University of Bologna - Department of Mathematics

Piazza di Porta S.Donato, 5 - 40127 - Bologna

MATRIX library
Programming Guide - Version 1.0

S. Bonetti G. Casciola

Department of Mathematics
University of Bologna

Bologna 2000

Abstract

This report describes the *MATRIX* library. It is designed to set up and develop matrices, in order to make geometric projections and transformations.

S. Bonetti G. Casciola
Department of Mathematics, University of Bologna, P.zza di Porta S.Donato 5, Bologna, Italy. E-mail: casciola@dm.unibo.it.

Contents

Chapter 1

What is *MATRIX* ?

MATRIX is a library of useful functions to set up and develop matrices, in order to make geometrical projections and transformations. Its creation is justified by the fact various packets of *xcmodel* (see [XCMODEL00]) use it. It functions on the following types of data:

```
typedef double VEC_real_t;
typedef VEC_real_t VEC_vector_2_t[2];
typedef VEC_real_t VEC_vector_t[3];
typedef VEC_real_t Plane_t[4];
typedef VEC_real_t VEC_matrix_4x3_t[4][3];
typedef VEC_real_t VEC_matrix_4x4_t[4][4];
```

The matrices treated all have real (double) elements. Their names are self-explanatory. A brief description is provided below.

Chapter 2

Generic utility functions

The following functions manage 4x3 matrices. These are an optimised implementation of 4x4 transformation matrices, whose last column is always $(0, 0, 0, 1)$.

```
BOOLEAN MAT_equal(VEC_real_t *m1, VEC_real_t *m2);
```

compares two 4x3 matrices `m1` and `m2`, element by element. Returns TRUE if equal, otherwise FALSE (equal means according to a fixed tolerance).

```
void MAT_identity(VEC_matrix_4x3_t m);
```

initializes the 4x3 matrix `m` with the identity matrix.

```
void MAT_null(VEC_real_t *m);
```

initializes the 4x3 matrix `m` with the null matrix. (zeros the elements of `m`).

```
void MAT_add(VEC_real_t *m, VEC_real_t *m1, VEC_real_t *m2);
```

adds two matrices `m1` and `m2`; stores the result in `m`.

```
void MAT_sub(VEC_real_t *m, VEC_real_t *m1, VEC_real_t *m2);
```

subtracts the matrix `m2` from `m1`; stores the result in `m`.

```
void MAT_copy(VEC_real_t *m1, VEC_real_t *m2);
```

copies the elements of the matrix `m2` to `m1`. Because a matrix is seen as an array we need to do the cast

```
(VEC_real_t *)
```

to the passed matrices.

```
void MAT_multiply(VEC_matrix_4x3_t m, VEC_matrix_4x3_t m1,
                  VEC_matrix_4x3_t m2);
```

performs the matrix multiplication of `m1` and `m2`; stores the result in `m`.

```
VEC_real_t MAT_det_3x3(VEC_real_t B[3][3]);
```

computes the determinant of a 3x3 matrix.

```
BOOLEAN MAT_inverse(VEC_matrix_4x3_t Ainv, VEC_matrix_4x3_t A);
```

computes the inverse of a matrix `A`; stores the result in `Ainv`. A return value 0 means that `A` is not invertible.

Chapter 3

Affine tranformations

The following functions generate transformation matrices, that is, those matrices typically used to translate, rotate and scale an object. Each of these finishes with `abs` (absolute transformation) or `rel` (relative transformation), according to the type of transformation that you wish to create. In the first case, the matrix provided will be written over. If a rel function is used, the transformation required will be added to the one already present in the matrix. These transformations are defined with respect to the origin. If you wish to make a transformation with respect to another point, the object should be translated from that point to the origin, the operation should be carried out, and then the inverse translation performed.

```
void MAT_x_rotate_abs(VEC_matrix_4x3_t m, VEC_real_t c,
                      VEC_real_t s);
void MAT_x_rotate_rel(VEC_matrix_4x3_t m, VEC_real_t c,
                      VEC_real_t s);
void MAT_y_rotate_abs(VEC_matrix_4x3_t m, VEC_real_t c,
                      VEC_real_t s);
void MAT_y_rotate_rel(VEC_matrix_4x3_t m, VEC_real_t c,
                      VEC_real_t s);
void MAT_z_rotate_abs(VEC_matrix_4x3_t m, VEC_real_t c,
                      VEC_real_t s);
void MAT_z_rotate_rel(VEC_matrix_4x3_t m, VEC_real_t c,
                      VEC_real_t s);
```

there is a rotate function for each co-ordinate axis, that is return in `m` or add to `m` a transformation matrix that rotates points around the co-ordinate axis; `c` and `s` parameters are the sin and cos of the rotation angle measured anticlockwise.

```
void MAT_scale_abs(VEC_matrix_4x3_t m, VEC_real_t a,
                   VEC_real_t b, VEC_real_t c);
```

```
void MAT_scale_rel(VEC_matrix_4x3_t m, VEC_real_t a,
                   VEC_real_t b, VEC_real_t c);
```

return in `m` or add to `m` a transformation matrix that scales dimensions by `a`, `b` and `c` with respect to the coordinate axes;

```
void MAT_translate_abs(VEC_matrix_4x3_t m, VEC_real_t a,
                       VEC_real_t b, VEC_real_t c);
void MAT_translate_rel(VEC_matrix_4x3_t m, VEC_real_t a,
                       VEC_real_t b, VEC_real_t c);
```

return in `m` or add to `m` a transformation matrix that translates dimensions by `a`, `b` and `c` with respect to the coordinate axes;

```
void MAT_x_shear_abs(VEC_matrix_4x3_t m, VEC_real_t a,
                     VEC_real_t b);
void MAT_x_shear_rel(VEC_matrix_4x3_t m, VEC_real_t a,
                     VEC_real_t b);
void MAT_y_shear_abs(VEC_matrix_4x3_t m, VEC_real_t a,
                     VEC_real_t b);
void MAT_y_shear_rel(VEC_matrix_4x3_t m, VEC_real_t a,
                     VEC_real_t b);
void MAT_z_shear_abs(VEC_matrix_4x3_t m, VEC_real_t a,
                     VEC_real_t b);
void MAT_z_shear_rel(VEC_matrix_4x3_t m, VEC_real_t a,
                     VEC_real_t b);
```

return in `m` or add to `m` a transformation matrix that shears points on the coordinate axes. Shear is a particular type of transformation. Consider, for example, the one in x: the dimensions in x are fixed and those in y and z are modified respectively of a factor a and b. The x_axis shear matrix is

$$\begin{pmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Chapter 4

Conversion functions

```
void MAT_convert_to_4x4(VEC_matrix_4x4_t m1,
                        VEC_matrix_4x3_t m2);
```

This function constructs the matrix 4x3 `m2` from the transformation matrix 4x4 `m1`. Remember that a transformation matrix in homogeneous coordinates has as its final column the common vector $(0, 0, 0, 1)$.

```
void MAT_convert_to_4x3(VEC_matrix_4x3_t m1,
                        VEC_matrix_4x4_t m2);
```

When the function is defined, it is useful to have the inverse function. This is the inverse of the previous function.

```
void MAT_transform_plane(VEC_real_t *result,
                         VEC_real_t *start_eq,
                         VEC_matrix_4x3_t inv_mat);
```

Transforms the plane equation `start_eq` in `result` by the matrix `inv_mat`.

Chapter 5

Functions for 4x4 matrices

Follow some functions, similar to the previous ones, but designed for generic 4x4 matrices.

```
void MAT_identity_4x4(VEC_matrix_4x4_t m);
```

sets `m` to the identity matrix.

```
void MAT_null_4x4(VEC_real_t *m);
```

sets `m` to the null matrix.

```
void MAT_transpose_4x4(VEC_matrix_4x4_t m1, VEC_matrix_4x4_t m2);
```

computes the transpose of `m2`; stores the result in `m1`.

```
void MAT_copy_4x4(VEC_real_t *m1, VEC_real_t *m2);
```

copies `m2` in `m1` element by element; because a matrix is seen as an array we need to do the cast

```
(VEC_real_t *)
```

to the passed matrices.

```
void MAT_multiply_4x4(VEC_matrix_4x4_t m, VEC_matrix_4x4_t m1,
                      VEC_matrix_4x4_t m2);
```

performs the matrix multiplication of `m1` and `m2`; stores the result in `m`.

```
BOOLEAN MAT_inverse_4x4(VEC_matrix_4x4_t Ainv, VEC_matrix_4x4_t A);
```

computes the inverse of a matrix `A`; stores the result in `Ainv`. A return value 0 means that `A` is not invertible.

```
VEC_real_t MAT_det_4x4(VEC_matrix_4x4_t A);
```

computes the determinant of a 4x4 matrix.

```
void minore_4x4(VEC_real_t B[4][4], VEC_matrix_4x4_t A,
                int i, int j);
```

returns in `B` the square submatrix of `A` (4x4) starting from the subscripts `i` and `j`.

Chapter 6

Functions for vectors

Follows some generic function for vectors.

```
VEC_real_t VEC_max(VEC_vector_t start_vec, VEC_real_t *result,
                   int *result_loc);
```

returns in `result` the largest element of the vector `start_vec`; `result_loc` is the subscript of the largest element of the vector.

```
VEC_real_t VEC_min(VEC_vector_t start_vec, VEC_real_t *result,
                   int *result_loc);
```

returns in `result` the smallest element of the vector `start_vec`; `result_loc` is the subscript of the smallest element of the vector.

```
void VEC_triple(VEC_vector_t result_vec, VEC_vector_t a,
                VEC_vector_t b, VEC_vector_t c);
```

computes the triple product of `a`, `b` and `c`; stores the result in `result_vec`.

Bibliography

[XCMODEL00] G.Casciola, *xcmodel*: a system to model and render NURBS curves and surfaces; User's Guide - Version 1.0, Progetto MURST: "Analisi Numerica: Metodi e Software Matematico", Ferrara (2000), http://www.dm.unibo.it/∼casciola/html/xcmodel.html