Homogeneous Coordinates

Tom Davis
tomrdavis@earthlink.net
http://www.geometer.org/mathcircles
October 25, 2000

The matrix projections and transformations in standard computer graphics libraries (such as OpenGL) provide enough flexibility for most people, but some developers require non-standard matrix operations, and need to know more about the mathematics, which turns out to be (real) projective geometry.

The calculations are simple, but may appear mysterious or even somewhat magical the first time through. What in the heck is the w coordinate? Where on earth did the formula for perspective come from? A complete de-mystification requires both a knowledge of the calculations (a left-brain task), and some good geometric mental images for your right brain. This article relates the calculations to a good mental model of projective geometry.

With a good mental picture, it's not hard to answer questions like these:

- Is there any way to discover the location of the viewing-position after transforming the scene with a matrix?
- I set my near clipping plane to 0.0000001 and my far clipping plane to 1000000.0. Why are my pictures so horrible?
- How can I construct a rotation matrix about an arbitrary axis?
- What's the formula for a transformation that does 2-point perspective (the OpenGL perspective() command is 3-point perspective).
- I need a shearing transformation. How do I do it?

1 Projective Geometry

Projective geometry is not the same as Euclidean geometry, but it is closely related. This article considers 2 and 3 dimensional projective geometry. Since (as in Euclidean geometry) it's easier to visualize things in 2 dimensions, we'll begin with 2D projective geometry. Besides, it's much easier to draw the pictures.

Here are two postulates from 2D Euclidean geometry:

- Every two points lie on a line.
- Every two lines lie on a point, unless the lines are parallel, in which case, they don't.

In 2D projective geometry, they are replaced by:

- Every two points lie on a line.
- Every two lines lie on a point.

That's basically the whole difference. How can we visualize a model for such a thing? The model must describe all the points, all the lines, what points are on what lines, and so on.

The easiest way is to take the points and lines from a standard 2D Euclidean plane and add stuff until the projective postulates are satisfied. The first problem is that the parallel lines don't meet. Lines that are almost parallel meet way out in the direction of the lines, so for parallel lines, add a single point for each possible direction and add it to all the parallel lines going that way. You can think of these points as being points at infinity—at the "ends" of the lines. Note that each line includes a single point at infinity—the north-south line doesn't have both a north and south point at infinity. If you "go to infinity" to the north and keep going, you will find yourself looping around from the south. Projective lines form loops.

Now take all the new points at infinity and add a single line at infinity going through all of them. It, too, forms a loop that can be imagined to wrap around the whole original Euclidean plane. These points and lines make up the projective plane.

You might make a mental picture as follows. For some small configuration of points and lines that you are considering, imagine a *really* large circle centered around them, so large that the part of the figure of interest is like a dot in its center. Now any parallel lines that go through that "dot" will hit the large circle very close together, at a point that depends only on their direction. Just imagine that all parallel lines hit the circle at that point. That is the "line at infinity".

Check the postulates. Two points in the Euclidean plane still determine a single projective line. One point in the plane and a point at infinity determine the projective line through the point and going in the given direction. Finally, the line at infinity passes through any two points at infinity.

How about lines? Two non-parallel lines in the Euclidean plane still meet in a point, and parallel lines have the same direction, so meet at the point at infinity in that direction. Every line on the original plane meets the line at infinity at the point at infinity corresponding to the line's direction.

Note: The projective postulates do not distinguish between points and lines in the sense that if you saw them written in a foreign language:

- Every two glorphs lie on a smynx,
- Every two smynxes lie on a glorph,

there is no way to figure out whether a smynx is a line and a glorph is a point or vice-versa. If you take any theorem in 2D projective geometry and replace "point" with "line" and "line" with "point", it makes a new theorem that is also true. This is called "duality"—see any text on projective geometry.

2 Homogeneous Coordinates

So we've got a nice mental picture—how do we assign coordinates and calculate with it? The answer is that every triple of real numbers $[x, y, w]$ except $[0.0, 0.0, 0.0]$ corresponds to a projective point. If w is non-zero, $[x, y, w]$ corresponds to the point $[x/w, y/w]$ in the original Euclidean plane; $[x, y, 0.0]$ corresponds to the point at infinity corresponding to the direction of the line passing through $[0.0, 0.0]$ and $[x, y]$. Generally, if α is any non-zero number, the homogeneous coordinates $[x, y, w]$ and $[\alpha x, \alpha y, \alpha w]$ represent the same point.

It's a bit disturbing that the same projective point can be represented in many different ways. For example, $[1, 2, 1]$, $[2, 4, 2]$, and $[-11, -22, -11]$ all refer to the Euclidean point $[1, 2]$. Don't panic, however; you've seen the same kind of thing before in third grade—the fractions $1/2$, $2/4$, and $55/110$ all represent the same number. Just as we usually use the fraction reduced to lowest terms, we usually use projective coordinates with w equal to 1.0 when that's possible.

Since projective points and lines are in some sense indistinguishable, it had better be possible to give line coordinates as sets of three numbers (with at least one non-zero). If the points are row vectors (as in the OpenGL), the lines will be column vectors (written with a "T" exponent that represents "transpose"), so $[a, b, c]^T$ represents a line. The point $P = [x, y, w]$ lies on the line $L = [a, b, c]^T$ if $ax + by + cw = 0$. In the Euclidean plane, the point $[x, y]$ can be written in projective coordinates as $[x, y, 1.0]$, so the condition becomes $ax + by + c = 0$—high-school algebra's equation for a line. The line passing through all the points at infinity has coordinates $[0.0, 0.0, 1.0]^T$. As with points, for any non-zero α, the line coordinates $[a, b, c]^T$ and $[\alpha a, \alpha b, \alpha c]^T$ represent the same line.

In matrix notation, the point P lies on the line L if and only if $PL = 0$. If we had chosen to represent lines as column vectors and points as row vectors, that would work fine, too. It has to work because points and lines are dual concepts.

3 Projective Transformations

Projective transformations transform (projective) points to points and (projective) lines to lines such that incidence is preserved. In other words, if T is a projective transformation and points P and Q lie on line L then $T(P)$ and $T(Q)$ lie on $T(L)$. Similarly, if lines L and M meet at point P, then the lines $T(L)$ and $T(M)$ meet at the point $T(P)$.

The reason projective transformations are so interesting is that if we use the model of the projective plane described above where we've simply added some stuff to the Euclidean plane, the projective transformations restriced to the Euclidean plane include all rotations, translations, non-zero scales, and shearing operations. This would be powerful enough, but if we don't restrict the transformations to the Euclidean plane, the projective transformations also include the standard projections, including the very important perspective projection.

Rotation, translation, scaling, shearing (and all combinations of them) map the line at infinity to itself, although the points on that line may be mapped to other points at infinity. For example, a rotation of 5 degrees maps each point at infinity corresponding to a direction to the point corresponding to the direction rotated 5 degrees. Pure translations preserve the directions, so a translation maps each point at infinity to itself.

The standard perspective transformation (with a 90° field of view, the eye at the origin, and looking down the y-axis) maps the origin to the point at infinity in the y-direction. The viewing trapezoid maps to a square.

Every non-singular 3×3 matrix (non-singular means that the matrix has an inverse) represents a projective transformation, and every projective transformation is represented by a non-singular 3×3 matrix. If M is such a transformation matrix and P is a projective point, then PM is the transformed point. If L is a line, $M^{-1}L$ represents the transformed line. It's easy to see why this works: if P lies on L, $PL = 0$, so $PMM^{-1}L = 0$, so $(PM)(M^{-1}L) = 0$. The matrix representation is not unique—as with points and lines, any constant multiple of a matrix represents the same projective transformation.

Combinations of transformations are represented by products of matrices; a rotation represented by matrix R followed by a translation (matrix T) is represented by the matrix RT.

A (2D) projective transformation is completely determined if you know the images of 4 independent points (or of 4 independent lines). This is easy to see, since a 3×3 matrix has nine numbers in it, but since any constant multiple represents the same transformation, there are basically 8 degrees of freedom. Each point transformation that you lock down eliminates 2 degrees of freedom, so the images of 4 points completely determine the transformation.

Let's look at a simple example of how this can be used by deriving from scratch the rotation matrix for a 45° rotation about the origin. The origin maps to itself, the points at infinity along the x and y axes map to points at infinity rotated 45°, and the point $[1, 1]$ maps to $[0, \sqrt{2}]$. See figure 4

If R is the unknown matrix:

$$
\begin{aligned}
[0,0,1]\,R &= k_1[0,0,1] \\
[1,0,0]\,R &= k_2[\sqrt{2},\sqrt{2},0] \\
[0,1,0]\,R &= k_3[-\sqrt{2},\sqrt{2},0] \\
[1,1,1]\,R &= k_4[0,\sqrt{2},1]
\end{aligned}
$$

The $k_1, \ldots, k_4$ can be any constants since any multiple of a projective point's coordinates represents the same projective point. M has basically 8 unknowns, so those 8 plus the 4 k_i's make 12. Each matrix equation represents 3 equations, so there is a system of 12 equations and 12 unknowns that can be solved. The computations may be ugly, but it's a straight-forward brute-force solution that gives the rotation matrix as any multiple of:

$$R = \begin{pmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ -\sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

There's nothing special about rotation. Every projective transformation matrix can be determined in the same brute-force manner starting from the images of 4 independent points.

We illustrate with the determination of the (2D) perspective transformation (see figure 3). We want to map the shaded trapezoidal area into the square with corners $[-1, -1]$ and $[1, 1]$. The unknown projection must satisfy:

$$\begin{aligned} [0, 0, 1]\,P &= k_1[0, 1, 0] \\ [0, n, 1]\,P &= k_2[0, -1, 1] \\ [f, f, 1]\,P &= k_3[-1, 1, 1] \\ [1, 0, 0]\,P &= k_4[1, 0, 0] \end{aligned}$$

The same brute-force calculation gives P as (any multiple of):

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -(f+n)/(f-n) & -1 \\ 0 & 2fn/(f-n) & 0 \end{pmatrix}$$

Three Dimensional Projective Space

3D projective space has a similar model. Take 3D Euclidean space, add points at infinity in every 3-dimensional direction, and add a plane at infinity going through the points. In this case there will also be an infinite number of lines at infinity as well. In 3D, points and planes are dual objects.

Projective transformations in 3 dimensions are exactly analogous. Points are represenated 4-tuple row vectors: $[x, y, z, w]$, and planes by column vectors: $[a, b, c, d]^T$. Any multiple of a point's coordinates represents the same projective point. A point P lies on a plane M if $PM = 0$. All 3D projective transformations are represented by 4×4 non-singular matrices.

In 3 dimensions, the images of 5 points (or planes) completely determine a projective transformation. (A 4×4 matrix has 16 numbers, but 15 degrees of freedom because any multiple represents the same transformation. Each point transformation that you nail down eliminates 3 degrees of freedom, so the images of 5 independent points completely determine the transformation.)

The brute-force solution has 20 equations and 20 unknowns (there will be 5 k_i's in addition to the 15 unknowns), and although the solution is time-consuming, it is straightforward.

The calculation can be simplified. Suppose you want a transformation that takes P_1 to

Q_1, ..., and P_5 to Q_5. Let

$$\begin{aligned} I_1 &= [1,0,0,0] \\ I_2 &= [0,1,0,0] \\ I_3 &= [0,0,1,0] \\ I_4 &= [0,0,0,1] \\ I_5 &= [1,1,1,1] \end{aligned}$$

Find the transformation P that takes P_i to I_i and the transformation Q that takes Q_i to I_i. Because of all the zeroes, these are much easier to work out. The transformation you want is PQ^{-1}.

4 Using the Mental Model

Let's conclude by taking another look at the questions from the first section.

Perspective projections map the eye point to infinity, so if you know the projection matrix and want to find the eye point, find the point that maps via the projection to infinity in the z-direction.

If you're wondering about the bizarre effects of widely spaced near and far clipping planes, look at how much streching occurs between the origin and .0000001.

Construction of the three projections is simply a matter of listing 5 independent points and their images and calculating the matrix by brute force.

The more ways you have to look at a mathematical concept, the better you will understand it. Perhaps the mental images here may provide you with some additional insight.