Homogeneous Coordinates

*August 6, 2006*

In computer graphics we usually use homogeneous coordinates to represent 3D points. Each coordinate has four dimensions: the normal three plus a “1”. Matrices are 4×4, and they can encapsulate not only rotations and scales, but also translations and perspective. I had never really understood how this worked. Reading a Jim Blinn article about an unrelated subject recently gave me an insight into homogeneous math, and this article attempts to explain this insight. Since thinking in 4D is hard, we’ll pretend that our problem space is in 2D and homogeneous coordinates extends that to three dimensions.

Shear

With 2D matrices we can take any set of points and rotate them around the origin, or scale them toward or away from the origin. We can also shear them. A *shear* is a movement in one direction that’s proportional to the distance away from the origin in the other dimension. So a point at ***Y = 5*** (with any X value) might move to the right 3 units, and a point at ***Y = 10*** (again with any X value) would move to the right 6 units. Being twice as far from the X axis doubles your horizontal movement. If you have Java installed, the applet to the right will show you this type of shearing. Click and drag the mouse over the applet to change the amount of shear. (The applet was created with [Processing](http://processing.org/).)

Notice that the Y coordinates of points don’t change. It’s not a rotation. The object is distorted, pulled horizontally by different amounts. We almost never use shearing in graphics, which is why I wanted to explain it here because it will be used below to translate points.

Translation

A 2D matrix is incapable of *translation*, which is moving all points the same direction and distance. You can’t write a 2D matrix to move all points up 2 units and right 3 units. That’s because the matrix is multiplied by each point, and a multiplication can’t act like an addition. You also can’t do other fancy things, like multiply each point’s Y value by its X value (a projection).

So if your pipeline were made of 2D matrices, you’d have to perform translations some other way. If you wanted to take a set of points, rotate it, translate it, then rotate it again, you’d be forced to write it like this:

***P' = R2 × (T + (R1 × P))***

Each point would go through a multiplication, then an addition, then another multiplication. This is unfortunate, because if it weren’t for the translation you could just multiply the two rotation matrices together ahead of time (because matrix multiplication is associative) and each point would just be multiplied by a single matrix, saving a lot of work at each vertex of a potentially very large model.

That’s where homogeneous coordinates come in. They allow you to express a translation as a multiplication, so that all matrices can be collapsed up front. But how to translate by multiplying? The key is to visualize this in 3D space. Your problem space is the ground, going horizontally in the two dimensions (X and Y). The new homogeneous dimension (W) is vertical. Assign all your points a W value of 1. You can visualize all your points hovering one unit above the ground. All 2D matrices (rotations and scales) are extended to 3D, with a 1 in the lower-right cell, meaning that everything just works in this ***W = 1*** plane as it did before.

Now remember how shears work: points are moved in one dimension by their distance along another dimension times a constant. In our case, the “distance along another dimension” is the distance along W, which is always 1. So we can move all points toward the positive X axis 3 units by shearing. The formula for each point is basically ***Px' = Px + Tx × Pw***. Visualize the whole 3D space shearing toward positive X as W goes up. Normally this would distort a full 3D object, but our object lies entirely at ***W = 1***, so it doesn’t get distorted at all. Drag in the applet at the right to see this in 3D.

Mathematically, a 2D sheer looks like this (column-vector notation):

|  |  |
| --- | --- |
| 1 | C |
| 0 | 1 |

This means, “To each X, add Y×C.” The homogeneous 2D version for translation looks like this:

|  |  |  |
| --- | --- | --- |
| 1 | 0 | Tx |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

This means the same thing: “To each X, add W×Tx.” Since W is always 1, we’re just adding Tx to each X. We can add to both X and Y at the same time:

|  |  |  |
| --- | --- | --- |
| 1 | 0 | Tx |
| 0 | 1 | Ty |
| 0 | 0 | 1 |

The simpler way of understanding this (and probably the way most people understand it) is just that Tx is being added to the final X value after it has gone through the normal (2D in this case) dot product. Looking at it as a sheer is interesting because it prepares us for understanding perspective.

Perspective

By definition, you can take a homogeneous vector and multiply it by a constant and get the same point. So again in our 2D problem space (3D homogeneous vectors), this vector:

|  |
| --- |
| 4 |
| 5 |
| 6 |

and this vector:

|  |
| --- |
| 8 |
| 10 |
| 12 |

are equivalent. We start with all our models at the ***W = 1*** plane, and in our final space we’re on that plane too, but in between we can use the above equivalence to perform divisions. In a perspective view, objects that are further away look smaller in the final image. “Further away” means with a greater Z value (in 3D). “Smaller” means scaled down toward the origin. So what we want to do is divide X and Y by Z. We can’t do that with a normal matrix multiply, but with homogeneous coordinates we can take advantage of the fact that multiplying W by a number is the same as dividing X, Y, and Z by that same number.

So back to 2D. Let’s pretend that Y is our “depth”, and our image is only one-dimensional (X). We want to make X coordinates smaller as Y gets further away. In other words, we want to divide X by Y. We can accomplish the same thing by multiplying W by Y. (The latter will have the side-effect of dividing Y by itself, resulting in a useless 1 in all Y coordinates. We can fix that by putting W (*i.e.,* 1) into Y, effectively swapping the two and ending up with 1/Y in the Y coordinate. We’ll ignore that here since the primary purpose of doing that is for depth-testing.) This matrix will do the trick:

|  |  |  |
| --- | --- | --- |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

This isn’t actually multiplying Y by W, since you can’t do that with a matrix multiplication. It’s just moving Y to W (bottom-middle cell) and throwing away the old W value (bottom-right cell). When we later divide X and Y by W to get our non-homogeneous 2D coordinate, we’ll effectively divide X by Y.

How do we visualize this the way we did with translation? Firstly, since multiples of a point are equivalent, you can imagine a line going through the origin. All points on this line are equivalent. Normally we keep ***W = 1***, so you can now imagine that plane and your line intersecting at a point. That’s the point we normally deal with.

The goal is to divide X by Y. Again we’re doing a shear, except this time shearing W by Y. (Earlier we were shearing X and Y by W.) Visualize your model in 2D sitting on the ***W = 1*** plane. It’s easiest to do this in two steps. First get rid of W (the lower-right cell in the above matrix). That just drops the model down to ***W = 0***. Now shear W (upwards) as Y increases. Your model, which sits in a plane, has just been tilted up 45° so that it passes through the X axis and goes up where Y is positive.

At the end of the pipeline, all points will have their coordinates normalized so that ***W = 1***. This involves dividing X, Y, and W by W. Or, visually, moving each point toward or away from the origin (along that line you were imagining) so that it ends up at ***W = 1***. Since our model was in a 45° plane going through the X axis, all Y coordinates will end up as 1 (since the plane intersects ***W = 1*** at ***Y = 1***). But X coordinates will get squeezed toward the middle the further away from the X axis the point was, resulting in the perspective effect we wanted.

(I tried to visualize this with an applet the way I did above with translation, but it was too visually confusing.)

[Back to graphics](http://www.teamten.com/lawrence/graphics/)