Matrix Animation and Polar Decomposition

Ken Shoemake
Computer Graphics Laboratory
University of Pennsylvania
Philadelphia, PA 19104

Tom Duff
AT&T Bell Laboratories
Murray Hill, NJ 07974

Abstract

General 3×3 linear or 4×4 homogenous matrices can be formed by composing primitive matrices for translation, rotation, scale, shear, and perspective. Current 3-D computer graphics systems manipulate and interpolate parametric forms of these primitives to generate scenes and motion. For this and other reasons, decomposing a composite matrix in a meaningful way has been a long-standing challenge. This paper presents a theory and method for doing so, proposing that the central issue is rotation extraction, and that the best way to do that is Polar Decomposition. This method also is useful for renormalizing a rotation matrix containing excessive error.

Résumé

Des matrices correspondant à des transformations linéaires en 3 dimensions, ou bien à des transformations homogènes en 4 dimensions, peuvent être construites en composant des matrices qui décrivent des transformations élémentaires: déplacement, rotation, homothétie, glissement, et perspective. Les systèmes actuels de visualisation graphique a trois dimensions manipulent des formes paramétriques de ces transformations élémentaires, pour recréer des scènes et des mouvements. Il en découle l'intéret de trouver des décompositions pratiques de matrices composées. Nous présentons ici une technique pour trouver de telles decompositions. Le problème fondamental est l'extraction des rotations, et nous démontrons qu'une décomposition polaire est la méthode de choix. Cette méthode est aussi utile quant il faut renormaliser une matrice de rotation qui contient des erreurs excessives.

Keywords: homogeneous matrix, matrix animation, interpolation, rotation, matrix decomposition, Polar Decomposition, QR Decomposition, Singular Value Decomposition, Spectral Decomposition, greedy algorithm

Introduction

Matrix composition is well established as an important part of computer graphics practice and teaching [Foley 90]. It is used to simplify and speed the transformation of points, curves, and surfaces for modeling, rendering, and animation. Matrix decomposition—the focus of this paper—is less well known in computer graphics. It is useful for a variety of purposes, especially animation and interactive manipulation.

The usual transformations of an object can be described by 3×4 affine matrices; but the 12 entries of such a matrix are not very meaningful parameters. To understand, much less modify, matrices requires a good decomposition. Any decomposition must account for all 12 degrees of freedom (16 for 4×4 matrices) in the independent parameters of the primitives used. A decomposition that provides too few parameters will not be able to handle all inputs, while one that provides too many will not be stable and well-defined. The greatest problem, however, is ensuring that the decomposition is meaningful.

Most widely used 3-D animation systems, typified by Stern's *bbop* at NYIT [Stern 83], Gomez's *twixt* at Ohio State [Gomez 84] and Duff's *md* at Lucasfilm (later Pixar) allow the parameters of primitive transformations to be set interactively at key times, and compute transformations at intermediate times by spline interpolation in parameter space. Sometimes, however, only a composite matrix is available at each key frame, or more design flexibility is needed than that allowed by a hierarchy of primitive transformations. It is possible to interpolate the entries of a composite matrix directly, but the results are usually unsatisfactory. Decomposition allows the use of standard interpolation methods, and can give much better results. Matrix animation is discussed in more detail below.

Most authors have considered decomposition with less stringent criteria than ours. A common motivation is the need to synthesize an arbitrary matrix from a limited set of primitives, without regard for meaningfulness of the decomposition [Thomas 91]. Typically, these methods rely on a sequence of shears [Greene 86], and give factors that depend on the coordinate basis used. Shears are one of the less common operations in graphics, and a sequence of shears is a poor choice for animation. In contrast, the decomposition we propose has a simple, physical, coordinate independent interpretation, preserves rigid body motion as much as possible, and animates well.

Composition and Decomposition

Three types of matrix are commonly used for 3-D graphics: 3×3 linear, 3×4 affine, and 4×4 homogeneous; similar types with one less column and row are used for 2-D graphics. The homogeneous matrix is most general, as it is able to represent all the transformations required to place and view an object: translation, rotation, scale, shear, and perspective. Any number of transformations can be multiplied to form a composite matrix, so that a point can be transformed from model space coordinates to screen space coordinates in a single step.Generally, however, perspective is treated as a separate step in the viewing process—because lighting calculations must be done first—and not used for modeling or object placement. All the transformations except perspective can be accomodated by an affine matrix, which, in turn, can be considered just a 3×3 linear matrix with a translation column appended. (Following [Foley 90], we write points as column vectors, $[x\ y\ z\ 1]^T$, which are multiplied on the left by the matrix.)

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & w+1 \end{pmatrix} \qquad \mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) & 0 \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz-wx) & 0 \\ 2(xz-wy) & 2(yz+wx) & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{K} = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{H} = \begin{pmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1. Primitive Transformation Matrices

Each primitive transformation has a more meaningful and concise representation than its matrix: a vector for most, a quaternion for rotations. It is not too difficult to recover the concise form if the matrix for the primitive is available [Goldman 91][Shoemake 91]. Once primitives have been multiplied into a composite matrix, however, recovery is generally impossible. Even so, a great deal can be done, as we shall see.

Primitive recovery is difficult for three reasons: absorption, order, and interaction. The first two problems are intractable; the third is the focus of this paper. Absorption is simple: a sequence of translations gives a result which is indistinguishable from a single translation, or from any number of *different* sequences; the same is true of other primitives. Order is also simple: the effect of a translation followed by a scale could as easily be achieved by composing primitives in the opposite order; likewise for other pairs. Interaction is more subtle: most transformations change all columns of the matrix, so scaling (for example) affects translation; all pairs of primitives interact. Notice any shear can be achieved by combining rotation and scale.

While absorption and order cannot be unscrambled, they can be standardized; for animation and other applications of interest, this usually suffices. Absorption can simply be ignored; that is, no attempt is made to tease apart a translation (except perhaps into x, y, and z components). Order is handled most easily by assuming a canonical order, such as Perspective of Translation of Rotation of Scale of object. Which canonical order is chosen is partly a matter of taste; this particular one makes translation trivial to extract, and places perspective in the order expected for a transformation to camera coordinates. If more information is made available in a particular situation, it may be possible to improve upon these standard assumptions; for example, it may be known that only x translation took place, or that scaling was done last. Such special case extraction is outside the scope of this paper.

Rigidity and Rotation

A perspective matrix of the form given above is easy to extract as a left factor of a composite homogeneous matrix, $\mathbf{C} = \mathbf{PA}$, with non-singular 3×3 corner; the details are left as an exercise for the reader. Notice that the usual perspective matrix includes translation and scale; we have chosen the minimal form necessary to reduce $\mathbf{C}$ to an affine matrix.† Likewise, a translation is easy to extract as the left factor of the remaining affine matrix, $\mathbf{A} = \mathbf{TM}$; simply strip off the last column. The matrix $\mathbf{M}$ then essentially will be the 3×3 matrix of a linear transformation. It would be simplest not to factor $\mathbf{M}$ at all, but to animate its entries directly. The results of this overly simple approach are visually disconcerting, but worth investigating.

Direct matrix interpolation treats each component of the matrix separately, and creates intermediate matrices as weighted sums of nearby key matrices. For example, linear interpolation between keys $\mathbf{M}_1$ and $\mathbf{M}_2$ uses $(1-t)\mathbf{M}_1+t\mathbf{M}_2$, while cubic spline interpolation uses affine combinations, $\alpha_1\mathbf{M}_1+\alpha_2\mathbf{M}_2+\alpha_3\mathbf{M}_3+\alpha_4\mathbf{M}_4$, with $\alpha_1+\alpha_2+\alpha_3+\alpha_4 = 1$. The results of this approach are immediately deduced from the linearity of matrix multiplication.

Proposition: A point transformed by a weighted sum of matrices equals the weighted sum of the transformed points; that is,

$$(\sum_i \alpha_i \mathbf{M}_i)\, p = \sum_i \alpha_i (\mathbf{M}_i p).$$

An example of this behavior can be seen in Figure 2, where the chin and hat back move steadily along a line from initial to final position, as do all the points. (We will use planar examples because they are easier to interpret on the page, but illustrate the same issues as spatial examples.) Notice that the interpolated matrix twice becomes singular as the

† But a permutation matrix may also be needed to provide pivoting for what is, in effect, a block LU decomposition.

image appears to flip over. At any moment of singularity the image will collapse onto a line (or worse, a point).

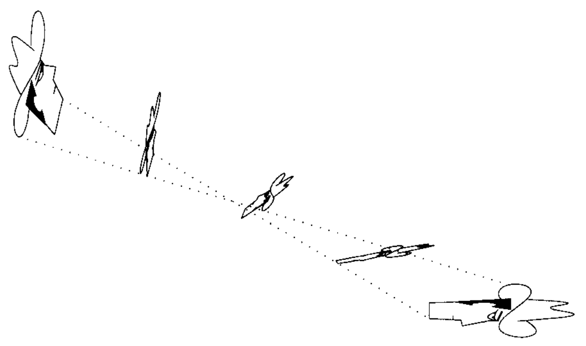

Figure 2. Direct Matrix Interpolation

Consider a square centered at the origin, and two key matrices: the identity and a 180° rotation. Since there are only two keys, only linear interpolation makes sense. Then, however, the theorem implies that each corner of the square will move linearly to its rotated position, which is diagonally opposite; the square will collapse through the origin. Although the distortions diminish with smaller angles of rotation, the square loses its shape. We expect rotations to transform the shape rigidly; direct matrix interpolation fails to do so. On the other hand, there is no problem with matrices for translation, scale, or shear.

Experiments with apparent motion (flash one image, flash another, see motion) suggest that the human visual system infers rigid motion as much as possible [Carlton 90] [Shepard 84]. Rotation is the only rigid transformation that is distorted by direct matrix interpolation. It therefore seems reasonable to conclude that the central problem for matrix animation is to extract a rotation in the best possible way, so that it can be interpolated *as* a rotation.

Decomposition Methods

Rotation matrices have simple defining properties: each column is a unit length vector which is perpendicular to the others, and the third column is the cross product of the first two. (Rows satisfy the same properties.) The first two properties are those of *orthogonality*, and can be summarized as $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$; the last makes the orthogonality *special*, and can be stated as $\det(\mathbf{Q}) = +1$. Orthogonality alone implies that the determinant must be either +1 or −1, with the latter indicating the presence of a reflection in the matrix. A 3×3 orthogonal matrix with negative determinant can be converted to a pure rotation by factoring out a $-\mathbf{I}$.

Numerical analysts have developed a number of algorithms for orthogonal matrices [Golub 89] [Press 88], in large part because orthogonality limits the accumulation of numerical error. Given a square—and presumably non-singular—matrix, three promising orthogonal decompositions are available: QR decomposition, Singular Value Decomposition (SVD), and Polar Decomposition. The QR factors of a matrix $\mathbf{M} = \mathbf{QR}$ are, respectively, orthogonal and lower triangular. The SVD gives three factors, $\mathbf{M} = \mathbf{UKV}^T$, with $\mathbf{U}$ and $\mathbf{V}$ orthogonal and $\mathbf{K}$ diagonal and positive. The less common Polar Decomposition, $\mathbf{M} = \mathbf{QS}$, yields an orthogonal factor and a symmetric positive definite factor. The latter two decompositions can factor singular matrices, with “positive” replaced by “non-negative” in the factors.

More than one algorithm is available to compute each decomposition. The oldest and best-known method for QR Decomposition is called Gram-Schmidt orthogonalization. Each row of the matrix is considered in turn, with each divided by its magnitude to give a unit vector, then projected onto the remaining rows to subtract out any parallel component in each of them. A better method is to accumulate Householder reflections, orthogonal transformations which can zero out the elements above the diagonal.

There is no simple SVD algorithm. The most common approach is first to use Householder reflections to make $\mathbf{M}$ bidiagonal, then to perform an iteration involving QR Decomposition until the off-diagonal entries converge to zero. While this is numerically reliable, it is complicated to code, and by no means cheap.

It is possible to compute a Polar Decomposition using the results of SVD, suggesting great cost; but a simpler method is available [Higham 86]. Compute the othogonal factor by averaging the matrix with its inverse transpose until convergence: Set $\mathbf{Q}_0 = \mathbf{M}$, then $\mathbf{Q}_{i+1} = {}^1\!/_2(\mathbf{Q}_i + \mathbf{Q}_i^{-T})$ until $\mathbf{Q}_{i+1} - \mathbf{Q}_i \approx \mathbf{0}$. This is essentially a Newton algorithm for the square root of $\mathbf{I}$, and converges quadratically when $\mathbf{Q}_i$ is nearly orthogonal. Finding the $\mathbf{Q}$ factor of a 2×2 matrix is easy. Suppose

$$\mathbf{M} = \begin{pmatrix} a & b \\ c & d \end{pmatrix};$$

then

$$\mathbf{Q} = \mathbf{M} + \operatorname{sign}(\det(\mathbf{M}))\begin{pmatrix} d & -c \\ -b & a \end{pmatrix},$$

scaled by a factor that makes the columns unit vectors.

Polar Decomposition Advantages

Care is needed in choosing among the possibilities, since the purposes of numerical linear algebra are different from those of computer graphics. The worst of the three choices seems to be SVD: it is the most expensive to compute, and the orthogonal matrices it produces are practically useless. A matrix which is already a pure rotation can be factored in an infinite variety of ways into the two orthogonal matrices of the decomposition, which is disastrous in the context of matrix animation. Small perturbations of the input matrix can cause different orthogonal factors to be chosen, even though the set of singular values is stable. Interpolating unreliable matrices will produce erratic results: consider the following two decompositions.

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}99.3&0&0\\0&99.4&0\\0&0&99.5\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=$$
$$\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix}\begin{pmatrix}99.4&0&0\\0&99.3&0\\0&0&99.5\end{pmatrix}\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}.$$

Although both are perfectly valid decompositions, interpolation of the primitives will give visible distortions —not at all what the user expects! Floating point variations in the least significant digit can cause an SVD algorithm to choose the first decomposition for one key, and the second for the next. Many SVD routines order the singular values by magnitude, which only exacerbates the problem. There seems to be no way to avoid having small input changes cause large output changes.

QR Decomposition is a much better choice, though it still presents problems. Unlike the factors of SVD, the QR factors can be determined uniquely, and are stable under small perturbations. Also, the algorithms for QR are simple and efficient. The drawback is that the orthogonal matrix extracted is not particularly meaningful: it is not independent of the coordinate basis used, and so has no "physical" significance. That is, if the matrix **M** is given in a rotated and uniformly scaled basis $\mathbf{M}' = \mathbf{BMB}^{-1}$, coordinate independent factors would have the form $\mathbf{Q}' = \mathbf{BQB}^{-1}$ and $\mathbf{R}' = \mathbf{BRB}^{-1}$; but the latter is no longer a lower triangular matrix, since that property is not preserved under similarity transforms. This is unfortunate for animation purposes, because it makes results much less predictable. Suppose, for example, **M** is constructed by rotating then scaling; although the **Q** factor might be expected to capture the rotation, it will not. Only when **M** is constructed by scaling then rotating will QR recover the original factors.

The Polar Decomposition factors are unique, coordinate independent, and simple and efficient to compute. Furthermore, the orthogonal factor **Q** is the closest possible orthogonal matrix to **M**, a property which is also coordinate independent. That is, **Q** satisfies the following conditions.

Find **Q** minimizing $\| \mathbf{Q}-\mathbf{M} \|_F^2$
subject to $\mathbf{Q}^T\mathbf{Q} - \mathbf{I} = \mathbf{0}$,

where the measure of closeness, the Frobenius matrix norm squared, is

$$\| \mathbf{Q}-\mathbf{M} \|_F^2 = \sum_{i,j} (q_{ij}-m_{ij})^2 .$$

Since this important claim appears in [Higham 88] without proof, a proof is given in the Appendix. When **M** has positive determinant, **Q** will be a pure rotation, otherwise it will include a reflection. It might seem preferable to exclude reflections, but there is no well-defined nearest rotation. For example, every 2-D rotation is equally distant from every 2-D reflection. (Polar Decomposition is applicable to matrices of any size and shape.) A rotation has the form $\begin{pmatrix}c&-s\\s&c\end{pmatrix}$, with $c^2+s^2=1$, while a reflection is $\begin{pmatrix}a&b\\b&-a\end{pmatrix}$, with $a^2+b^2=1$. The sum of the squares of the differences is $(c-a)^2+(-s-b)^2+(s-b)^2+(c+a)^2 = 2(c^2+s^2)+2(a^2+b^2) = 4$. As noted earlier, however, a 3x3 **Q** matrix which includes a reflection (indicated by a negative determinant) can be factored as $\mathbf{Q} = \mathbf{R}(-\mathbf{I})$.

Closeness also makes Polar Decomposition good for matrix renormalization. Moderate amounts of numerical noise can be removed in a single iteration of the averaging algorithm. This improves and formally grounds [Raible 90].

The combination of uniqueness and closeness guarantees that small input perturbations will not produce large output variations. The **Q** factor of Polar Decomposition appears to be the best possible rotation. What, then, is the **S** factor? As the appendix shows, in some rotated coordinate system **S** is diagonal—in other words, a scale matrix. This form of scaling is preserved through coordinate changes, and has a good claim to being a new primitive, *stretch*. The **S** factor can move to the other side of the **Q** factor without changing form, though its value will change to $\mathbf{Q}^T\mathbf{S}\mathbf{Q}$. Thus Polar Decomposition has a very physical interpretation.

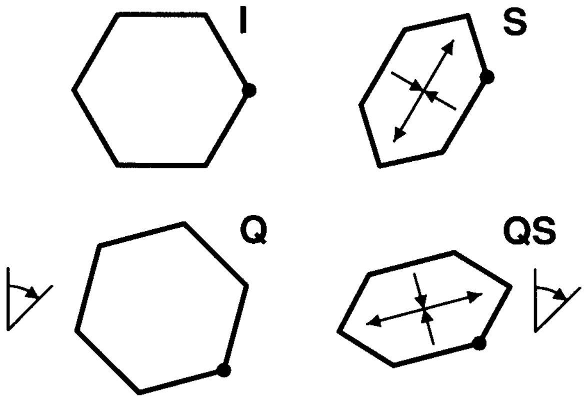

Figure 3. Physical View of Polar Decomposition

One drawback of Polar Decomposition is that there is no explicit representation of shear. As explained earlier, interaction is to blame; shear will be factored as rotation and stretch. In two dimensions, for example, a simple shear will factor as

$$\begin{aligned}\mathbf{H} &= \begin{pmatrix}1&h\\0&1\end{pmatrix}\\ &= \frac{1}{\sqrt{4+h^2}}\begin{pmatrix}2&h\\-h&2\end{pmatrix}\begin{pmatrix}2&h\\h&2+h^2\end{pmatrix}\frac{1}{\sqrt{4+h^2}}\\ &= \mathbf{QS}.\end{aligned}$$

As Figures 4 and 5 show, the appearance of a factored animation can be quite different from that of a direct animation for shear. Nevertheless, factorization gives a reasonable result.

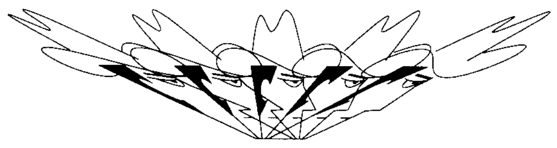

Figure 4. Direct Shear Interpolation

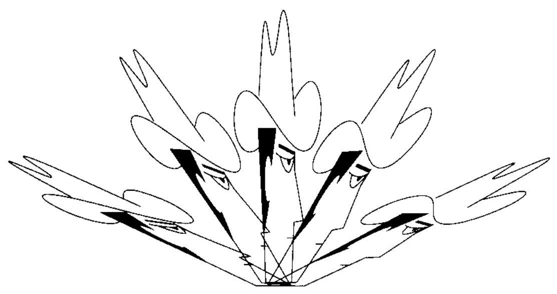

Figure 5. Decomposed Shear Interpolation

Direct Stretch Animation

Although $\mathbf{S}$ can be factored into diagonal form, $\mathbf{S} = \mathbf{UKU}^T$ (using a symmetric eigenvalue routine [Golub 89] [Carnahan 69]), as with SVD the factorization is not unique. This unavoidable indeterminacy combined with small numerical errors could cause different $\mathbf{U}$'s to be chosen at different keys, and the resulting interpolation would suffer greatly. Fortunately, however, $\mathbf{S}$ matrices can be interpolated directly, and will preserve their form and meaning. That is, $\alpha_1\mathbf{S}_1+\alpha_2\mathbf{S}_2+\ldots$ yields a symmetric matrix, which for non-negative α_i will also be positive definite, so it is not necessary to choose a diagonalizing rotation $\mathbf{U}$. If some $\mathbf{U}$ diagonalizes both $\mathbf{S}_1$ and $\mathbf{S}_2$ simultaneously, then $\alpha_1\mathbf{S}_1+\alpha_2\mathbf{S}_2 = \mathbf{U}(\alpha_1\mathbf{K}_1+\alpha_2\mathbf{K}_2)\mathbf{U}^T$, so direct interpolation simply interpolates the scale factors, as desired. Weights α_i for interpolation will usually include negative values (to ensure smooth motion), so the interpolated $\mathbf{S}$ matrices can become singular; but the same thing can happen with pure scale matrices. In both cases this does not seem to be a serious problem, and can be solved using spline tension.

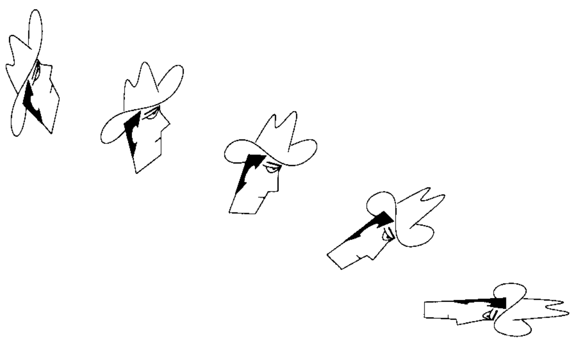

Figure 6. Polar Decomposed Matrix Interpolation

Factored Stretch Animation

Diagonalization of $\mathbf{S}$ as $\mathbf{UKU}^T$ is still a useful alternative if it can be stabilized across keys. (Even without stabilization, an interactive user interface will certainly deal with stretch in factored form.) So in this section—which can be skipped on first reading—we consider the following problem: Given two stretch matrices, $\mathbf{S}_1$ and $\mathbf{S}_2$, interpolated in that order, how can their diagonalizing rotations, $\mathbf{U}_1$ and $\mathbf{U}_2$, be chosen to be as similar as possible? More precisely, if the rotation taking $\mathbf{U}_1$ into $\mathbf{U}_2$ is designated by $\mathbf{U}_{12} = \mathbf{U}_1{}^T\mathbf{U}_2$, the problem is to minimize the absolute angle of rotation performed by $\mathbf{U}_{12}$. Furthermore, so that the results can easily be generalized to a series of matrices $\mathbf{S}_i$, let $\mathbf{U}_1$ be fixed. (Then fix $\mathbf{U}_2$ while minimizing $\mathbf{U}_{23}$, and so on. Begin with $\mathbf{U}_0 = \mathbf{I}$.)

There are three cases, depending on how many identical values occur on the diagonal $\mathbf{K}_2$. When all three values are the same, it is possible to set $\mathbf{U}_2 = \mathbf{U}_1$. Uniform scaling is common in computer graphics practice, and is easily detected by inspection of $\mathbf{S}_2$, which will already be diagonal with identical values. When all three values are different, we have 24 choices for $\mathbf{U}_2$. These are obtained by all axis permutations (6), times all axis sign combinations (8), achievable by a rotation (divide by 2). When exactly two values are the same, we have an extension of the all different case: free rotation is allowed around one of the axes. The last two cases are discussed more fully below.

An easy way to measure the rotation $\mathbf{U}_{12}$ is to convert it into a unit quaternion. ([Shoemake 85] introduces unit quaternions as a representation of 3-D rotation and discusses how to interpolate them.) Its real component is $\cos(\theta/2)$, where θ is the total rotation angle. Picking $\mathbf{U}_2$ to maximize the quaternion's real component minimizes the angle.

There is a quick way to do this maximization. Let q be the quaternion corresponding to $\mathbf{U}_{12}$. The 24 variations correspond to qp, where p is one of 48 quaternions (including both p and $-p$) that map the coordinate axes into themselves: $p = [x\ y\ z\ w]$ can be one of $[0\ 0\ 0\ \pm 1]$, $[0\ 0\ \pm 1\ \pm 1]/\sqrt{2}$, $[\pm 1\ \pm 1\ \pm 1\ \pm 1]/2$, or a permutation of these. The real part of qp is $w_q w_p - x_q x_p - y_q y_p - z_q z_p$, which is simple to maximize because of the simple form of each p. We take the absolute values of the components of q, sort them, and choose the maximum of either the largest, or half the sum of all four, or $1/\sqrt{2}$ times the sum of the two largest. Then we can work backwards from our choices to deduce the corresponding p.

If exactly two of $\mathbf{K}_2$'s values are the same, we have a continuous optimization. As before, we are free to permute the axes, but we have the additional freedom to rotate by any angle in the plane of equal scaling. We can arrange for the equal values to be the first two, so that a change of coordinates rotating around the z axis leaves $\mathbf{K}_2$ unchanged. So our problem is to pick p and r to maximize the real component of qpr, where p is one of the 48 quaternions above and $r = [0\ 0\ s\ c]$, with $c^2+s^2=1$, is a quaternion that rotates about the z axis.

The product of a quaternion $[x\ y\ z\ w]$ with $r = [0\ 0\ s\ c]$ is $[xc+ys\ \ yc-xs\ \ zc+ws\ \ wc-zs]$. Choosing $c = w/\sqrt{w^2+z^2}$ and $s = -z/\sqrt{w^2+z^2}$ maximizes the real component to $\sqrt{w^2+z^2}$. Consequently, the best p is one that maximizes w^2+z^2. Only six values of p give essentially different results. These are $[0\ 0\ 0\ 1]$, $[1\ 1\ 1\ 1]/2$, $[1\ 1\ 1\ -1]/2$ and each of these times $[1\ 0\ 0\ 0]$. Summing the squares of the w and z components from the product of q with each of these and subtracting $^1/_2$ gives $\pm(w^2+z^2-{}^1/_2)$, $\pm(xz-wy)$ and $\pm(wx+yz)$. Choose the p corresponding to the largest positive value, and if the negative sign was used, post-multiply p by $[1\ 0\ 0\ 0]$.

This method for stabilizing the S decomposition is a greedy algorithm. It extends partial solutions at each stage by finding an optimal continuation, with no backtracking. There is no guarantee that this produces a global optimum—a locally inferior choice could possibly be warranted because it allows better choices further on that more than compensate. However, we can prove the following:

> **Proposition**: Given a sequence $\mathbf{S}_i$ of symmetric, positive definite matrices, none of which has a diagonalization with exactly two equal values, the greedy algorithm given above picks a sequence of rotation matrices $\mathbf{U}_i$ that minimizes the sum of the rotation angles between adjacent rotations.

The proof depends on two observations. First, the $\mathbf{S}_i$ with three equal values do not affect the sum; and second, the axis-permuting rotations p form a group. With this in mind, let $\langle p_i \rangle$ be the greedy sequence of permutations, and $\langle P_i \rangle$ the optimal sequence. Suppose now that some $p_k \neq P_k$. Then the discrepency $\delta = P_k^{-1}\ p_k$ is in the group, and can post-multiply every P_i, $i \geq k$ without increasing the angle sum. For P_k is replaced by p_k, which by definition of the greedy sequence gives the smallest angle possible at that step; and none of the other angles change, since $\delta^{-1} q_i^{-1} q_{i+1} \delta$ has the same angle as the original $q_i^{-1} q_{i+1}$. So $\langle p_i \rangle$ is also optimal.

With double values, however, some greedy sequences are not optimal. In mitigation, we point out that floating-point arithmetic stands between us and any reliable determination of equality of values, and that the additional freedom offered by equal values only causes the greedy algorithm to find solutions with smaller total rotation. Furthermore, the global optimization problem in the general case is a mixed-integer programming problem of the sort that is often NP-complete. (But we make no claims as to the status of this particular problem.)

Lest this extended discussion leave the wrong impression, we point out that diagonalization has not been necessary in our experience. The animations achieved by direct S interpolation look as good as those using the more elaborate procedure. (Also, the code required is much shorter than the discussion.) Since the developer of an animation system may choose not to introduce our new stretch primitive, however, we have offered a reasonable alternative.

Conclusions

With the assistance of Polar Decomposition, a non-singular 4×4 homogeneous matrix $\mathbf{M}$ can be factored into meaningful primitive components, as

$$\mathbf{M} = \mathbf{PTRNS},$$

where $\mathbf{P}$ is a simple perspective matrix, $\mathbf{T}$ is a translation matrix, $\mathbf{R}$ is a rotation matrix, $\mathbf{N}$ is $\pm\mathbf{I}$, and $\mathbf{S}$ is a symmetric positive definite stretch matrix. The stretch matrix can optionally be factored, though not uniquely, as $\mathbf{UKU}^\mathrm{T}$, where $\mathbf{U}$ is a rotation matrix and $\mathbf{K}$ is diagonal and positive. For a 4×3 affine matrix the perspective factor can be dropped; and for a 3×3 linear matrix, so can the translation. Also, $\mathbf{N}$ can be multiplied into $\mathbf{S}$ if desired.

Polar Decomposition produces factors $\mathbf{QS}$ which are unique, coordinate independent, and both simple and efficient to compute. The factors have a physical, visual interpretation not found with other decomposition methods. The **PTRNS** decomposition is useful for a variety of purposes, including matrix animation and interactive interfaces. It has the minor disadvantage that it does not directly represent shear.

Acknowledgements

Craig Reynolds first raised this challenge with Ken. John Gilbert, Eric Grosse, and Eugene Salamin shared their wisdom regarding numerical techniques and linear algebra. We thank Ed Shonberg for providing our French translation.

Appendix

Theorem: The Polar Decomposition factor $\mathbf{Q}$ is the closest possible orthogonal matrix to $\mathbf{M}$, with closeness measured using the Frobenius matrix norm. That is, $\mathbf{Q}$ satisfies the following conditions.

$$\text{Find } \mathbf{Q} \text{ minimizing } \|\mathbf{Q}-\mathbf{M}\|_\mathrm{F}^2$$
$$\text{subject to } \mathbf{Q}^\mathrm{T}\mathbf{Q} - \mathbf{I} = \mathbf{0},$$

where

$$\|\mathbf{Q}-\mathbf{M}\|_\mathrm{F}^2 = \sum_{i,j} (q_{ij}-m_{ij})^2 .$$

Proof: Though expressed in matrix terms, the proof simply requires finding the minimum of a quadratic function, which we learned to do in calculus by finding where the derivative is zero. We can express $\|\mathbf{M}\|_F^2$ as the diagonal sum—the trace—of $\mathbf{M}^T\mathbf{M}$, and incorporate the orthogonality constraint as a linear term using a symmetric Lagrange multiplier matrix $\mathbf{Y}$. So, as the reader can verify, we can differentiate

$$\text{trace}\,[\,(\mathbf{Q}-\mathbf{M})^T(\mathbf{Q}-\mathbf{M}) + (\mathbf{Q}^T\mathbf{Q} - \mathbf{I})\mathbf{Y}]$$

with respect to $\mathbf{Q}$ and equate to zero to obtain

$$2(\mathbf{Q}-\mathbf{M}) + 2\mathbf{QY} = \mathbf{0}$$

which simplifies to

$$\mathbf{Q}(\mathbf{I}+\mathbf{Y}) = \mathbf{M}.$$

Thus $\mathbf{M}$ will be factored as our desired $\mathbf{Q}$ times a symmetric $\mathbf{S} = \mathbf{I}+\mathbf{Y}$.

$$\mathbf{M} = \mathbf{QS}.$$

This factorization is the Polar Decomposition of $\mathbf{M}$. To use it we need to solve for $\mathbf{S}$ in terms of $\mathbf{M}$. Since $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, we must have

$$(\mathbf{MS}^{-1})^T(\mathbf{MS}^{-1}) = \mathbf{I}.$$

A symmetric $\mathbf{S}$ has a symmetric inverse, so this simplifies to

$$\mathbf{S}^{-1}\mathbf{M}^T\mathbf{M}\mathbf{S}^{-1} = \mathbf{I},$$

and finally to

$$\mathbf{S}^2 = \mathbf{M}^T\mathbf{M}.$$

Now, $\mathbf{M}^T\mathbf{M}$ is guaranteed to be symmetric and positive definite (or semi-definite if $\mathbf{M}$ is singular), and so there is a similarity transform that makes $\mathbf{M}^T\mathbf{M}$ diagonal, with positive (or zero) real entries. This gives the Spectral Decomposition of $\mathbf{S}^2$.

$$\mathbf{S}^2 = \mathbf{UKU}^T; \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}, \ \ \mathbf{K} = \begin{pmatrix} \kappa_1 & 0 & 0 \\ 0 & \kappa_2 & 0 \\ 0 & 0 & \kappa_3 \end{pmatrix}; \ \ \kappa_i \geq 0.$$

Taking either the positive or negative square root of each diagonal element of $\mathbf{K}$, we obtain eight candidates for $\mathbf{S}$,

$$\mathbf{U}\begin{pmatrix} \pm\sqrt{\kappa_1} & 0 & 0 \\ 0 & \pm\sqrt{\kappa_2} & 0 \\ 0 & 0 & \pm\sqrt{\kappa_3} \end{pmatrix}\mathbf{U}^T.$$

However, for $\mathbf{Q}$ to be a minimal solution, the second derivative, $2(\mathbf{I}+\mathbf{Y}) = 2\mathbf{S}$, of our function must be positive definite, which means only the positive square roots are allowed, and so $\mathbf{S}$ is uniquely determined. For any $\mathbf{M}$ which is non-singular, $\mathbf{Q}$ is also uniquely determined.

■

References

[Carlton 90] Carlton, Eloise H. and Shepard, Roger N. "Psychologically Simple Motions as Geodesic Paths," *Journal of Mathematical Psychology*, **34**(2), June 1990, 127–228

[Carnahan 69] Carnahan, Brice, Luther, H. A. and Wilkes, James O., *Applied Numerical Methods*, Wiley, 1969

[Foley 90] Foley, James D., van Dam, Andries, Feiner, Steven K., and Hughes, John F. *Computer Graphics: Principles and Practice, 2nd ed.*, Addison-Wesley, 1990

[Goldman 91] Goldman, Ronald N. "Recovering the Data from the Transformation Matrix," Gem VII.2, *Graphics Gems II*, Academic Press, 1991, 324–331

[Golub 89] Golub, Gene H., and Van Loan, Charles F. *Matrix Computations, 2nd ed.*, Johns Hopkins University Press, 1989

[Gomez 84] Gomez, Julian, "Twixt: a 3-d Animation System," *Proceedings of Eurographics '84*, Elsevier Science Publishers, 1984

[Greene 86] Greene, Ned, "Extracting Transformation Parameters from Transformation Matrices", NYIT, Personal communication

[Higham 86] Higham, Nicholas, "Computing the Polar Decomposition—With Applications", *SIAM J. Sci. and Stat. Comp.* **7**(4), October 1986, 1160-1174

[Higham 88] Higham, Nicholas, and Schreiber, Robert S. "Fast Polar Decomposition of An Arbitrary Matrix," Technical Report 88-942, October 1988, Department of Computer Science, Cornell University

[Press 88] Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T., *Numerical Recipes in C*, Cambridge University Press, 1988

[Raible 90] Raible, Eric. "Matrix Orthogonalization," *Graphics Gems*, Academic Press, 1990, p. 464.

[Shepard 84] Shepard, Roger N. "Ecological Constraints on Internal Representation: Resonant Kinematics of Perceiving, Imagining, Thinking, and Dreaming," *Psychological Review*, **91**(4), October 1984, 417–447

[Shoemake 85] Shoemake, Ken. "Animating Rotation with Quaternion Curves," *Proceedings of SIGGRAPH '85 (San Francisco, California, July 22–26, 1985)*, In *Computer Graphics* **19**(3), July 1985, 245–254

[Shoemake 91] Shoemake, Ken. "Quaternions and 4×4 Matrices," Gem VII.6, *Graphics Gems II*, Academic Press, 1991, 351–354

[Stern 83] Stern, G., "Bbop—A System for 3d Keyframe Figure Animation," SIGGRAPH '83 Course Notes, Introduction to Computer Animation, 1983

[Strang 86] Strang, Gilbert. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986

[Thomas 91] Thomas, Spencer W. "Decomposing a Matrix into Simple Transformations," Gem VII.1, *Graphics Gems II*, Academic Press, 1991, 320–323