Matrix Visualisation

Neil Edelman

2000-06

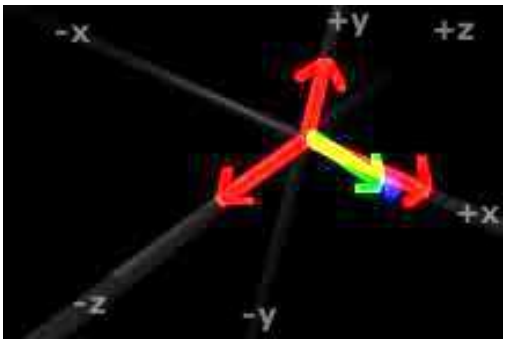

Figure 1: A point is defined in 3D via three components; these usually represent the distances along each axis. Often, the handedness or labels are different; standard handedness is often abandoned to have z decreasing away from the screen, as it is here. OpenGL has z the other way preserving right-handedness, a system I prefer. Here, x, y, and z distances are marked in red. These denote a vector, in green, which points to the blue point.

With the advent of 3D Accelerators on every machine, this is a moot point since everything is uploaded to the Card anyway. But it's interesting in a purely acedemic context.

A 3D engine converts points in a 3D world to points on the screen. The point (x, y, z) in World Space is transformed into (x', y', z') in View Space (Camera Space, Eye Space, et cetera.) It is then projected to the 2D screen, usually by perspective projection in 3D engines. The transformation to View Space can be seen as a matrix,

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} x_A & x_B & x_C & x_D \\ y_A & y_B & y_C & y_D \\ z_A & z_B & z_C & z_D \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

The big thing is the matrix; it's the coördinates of the camera. It's not an esoteric, abstract thing; it's just a collection of simple dot products grouped together for convenience. It's all very neat and good. The computer just adds and multiplies, a matrix is a human construction to organise the information. We are free to let any isomorphism represent the transformation. Let's deconstruct the matrix as a (relatively) simple set of three plane equations.

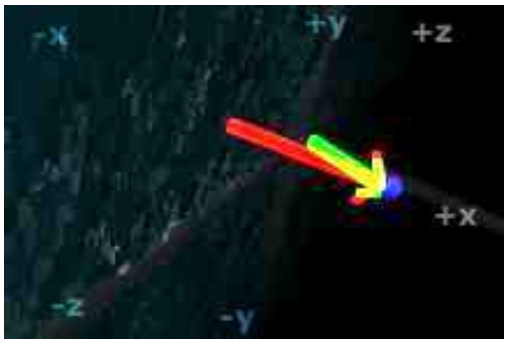

Figure 2: Instead of a distance along the x axis, the vector's (green) x component (red) can be though of as the distance from the point (blue) to the plane encompassing the y and z axes.

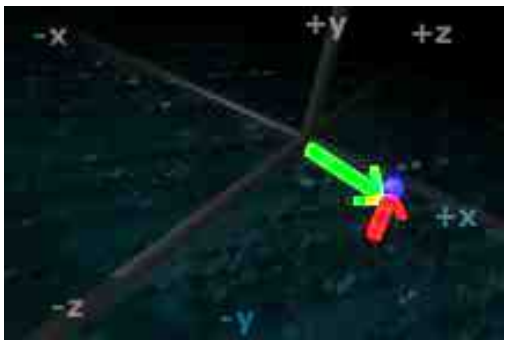

Figure 3: Instead of a distance along the y axis, the vector's (green) y component (red) can be though of as the distance from the point (blue) to the plane encompassing the x and z axes.

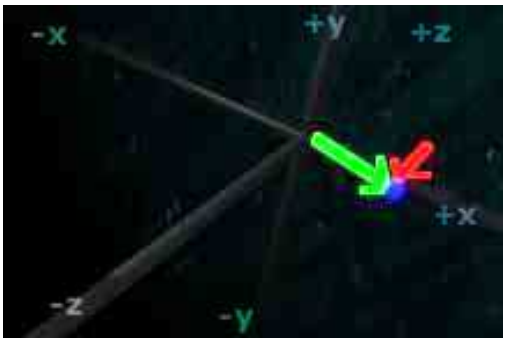

Figure 4: Instead of a distance along the z axis, the vector's (green) z component (red) can be though of as the distance from the point (blue) to the plane encompassing the x and y axes.

Actually, this is easy; you just separate the rows, eg $\begin{pmatrix} x' \end{pmatrix} = \begin{pmatrix} x_A & x_B & x_C & x_D \end{pmatrix} \cdot \begin{pmatrix} y & y & z & 1 \end{pmatrix}$ becomes, $x' = x_A x + x_B y + x_C z + x_D$. If $x' = 0$, this is the plane equation for the camera's x; it's locus of all points whose distance is equal to zero. The coefficients expressed as a vector, (x_A, x_B, x_C), will be the normal to the plane.

Plane with x-axis normal: $x' = x_A x + x_B y + x_C z + x_D$,
plane with y-axis normal: $y' = y_A x + y_B y + y_C z + y_D$,
plane with z-axis normal: $z' = z_A x + z_B y + z_C z + z_D$.

Having calculated three planes, it becomes a simple matter of plugging in the coördinates of a point to determine its distance to the plane. Substituting a point at (x, y, z) into the above equations will give three answers, one for each plane. These form a vector, (x', y', z'), that is the coordinates of the point with respect to the axes defined by these planes. This is an isomorphism to the matrix transform that doesn't use matrices, only simple geometry.

This might be useful to calculate z values and skip calculating the x and y if z doesn't fall between clipping distances, or maybe just never finding z in a parallel projection. I'm sure this is how it works in graphics accelerators.