · ~v of two three–dimensional vectors.

abs(v) — returns the length |~v| of a three–dimensional vector.

cross(u,v) — returns the cross product (vector product) ~u × ~v of two three–dimensional vectors.

Rot(t,v) — returns the affine transformation Rt~v for counterclockwise rotation by the angle t (in radians) about the vector ~v.

Trans(v) — returns the affine transformation T~v for translation by the vector ~v.

Scale(r) — returns the affine transformation Hr for uniform scaling by r with respect to the origin.

Scale(rx,ry,rz) — returns the affine transformation Hrx,ry,rz for inhomogeneous scaling by factors rx, ry, rz with respect to the origin.

Inverse(A) – returns the inverse A−1 of the affine transformation A. If A is not invertible, the function call results in undefined behavior.

You are to implemented the functions in the above header file (except for the ones already implemented). Your implementation file should be named Affine.cpp. Only Affine.h and the standard header file cmath may be included (note that Affine.h already includes cmath); you may not alter the contents of this header file in any way.