6 Transformation Matrices

Transform matrices should be 4x4 matrices of doubles. Use a structure (C) with a 4x4 array in the structure. In C, the type could be declared as below.

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

You will also need a Vector type, as below.

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.

typedef Point Vector;

6.1 Vector Functions

- void Vector_set(Vector *v, double x, double y, double z) – Set the Vector to $(x, y, z, 0.0)$.
- void Vector_print(Vector *v, FILE *fp) – Print out the Vector.
- void Vector_copy(Vector *dest, Vector *src) – Copy the src Vector into the dest Vector.
- double Vector_length(Vector *v) – Returns the length of the vector.

$$L = \| \vec{v} \| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

- void Vector_normalize(Vector *v) – Normalize the Vector to unit length.

$$\hat{v} = \vec{v} \left( \frac{1}{L} \right)$$

- double Vector_dot(Vector *a, Vector *b) – Returns the scalar product of $\vec{a}$ and $\vec{b}$.

$$d(a, b) = a_x b_x + a_y b_y + a_z b_z$$

- void Vector_cross(Vector *a, Vector *b, Vector *c) – Calculates the the cross product of $\vec{a}$ and $\vec{b}$ and puts the result in $\vec{c}$.

$$\begin{bmatrix}
    c_x \\
    c_y \\
    c_z 
\end{bmatrix} = \vec{a} \times \vec{b} = \begin{bmatrix}
    a_y b_z - a_z b_y \\
    a_z b_x - a_x b_z \\
    a_x b_y - a_y b_x 
\end{bmatrix}$$
6.2 2D and Generic Matrix Functions

The following functions should be defined for matrices.

- **void Matrix_print(Matrix *m, FILE *fp)** — Print out the matrix in a nice 4x4 arrangement with a blank line below.

- **void Matrix_clear(Matrix *m)** — Set the matrix to all zeros.

- **void Matrix_identity(Matrix *m)** — Set the matrix to the identity matrix.

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

- **double Matrix_get(Matrix *m, int r, int c)** — Return the element of the matrix at row r, column c.

- **void Matrix_set(Matrix *m, int r, int c, double v)** — Set the element of the matrix at row r, column c to v.

- **void Matrix_copy(Matrix *dest, Matrix *src)** — Copy the src matrix into the dest matrix.

- **void Matrix_transpose(Matrix *m)** — Transpose the matrix m in place.

- **void Matrix_multiply(Matrix *left, Matrix *right, Matrix *m)** — Multiply left and right and put the result in m.

\[
[M] = [\text{left}][\text{right}]
\]  

Make sure that the function is written so that the result matrix can also be the left or right matrices.

- **void Matrix_xformPoint(Matrix *m, 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.

\[
\vec{q} = M\vec{p}
\]  

- **void Matrix_xformVector(Matrix *m, 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 Matrix_xformPolygon(Matrix *m, Polygon *p)** — Transform the points and surface normals (if they exist) in the Polygon p by the matrix m.

- **void Matrix_xformPolyline(Matrix *m, Polyline *p)** — Transform the points in the Polyline p by the matrix m.

- **void Matrix_xformLine(Matrix *m, Line *line)** — Transform the points in line by the matrix m.

- **void Matrix_scale2D(Matrix *m, double sx, double sy)** — Premultiply the matrix by a scale matrix parameterized by s_x and s_y.
\[ M = S(s_x, s_y)M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} M \] (12)

- `void Matrix_rotateZ(Matrix *m, 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.

\[ M = R_Z(\cos(\theta), \sin(\theta))M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M \] (13)

- `void Matrix_translate2D(Matrix *m, double tx, double ty)`— Premultiply the matrix by a 2D translation matrix parameterized by \( t_x \) and \( t_y \).

\[ M = T(t_x, t_y)M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M \] (14)

- `void Matrix_shear2D(Matrix *m, double shx, double shy)`— Premultiply the matrix by a 2D shear matrix parameterized by \( sh_x \) and \( shy \).

\[ M = Sh(sh_x, shy)M = \begin{bmatrix} 1 & sh_x & 0 & 0 \\ sh_y & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M \] (15)