17

1.000	0.000	0.000
0.000	1.000	7.000
0.000	4.000	1.000

18

1.000	0.000	0.000
0.000	1.000	7.000
0.000	4.000	1.000

19

1.000	0.000	0.000
0.000	1.000	7.000
0.000	4.000	1.000

20

1.000	0.000	0.000
0.000	1.000	7.000
0.000	4.000	1.000

21

1.000	0.000	0.000
0.000	1.000	7.000
0.000	4.000	1.000

Matrix multiplication

The matrix product of two 3 x 3 matrices is also a 3 x 3 matrix. The multiplication rule is as follows:

> *product[i][j]* = the dot product of the *ith* row of the left matrix with the *jth* column of the right matrix.

```
 1.0  1.0  0.0       1.0  0.0  2.0       1.0  1.0  2.0
-1.0  1.0  0.0   x   0.0  1.0  0.0   =  -1.0  1.0 -2.0
 0.0  0.0  1.0      -2.0  0.0  1.0      -2.0  0.0  1.0

 1.0  1.0  0.0       1.0  0.0  2.0       1.0  1.0  2.0
-1.0  1.0  0.0   x   0.0  1.0  0.0   =  -1.0  1.0 -2.0
 0.0  0.0  1.0      -2.0  0.0  1.0      -2.0  0.0  1.0

 1.0  1.0  0.0       1.0  0.0  2.0       1.0  1.0  2.0
-1.0  1.0  0.0   x   0.0  1.0  0.0   =  -1.0  1.0 -2.0
 0.0  0.0  1.0      -2.0  0.0  1.0      -2.0  0.0  1.0
```

Notes:

1. Matrix multiplication is not commutative: *A x B != B x A* in general
2. Since the elements of a column of a matrix don't occupy adjacent locations in memory you *can't use* *vl_dot3* directly in this computation.

The identity matrix is given by:

```
1.0  0.0  0.0
0.0  1.0  0.0
0.0  0.0  1.0
```

Multiplying any matrix on the left or on the right by the identity matrix yields the original matrix

Multiplication of a matrix times a vector.

The product of a 3 x 3 matrix with a 3-d column vector is a 3-d vector. The multiplication rule is as follows:

product[i] = the dot product of the *ith* row of the matrix with the vector.

```
 1.0   1.0   0.0        1.0         1.0
-1.0   1.0   0.0   x    0.0    =   -1.0
 0.0   0.0   1.0       -2.0        -2.0

 1.0   1.0   0.0        1.0         1.0
-1.0   1.0   0.0   x    0.0    =   -1.0
 0.0   0.0   1.0       -2.0        -2.0

 1.0   1.0   0.0        1.0         1.0
-1.0   1.0   0.0   x    0.0    =   -1.0
 0.0   0.0   1.0       -2.0        -2.0
```

Notes:

The product is defined only with the matrix on the left and the column vector on the right
If the (logical) column vector is actually stored as a 3 element array in C then *vec_dot3* may be used in the computation.

The transpose of a matrix:

The transpose of a three by three matrix is also a three by three matrix. Its elements are given by a simple rule:

transpose[i][j] = original[j][i]

```
 1.0   3.0   2.0          1.0   1.0 -2.0
 1.0   2.0  -2.0     =    3.0   2.0   0.0
-2.0   0.0   1.0          2.0  -2.0   1.0
```

Notes:

The diagonal elements of a matrix and its transpose are identical. Off diagonal elements are interchanged in a symmetrical way.

The transpose of a matrix is in general not the same as the inverse of a matrix.

Safely transposing a matrix in place

We have emphasized the danger of in place transpose when the input and output parameters might be aliases to the same location. However, there is a safe and efficient way to transpose in place...

The cross product of two vectors

Given two linearly independent (not parallel) vectors:

$$V = (v_x, v_y, v_z)$$
$$W = (w_x, w_y, w_z)$$

The *cross product* sometimes called *outer product* is a vector which is orthogonal (perpendicular to) both of the original vectors.

$$V \times W = (v_y w_z - v_z w_y, \quad v_z w_x - v_x w_z, \quad v_x w_y - v_y w_x)$$
$$(1 \times 0 - 1 \times -1, \; 1 \times 0 - 1 \times 0, \; 1 \times -1 - 1 \times 0)$$
$$(0 - -1, 0 - 0, -1 - 0)$$

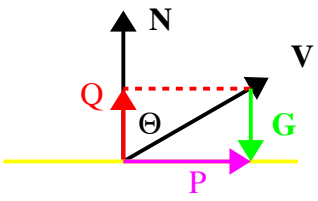
Notes:

The vector (0, -1, 0) is the negative *y axis*. Therefore, any vector that is perpendicular to it must lie in the *y = 0* plane. The projection of the vector (1, 1, 1) onto the *y = 0* plane is the vector (1, 0 1). The vector (1, 0, -1) is then perpendicular to this vector and lies in the *y=0* plane.

In a *right-handed* coordinate system

$$X \times Y = Z$$
$$Y \times Z = X$$
$$Z \times X = Y$$

Projection



Assume that *V and N* are unit vectors. The projection *Q* of *V* ***on N*** is shown in red. It is a vector in the same direction as *N* but having length cos(Θ). Therefore

$$Q = (N \text{ dot } V)\ N$$

Now assume that *N* is a normal to a plane shown as a yellow line. The projection *P* of *V* onto the plane is shown in *magenta* and is given by $V + G$ where *G* is the vector shown in green.

Since *G and Q* have clearly *have the same length* but *point in opposite directions*, $G = -Q$

Therefore the projection of a vector *V* onto a plane with normal *N* is given by:

$$P = V - (N \text{ dot } V)\ N \qquad (V - G)$$

or (possibly)

```
vl_diff3(vl_scale3(vl_dot3(N, V), N), V, P);
```

In building your new linear algebra routines it is desirable to build upon existing ones where possible but extreme levels of nesting of function calls as shown here *can complicate debugging.*

Rotation matrices

Rotation matrices are used to rotate coordinate systems in 3-space. They have some special properties:

The three rows are mutually orthogonal unit vectors. That is, the dot product of any pair of rows is 0.

The three columns are also mutually orthogonal unit vectors.

The inverse of a rotation matrix is its transpose.

The 1st row of a rotation matrix is a vector which will be mapped to [1, 0, 0] under the rotation. The 2nd row is a vector will be mapped to [0, 1, 0] and the third row is a vector that will be mapped to [0, 0, 1].

This example shows that the middle row is mapped to (0, 1, 0)

$$\begin{vmatrix} r_{0,0} & r_{0,1} & r_{0,2} \\ r_{1,0} & r_{1,1} & r_{1,2} \\ r_{2,0} & r_{2,1} & r_{2,2} \end{vmatrix} \begin{vmatrix} r_{1,0} \\ r_{1,1} \\ r_{1,2} \end{vmatrix} = \begin{vmatrix} 0 \\ 1 \\ 0 \end{vmatrix}$$

Operations on matrices and vectors:

It will be necessary to add the following functions to your vector algebra library. It is crucial that you support aliasing of parameters in a non-destructive way. For example, a call of the following format must work correctly:

```
mat_mul3(m1, m2, m1);
```

For this to work correctly, you will need to compute the answer in a *local matrix* and then *copy it back to the output matrix after it has been computed.*

```
/* Construct an identity matrix */

void identity_matrix3(double mtx[][3]);

/* Compute the outer product of two input vectors
 *
 * Parameters: v1 Left vector
 *             v2 Right vector
 *             v3 output vector
 */
void cross3(double *v1, double *v2, double *v3);


/* 3 x 3 matrix multiplier
 *
 * Parameters: x Left input matrix
 *             y Right input matrix
 *             z output matrix
 */

void mat_mul3(double x[][3], double y[][3], double z[][3]);
```

```
/*
 * 3 x 3 matrix transpose
 * Parameters: x original matrix
 *             z output matrix
 */

void xpose3(double  x[][3], double  z[][3]);


/*
 * Perform a linear transform in four dimensional space
 * By applying a 3 x 3 matrix to a 3 x 1 column vector
 * Parameters: y transform matrix
 *             x input vector
 *             z output vector

void xform3(double y[][3], double *x,double *z);


/*
 * project a vector onto a plane
 * Parameters: n    plane normal
 *             v    input vector
 *             w    projected vector
 */

void project3(double *n, double *v, double *w);
```

Examples of usage

```
#include "ray.h"
int main(){
   double m1[3][3];
   double m2[3][3];
   double m3[3][3];
   double v1[3];
   double v2[3];
   double v3[3];
   double xaxis[3] = {1.0, 0.0, 0.0};
   double yaxis[3] = {0.0, 1.0, 0.0};
   double zaxis[3] = {0.0, 0.0, 1.0};

   identity_matrix3(m1);   // create an identity matrix
   m1[0][1] = 2;   m1[0][2] = 3;
   m1[1][0] = 15;  m1[1][2] = 16;
   m1[2][0] = 22;  m1[2][1] = 23;
   mat_print3("Original", m1);
   xspose3(m1, m2);
   mat_print3("Transpose", m2);
   mat_mul3(m1, m2, m3);
   mat_print3("Product", m3);
```

```
Original
       1.000       2.000       3.000
      15.000       1.000      16.000
      22.000      23.000       1.000

 Transpose
       1.000      15.000      22.000
       2.000       1.000      23.000
       3.000      16.000       1.000

 Product
      14.000      65.000      71.000
      65.000     482.000     369.000
      71.000     369.000    1014.000
```

```
    v1[0] = 1; v1[1] = 2; v1[2] = 3;

    xform3(m2, v1, v2);
    vec_prn3("Transformed vector", v2);

    cross3(xaxis, yaxis, v2);
    vec_prn3("X x Y is:", v2);

    cross3(yaxis, zaxis, v2);
    vec_prn3("Y x Z is:", v2);

    cross3(zaxis, xaxis, v2);
    vec_prn3("Z x X is:", v2);

    cross3(xaxis, zaxis, v2);
    vec_prn3("X x Z is:", v2);

    cross3(xaxis, xaxis, v2);
    vec_prn3("X x X is:", v2);
}

Transformed vector
    97.000  73.000  38.000

X x Y is:
    0.000   0.000   1.000

Y x Z is:
    1.000   0.000   0.000

Z x X is:
    0.000   1.000   0.000

X x Z is:
    0.000  -1.000   0.000

X x X is:
    0.000   0.000   0.000
```