Notes on the Polar Decomposition for animation

Steve Marschner

13 November 2014

Interpolation of transformations is a big topic in animation. The basic problem is: a given object should have an affine transformation $\mathbf{M}_0$ at time 0 and a second transformation $\mathbf{M}_1$ at time 1. The naïve solution is to interpolate elementwise:

$$\mathbf{M}_t = (1-t)\mathbf{M}_1 + t\mathbf{M}_2$$

but this has some problems, in particular with transformations that involve rotation. For example, if $\mathbf{M}_0$ and $\mathbf{M}_1$ are both rotations we might like for $\mathbf{M}_t$ to also be a rotation. But this won't happen; a simple experiment demonstrates that the object will shrink, then expand, as it moves from $\mathbf{M}_0$ to $\mathbf{M}_1$.

If we know a transformation is a rotation, there are good ways to deal with it: in 2D, compute the rotation angle and interpolate that; in 3D use quaternions, discussed in the next lecture. Many systems make their lives easy by restricting all transformations to the form

$$\mathbf{M} = \mathbf{TRS}$$

where $\mathbf{T}$ is a translation, $\mathbf{R}$ is a rotation (an orthogonal matrix), and $\mathbf{S}$ is an axis-aligned, possibly nonuniform, scale (a diagonal matrix). Then one simply interpolates the parts separately:

$$\begin{aligned}
\mathbf{T}_t &= \text{interp_linear}(\mathbf{T}_0, \mathbf{T}_1, t) \\
\mathbf{R}_t &= \text{interp_rotation}(\mathbf{R}_0, \mathbf{R}_1, t) \\
\mathbf{S}_t &= \text{interp_linear}(\mathbf{S}_0, \mathbf{S}_1, t) \\
\mathbf{M}_t &= \mathbf{T}_t\mathbf{R}_t\mathbf{S}_t
\end{aligned}$$

using an appropriate method for the rotation, and simply linearly interpolating the others.

But requiring the TRS form limits the transformations that can be represented. (Count degrees of freedom: 3 each for $\mathbf{T}$, $\mathbf{R}$, and $\mathbf{S}$ does not add up to the 12 needed for an arbitrary affine transformation.) And generally one needs to be able to handle arbitrary matrices that came from somewhere else, and there are various good reasons for wanting to use arbitrary matrices.

The point of these notes is that it's possible to take matrices apart, "factored" or "decomposed," into a TRS-like form, even if they did not start out that way, so that the benefits of interpolating rotation separately can be had without the constraint and added complexity of keeping the three parts separate. Once we solve this matrix decomposition problem, we interpolate the pieces separately and all is good.

When the matrix is actually a TRS matrix If someone had a translation, a rotation, and a scale, but multipilied them together in an attempt to foil our interpolation efforts, we don't have to work that hard to fix the problem. First of all, the product **RS** has no translation part, and multiplying on the left with **T** simply inserts the translation in the upper-right corner of the matrix, where we can just read it off. So the only difficulty is with the upper-left 3x3, which is the product of 3x3 rotation and scale matrices. (I'm goint to be a little loose with notation and call both the 4x4 matrix and the 3x3 matrix by the same name: in the 4x4 context it just gets a row and colum of zeros and a one in the bottom-right corner.)

A rotation matrix **R** has three orthonormal columns, $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$. Multiplying this by a scale on the right just changes the lengths of the columns:

$$\begin{bmatrix} | & | & | \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} s_1 & & \\ & s_2 & \\ & & s_3 \end{bmatrix} = \begin{bmatrix} | & | & | \\ s_1\mathbf{r}_1 & s_2\mathbf{r}_2 & s_3\mathbf{r}_3 \\ | & | & | \end{bmatrix}$$

They are still orthogonal (compute $(\mathbf{RS})^T(\mathbf{RS})$), so we can just write down the three lengths, normalize them, and we have **R** and **S** back.

The case where M is arbitrary But even though the TRS form is a common case, I want to be able to handle any matrix that comes along. Not every 3x3 matrix can be written as a rotation times a scale, but there is a convenient generalization. A linear algebra tool known as the polar decomposition guarantees that any square matrix can be written in the form

$$\mathbf{M} = \mathbf{RS}$$

where **S** is a *symmetric* matrix. The factors **R** and **S** are unique and depend in a nice and smooth way on **M**.

It turns out that symmetric matrices interpolate just fine by simple linear interpolation, just as scales do. In fact, a symmetric matrix is really a scale anyway—it's just that, unless it's diagonal, the scaling axes are not aligned with the coordinate axes. See Section 6.1.6 in the textbook for a discussion.

So the polar decomposition gives us a TRS form where the S stands for "symmetric" rather than "scale" (or still for "scale" but with the understanding it might not be axis-aligned). Combining the polar decomposition with the TRS interpolation approach above leads to a well-behaved way to work with arbitrary matrices:

$$\begin{aligned} [\mathbf{T}_0, \mathbf{R}_0, \mathbf{S}_0] &= \mathit{polar_decompose}(\mathbf{M}_0) \\ [\mathbf{T}_1, \mathbf{R}_1, \mathbf{S}_1] &= \mathit{polar_decompose}(\mathbf{M}_1) \\ \mathbf{T}_t &= \text{interp\_linear}(\mathbf{T}_0, \mathbf{T}_1, t) \\ \mathbf{R}_t &= \text{interp\_rotation}(\mathbf{R}_0, \mathbf{R}_1, t) \\ \mathbf{S}_t &= \text{interp\_linear}(\mathbf{S}_0, \mathbf{S}_1, t) \\ \mathbf{M}_t &= \mathbf{T}_t\mathbf{R}_t\mathbf{S}_t \end{aligned}$$

Actually computing the polar decomposition In 2D it is easy to compute the polar decomposition and it's instructive to look at that case. We have $\mathbf{M}$ and are looking for two matrices $\mathbf{R}$ and $\mathbf{S}$ that multiply to $\mathbf{M}$:

$$\mathbf{M} = \mathbf{RS} \quad \text{or,} \quad \mathbf{R}^T\mathbf{M} = \mathbf{S}$$

I will find $\mathbf{R}$ first: $\mathbf{R}$ is a rotation that has the property that $\mathbf{R}^T\mathbf{M}$ is a symmetric matrix. In 2D we can just write out in components:

$$\begin{aligned}\mathbf{R}^T\mathbf{M} &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \\ &= \begin{bmatrix} cm_{11} + sm_{21} & cm_{12} + sm_{22} \\ -sm_{11} + cm_{21} & -sm_{12} + cm_{22} \end{bmatrix}\end{aligned}$$

where $s = \sin\theta$ and $c = \cos\theta$. Think of turning a knob to adjust theta and watching for the product to become symmetric. This happens when

$$\begin{aligned} cm_{12} + sm_{22} &= -sm_{11} + cm_{21} \\ s(m_{11} + m_{22}) &= c(m_{21} - m_{12}) \\ \frac{s}{c} &= \tan\theta = \frac{m_{21} - m_{12}}{m_{11} + m_{22}} \\ \theta &= \tan^{-1}\left(\frac{m_{21} - m_{12}}{m_{11} + m_{22}}\right) \end{aligned}$$

In practice the `atan2` function is the right tool for this job; it is stable and works when $c = 0$. Or you can compute c and s without trigonometry just by normalizing the vector $[m_{11} + m_{22}, m_{21} - m_{12}]$.

Once you have $\mathbf{R}$ you can compute $\mathbf{S} = \mathbf{R}^T\mathbf{M}$ and you are home free!

Polar decomposition in more than 2D Of course in the 3D case this analysis doesn't work, but there is still a very simple iterative algorithm available, and you can find code to compute it in `egl.math.Matrix3`.