Transform Matrices

Transform matrices should be 4x4 matrices of doubles. Use either a structure (C) or a class (C++) with a 4x4 array in the structure. Using the STL vector class for matrices would be a bit of overkill since the structures are fixed size. In C, the type would be declared as below.

```c
typedef struct {
    double m[4][4];
} Matrix;
```

You will also need a Vector type, as below.

```c
typedef struct {
    double v[4];
} Vector;
```

Vectors should have a zero as their homogeneous coordinate so they do not undergo translations. You can get away with only three values as part of the Vector class (because \( h = 0 \)). The implementation details are up to you. Note that the Vector_set function only takes three arguments \((x, y, z)\). If you wish, you can just define Vector and Point to be the same thing.

```c
typedef Point Vector;
```

Vector Methods

The following should be member functions of the Vector class.

- **void set(Vector *v, double x, double y, double z)** - Set the Vector to \((x, y, z, 0.0)\).
- **void print(Vector *v, FILE *fp)** - Print out the Vector.
- **double length(Vector *v)** - Returns the length of the vector.
  \[
  L = \|\vec{v}\| = \sqrt{v_x^2 + v_y^2 + v_z^2}
  \]  

- **void normalize(Vector *v)** - Normalize the Vector to unit length.
  \[
  \hat{v} = \vec{v} \left( \frac{1}{L} \right)
  \]  

The following should be outside the Vector class.

- **Vector &operator=(Vector *dest, Vector *src)** - Copy the src into the dest Vector.
- **double dot(Vector *a, Vector *b)** - Returns the scalar product of \(\vec{a}\) and \(\vec{b}\).
  \[
  d(a, b) = a_x b_x + a_y b_y + a_z b_z
  \]  

- **Vector cross(Vector *a, Vector *b)** - Calculates the cross product of \(\vec{a}\) and \(\vec{b}\) and returns the result.
  \[
  \begin{bmatrix}
  c_x \\
  c_y \\
  c_z
  \end{bmatrix} = \vec{a} \times \vec{b} =
  \begin{bmatrix}
  a_y b_z - a_z b_y \\
  a_z b_x - a_x b_z \\
  a_x b_y - a_y b_x
  \end{bmatrix}
  \]
2D and Generic Matrix Methods

The following methods should be defined for the Matrix class.

- void clear() – Set the matrix to all zeros.
- void identity() – Set the matrix to the identity matrix.

\[ M = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \] (9)

- double get(int r, int c) – Return the element of the matrix at row r, column c.
- void set(int r, int c, double v) – Set the element of the matrix at row r, column c to v.
- void set(Matrix *src) – Copy in the src matrix data.
- void transpose() – Transpose the matrix m in place.
- void xform(Point *p, Point *q) – Transform the point p by the matrix m and put the result in q. For this function, p and q need to be different variables.

\[ \vec{q} = M\vec{p} \] (10)

- void xform(Vector *p, Vector *q) – Transform the Vector p by the matrix m and put the result in q. For this function, p and q need to be different variables.
- void xform(Polygon *p) – Transform the points in the Polygon p (and surface normals, if they exist) by the matrix m.
- void xform(Polyline *p) – Transform the points in the Polyline p by the matrix m.
- void xform(Line *line) – Transform the points in line by the matrix m.
- void scale2D(double sx, double sy) – Premultiply the matrix by a scale matrix parameterized by sx and sy.

\[ M = S(s_x, s_y)M = \begin{bmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \] (11)

- void rotateZ(double cth, double sth) – Premultiply the matrix by a Z-axis rotation matrix parameterized by \( \cos(\theta) \) and \( \sin(\theta) \), where \( \theta \) is the angle of rotation about the Z-axis.

\[ M = R_Z(\cos(\theta), \sin(\theta))M = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 & 0 \\
\sin(\theta) & \cos(\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \] (12)

©2008 Bruce A. Maxwell
• **void translate2D(double tx, double ty)** – Premultiply the matrix by a 2D translation matrix parameterized by $t_x$ and $t_y$.

\[
M = T(t_x, t_y)M = \begin{bmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} M
\]  

(13)

• **void shear2D(double shx, double shy)** – Premultiply the matrix by a 2D shear matrix parameterized by $sh_x$ and $sh_y$.

\[
M = Sh(sh_x, shy)M = \begin{bmatrix}
1 & sh_x & 0 & 0 \\
sh_y & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} M
\]  

(14)

The following functions should also be defined on matrices, but not as part of the Matrix class.

• **Matrix &operator\*(Matrix &left, Matrix &right)** – Multiplies left times right and returns a reference to a the result.

\[
[M] = [left] [right]
\]  

(15)

• **void operator<<(std::ostream &os, const Matrix &right)** – Print out the matrix in a nice 4x4 arrangement with a blank line below.

• **Matrix &operator=(Matrix &left, Matrix &right)** – Does a proper copy of right to left. Returns a reference to left matrix.