IX.7

MATRIX INVERSION

Richard Carling
Independent Graphics Consultant
Bedford, Massachusetts

This Gem demonstrates how to calculate the inverse and adjoint of a 4-by-4 matrix.

Computation of the inverse of a matrix $M(M^{-1})$ is an important step in many image synthesis algorithms. Typically the projection of objects from world space to image space is accomplished with a transformation matrix. The inverse of this projection may be represented by the inverse of the transformation matrix.

The inverse matrix is used freely in computer graphics papers and published algorithms, but routines for its calculation are usually left to the reader. There are several methods for calculating the inverse of a matrix; the one presented here is one of the easiest to follow, though it is not the fastest. If the inverse of a matrix larger than 4-by-4 is needed or if speed is critical, other methods of calculating the inverse may be more suitable.

A common use of the inverse matrix is in texture mapping applications, where screen coordinates need to be mapped back onto the surface of an object in world space. The inverse is also useful in ray tracing, when one wishes to intersect a parametrically defined object. Rather than actually transform the object from it's parametric space into world space, the ray is inverse-transformed into the object's canonical space, where the intersection is typically easier and faster. One example of this technique is described in Blinn (1989).

The inverse is also useful for transforming from one device space to another. For example, the shadow map technique of Williams (1978) requires transforming a point in screen space to a 3D coordinate system

defined by the light source. In this algorithm, a depth buffer is constructed by rendering the scene from the point of view of the light source. To determine if a particular point is in shadow, one finds its line of sight in the coordinate system of the light source, and looks up (from the depth buffer) the nearest surface visible from the light in that direction. If the point in question is farther from the light than the stored object, it is blocked by that object and is in shadow. This calculation requires taking a screen space coordinate and transforming it first into world space, and then into the space of the light source for look-up in the depth buffer.

The adjoint matrix (M^*) is similar to the inverse, except for a scaling factor. In particular, $M^* = (1/\det(M))M^{-1}$ (each element $ad_{ij} = inv_{ij}/\det$), where det is the determinant of the original matrix. If the matrix is singular, then det = 0, so there is no inverse matrix. However, computing the adjoint does not require the determinant. The two main advantages of the adjoint are that it always exists, and that it is easier and faster to calculate than the inverse.

The distinction between the adjoint and the inverse may be demonstrated by observing the transformation of a vector. Suppose one calculates a vector V' by transforming an input V by a matrix M: $V' = VM$. We can post-multiply V' by M^{-1} to recover V: $V'' = V'M^{-1} = V$. Suppose instead we use the adjoint, then $V'' = V'M^* = VMM^* = VM(1/\det(M))M^{-1} = (1/\det(M))V$. Thus, using the adjoint rather than the inverse gets us back to the original input vector, but it has been scaled by $1/\det(M)$. A common use for the adjoint is for transforming surface normals, which typically must be rescaled to unit length after transformation, anyway, so there is no penalty for this scaling created by use of the adjoint.

See Appendix 2 for C Implementation (766)