A Trip Down The Graphics Pipeline: The Homogeneous Perspective Transform

James F. Blinn, California Institute of Technology

The perspective transform basically turns space inside out. Most people don't have an intuitive feel for what this does to a shape, so I will try to provide one.

Most of the transformations used in computer graphics are pretty boring: rotations, scales, translations, and even shears. But for pure weirdness, you can't top the perspective transformation. Most people don't have an intuitive feel for what this transform does to a shape, so in this month's column I will try to provide one. This has practical applications in selecting near and far clipping planes to avoid depth resolution problems with many types of rendering algorithms. But to understand perspective, we will first have to review some interesting topological properties of the space represented by homogeneous coordinates.

Homogeneous coordinate representation

First, let's review how homogeneous coordinates work. We represent a 3D point in what I'll call real space, (X, Y, Z), as a four-element vector (x, y, z, w) in homogeneous space. To help distinguish real space from homogeneous space, I'll write real-space coordinates in capitals and homogeneous-space coordinates in lower case. The relation between them is

$$(X, Y, Z) = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

(Notice that many different homogeneous space vectors can represent the same real-space point.) Even though we might have points represented as a four-vector during some of the processing, we ultimately will need to convert back to real space by dividing out the w component. Transformations consist of multiplying a point by a 4×4 matrix. By manipulating various elements in the matrix, we can rotate, scale, shear, translate, and (our favorite) perform a perspective projection.

Moebius space

One of the interesting features of homogeneous coordinates is that they provide a computationally tractable way to represent points that are infinitely far away. For example, consider the point $(1, 0, 0, w)$. This represents the real-space point $(1/w, 0, 0)$. If we make the value of w smaller and smaller, then the point's X coordinate gets bigger. Ultimately, if w reaches 0, we have a point with an infinitely large X coordinate. The homogeneous formulation, however, is simply $(1, 0, 0, 0)$; these values are perfectly ordinary and easy to deal with inside a computer. All points with $w = 0$ are at infinity and form what is called the *plane at infinity*. Of course, we can't convert these values back to real-space points because we can't divide the coordinates by w, but we can still perform geometrical calculations with them if we leave them in the form $(x, y, z, 0)$.

Now let's look at what happens if w becomes negative, say, -0.1. The real space analog of $(1, 0, 0, w)$ becomes $(-10, 0, 0)$. In other words, the point "wraps around infinity" and comes back in from the negative direction. In this scheme, therefore, it makes sense to say that the point at infinity in the positive direction, $(1, 0, 0, 0)$, is the same as the point at infinity in the negative direction, $(-1, 0, 0, 0)$. Points at the top of the universe wrap around to the bottom; points at the right wrap around to the left.

At first this overflow property might tempt you to say that the space formed by homogeneous coordinates is a toroidal universe, like the universe formed by overflow of integer coordinates. But the homogeneous universe is different. Consider Figure 1, where I labeled several infinite points in the XY plane with their negative counterparts. When you tie point A with A′, B with B′, C with C′, and so forth, you get a twist in the fabric of space when it connects around infinity. The shape formed when you do this with the 2D plane of Figure 1 is

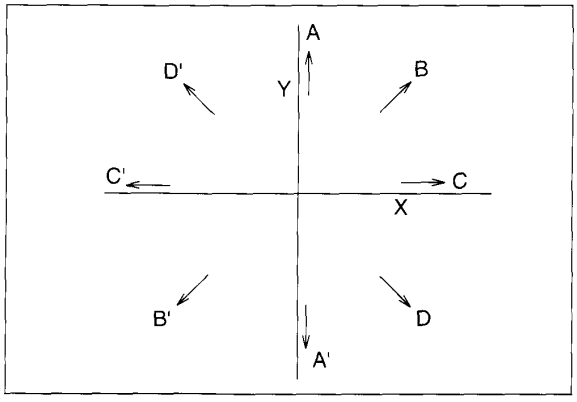


Figure 1. Points at infinity.

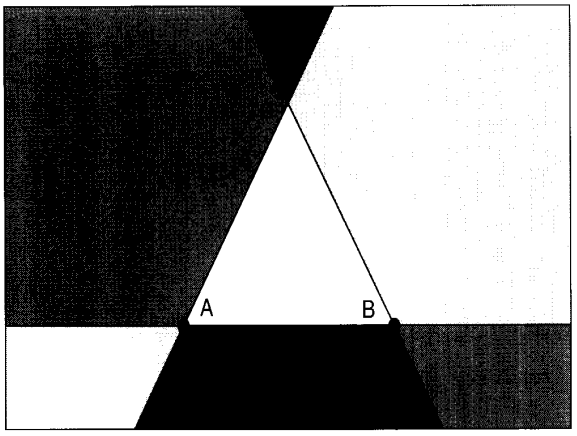


Figure 2. Four triangles. Three of them straddle the plane at infinity.

called a *projective plane.* This, together with the *torus* and the *Klein bottle,* is one of the three topological ways to tie 2D space together. Several pictures illustrating these shapes appear in Steven Barr's really fun book *Experiments in Topology*[1] and in Rich Riesenfeld's article, "Homogeneous Coordinates and Projective Planes in Computer Graphics."[2]

It's important to keep this Moebius twist in mind when trying to understand the homogeneous perspective transform, since that transform does, indeed, move points through infinity.

To better understand how this Moebius twist affects homogeneous shapes in real space, look at Figure 2. You should see four triangles. See them? Well, the one in the center is obvious. The other three straddle the plane at infinity, and I've colored them in three different shades of gray. This diagram also illustrates that there are two possible line segments connecting any two points. Look at points A and B. One line segment, called an *internal* segment, starts at A and moves to the right to point B. The other segment starts at B, moves to the right, wraps around infinity, comes in from the left, and terminates at A. This is called an *external* segment. In the homogeneous universe, both these segments are equally valid. Finally, note how the Moebius twist along the external segment keeps the proper triangle colors connected.

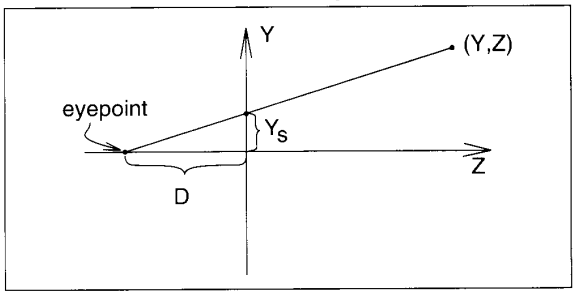
Coordinate systems

There are four coordinate spaces that I'm going to discuss here. Let me give explicit names to each space and to a canonical point in that space.

1. 3D: Real space before the perspective transform, (X, Y, Z)
2. 3DH: Homogeneous space before perspective. We usually form this by appending a $w = 1$ coordinate to the end of each real-space vector, so a point is $(X, Y, Z, 1)$. Thus, objects in this space typically lie in the $w = 1$ hyperplane of 4D space.
3. 3DHP: Homogeneous space after perspective. Here objects no longer lie in the $w = 1$ hyperplane and, thus, have the general coordinates (x_s, y_s, z_s, w_s).
4. 3DP: Real space after perspective. This is just the above space with the w_s coordinate divided out. This space is variously called *screen space* or *perspective space.* A point has the coordinates (X_s, Y_s, Z_s).

Whenever I draw a diagram in this article, I will label it with these coordinate system names. The most interesting such diagrams directly compare the shape of an object in 3D space to its distortion into 3DP space. The main purpose of this article is to provide an intuitive feel for what that distortion is.

Figure 3. Perspective geometry.



Simple perspective

It turns out that the version of the perspective transform that's easiest to understand is not the version typically used in applications. Nevertheless, to help us understand the transform, I'll start with this simple version, and I'll relate it to the more useful form later.

Let's derive a simple perspective transform from the geometry in Figure 3, where the eye is a distance D in front of the origin. For an arbitrary point (X, Y, Z) in space, we want to find its perspective projection onto the XY plane, which we will call (X_s, Y_s). Using similar triangles, we get

$$\frac{X_s}{D} = \frac{X}{Z + D}$$

and

$$\frac{Y_s}{D} = \frac{Y}{Z + D}$$

so

$$X_s = \frac{X}{(Z/D + 1)}$$

and

$$Y_s = \frac{Y}{(Z/D + 1)}$$

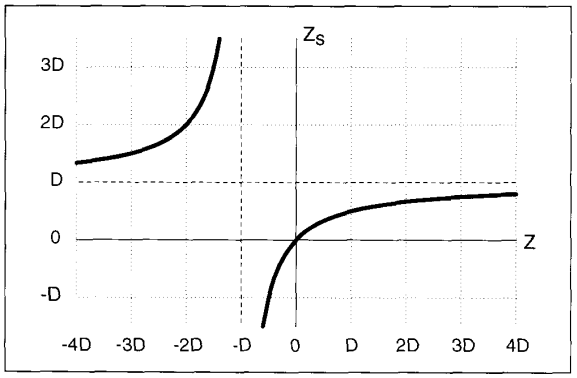
This by itself is not a linear transform. We can, however, piggyback the division onto the homogeneous division by defining

$$X_s = \frac{X}{(Z/D + 1)} = \frac{x_s}{w_s}$$

and

$$Y_s = \frac{Y}{(Z/D + 1)} = \frac{y_s}{w_s}$$

Figure 4. Postperspective Z versus preperspective Z.



We can define Z_s symmetrically as

$$Z_s = \frac{Z}{(Z/D + 1)} = \frac{z_s}{w_s}$$

so

$$(x_s, y_s, z_s, w_s) = (X, Y, Z, Z/D + 1)$$

Now we can express this as a homogeneous matrix multiplication

$$(x_s, y_s, z_s, w_s) = (X, Y, Z, 1)\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/D \\ 0 & 0 & 0 & 1 \end{bmatrix} = (X, Y, Z, 1)\,\boldsymbol{P}$$

In the homogeneous notation scheme, any column vector represents a plane. In particular, each column of a transformation matrix represents a plane. The right column of the matrix, $(0, 0, 1/D, 1)^t$, represents the plane $Z = -D$. This plane contains all the points that the matrix will map to $w = 0$. That is, $Z = -D$ is the plane that gets transformed to the plane at infinity. In general, you will combine a perspective matrix with other viewing and modeling transformations, making the right-hand column have four arbitrary values. But whatever its contents, the right-hand column, interpreted as a plane, will be the plane containing the eyepoint and perpendicular to the line of sight.

What does it all mean?

We've designed the matrix to generate the correct values for X_s and Y_s. But it's the Z_s values it generates that are interesting. Let's play around with these values to see what happens to 3D shapes subjected to this transform.

Cherchez la point

Let's start by looking at what happens to the Z coordinate. It transforms according to the equation

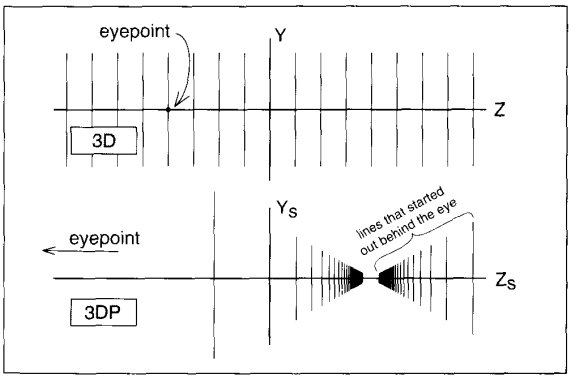


Figure 5. The picket fence in perspective.

$$Z_s = \frac{Z}{Z/D + 1}$$

A plot of this function appears in Figure 4. Note that when Z is infinite, $Z_s = D$. The transformation of certain key points is particularly illuminating:

$$(X, Y, 0, 1)\boldsymbol{P} = (X, Y, 0, 1)$$

This means that points in the $z = 0$ plane—the plane of the screen—don't move.

$$(0, 0, -D, 1)\boldsymbol{P} = (0, 0, -D, 0)$$

This means that the eyepoint moves to infinity.

$$(0, 0, D, 0)\boldsymbol{P} = (0, 0, D, 1)$$

This means that a point infinitely far forward becomes a local point.

Picket fence

Another way to look at this is to see what happens to a bunch of equally spaced parallel lines perpendicular to the line of sight. As shown in Figure 5, they transform to parallel lines of different lengths and unequal spacing.

Homogeneous space interpretation

In homogeneous space, the perspective transform is a simple shear in the w direction. Figure 6 shows this process for the wz slice of homogeneous space. Points in 3DH space (with $w = 1$) shear up and down, forming 3DHP space. Then they project back onto $w = 1$ to give 3DP. Note how the eyepoint, the $Z = 0$ point, and the $Z = \infty$ point transform.

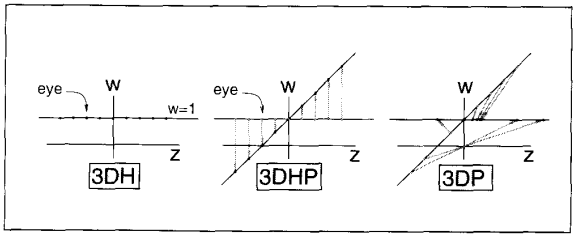


Figure 6. The homogeneous view of the perspective transform.

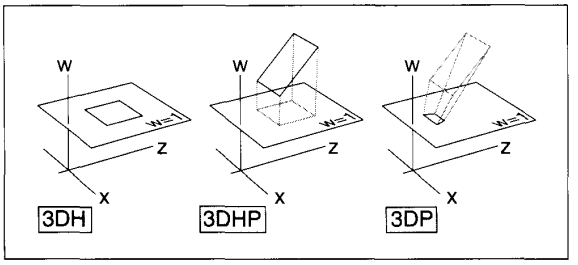


Figure 7. The homogeneous view of a square's transform.

Now, how about X and Y? Look at Figure 7. Here I just plotted x, z, and w; the y coordinate operates similarly to x. We start with a square, seen edge on. Perspective multiplication shears the square in w. Dividing out the w distorts the square into a trapezoid.

In Figure 8 you see the same thing applied to a cube, turning it into the frustum of a pyramid. Note that this diagram shows the real 3D space transformed to the real 3DP space.

Region mapping

The ultimate understanding of this transformation comes from Figure 9. Figure 9a shows several regions in the YZ slice of 3D space, and Figure 9b shows how they get distorted in 3DP space. The Y axis (at $Z = 0$) doesn't move. Region B, stretching from the screen to infinity, maps to the finite rectangular region B′. The point of the viewing pyramid (the eyepoint) moves to infinity, so the triangular region A transforms to the infinite rectangle A′. Regions H, J, E, and D bend appropriately. The interesting thing is to look at stuff that started out behind the eye: regions C, G, and F. These regions "wrap around infinity" and come back in front of the old infinite plane. Notice that, reading from the top down, regions G, C, and F map into regions G′, C′, and F′ reading bottom up. This is because of the Moebius property of homogeneous coordinates.

A better matrix

When planning a scene, you generally specify the view in terms of camera location and viewing direction. You then build up a viewing transformation by translating the camera position to the origin and rotating the viewing direction to point down the Z axis. Then, in order to use the transform we derived here, you must translate the eyepoint D units backward in Z to place the eyepoint at $Z = -D$. The following more convenient primitive perspective matrix has this move built in:

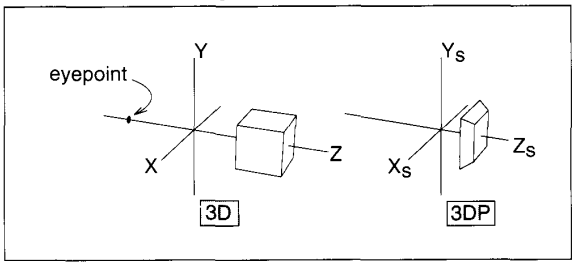
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -D & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/D \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/D \\ 0 & 0 & -D & 0 \end{bmatrix}$$

The matrix on the right performs eye-at-origin perspective. Note that its rightmost column, $(0, 0, 1/D, 0)^t$, is the plane $z = 0$ that will now transform to the plane at infinity.

It's also convenient to specify the field of view in terms of the angle, *fov*, at the apex of the view pyramid. A screen stretching from +1 to −1 viewed from distance D has

$$D = \cot\left(\frac{fov}{2}\right)$$

Figure 8. The perspective transform of a cube.



The net effect of our new perspective matrix is to map the 3D point (X, Y, Z) to the 3DH point

$$\left(D\frac{X}{Z}, D\frac{Y}{Z}, D\frac{Z-D}{Z}\right) = (X_s, Y_s, Z_s)$$

Note that the effect of a change in the field of view on X_s and Y_s is just a simple scaling. In other words, changing the field of view doesn't change the shape of any objects on the screen; it just scales the image uniformly so that more or less of the environment fits within the screen boundaries.

Depth information

Many rendering algorithms use the Z_s values that come out of this transformation to do depth comparisons, so it's good to have some idea of the range of values this transformation can have. Let's see what happens to a few key Z coordinate values under this new transform. Z coordinates that start at infinity map to $Z_s = D$, and Z coordinates that start out at $+D$ map to $Z_s = 0$. Any points that start out closer to the eye than $Z = +D$ map to some negative value of Z_s. Depending on how near the eye it is, a point can map to a rather enormous negative Z_s value. The eye itself, of course, maps to minus infinity. Objects very close to the eye might generate a divide error if the quotient of z_s and w_s is too big for a floating-point number. This is a nuisance. How can we avoid it? Typically we use a clipping plane to remove all objects nearer than a certain distance.

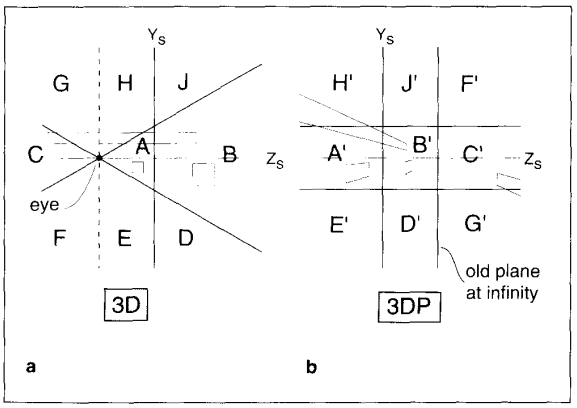
But close objects that generate divide errors are not the biggest problem. The biggest problem is distant objects. What happens to the resolution in Z for objects that are far from the eye? Let's look at a numerical example. Suppose the field of view is about 53 degrees, giving a value of $D = 2$. Suppose that two points on a viewed object are at distances $Z = 500$ and $Z = 501$. The perspective transform moves these to the positions

$$Z_s = 2\left(\frac{500-2}{500}\right) = 1.992$$

and

$$Z_s = 2\left(\frac{501-2}{501}\right) = 1.992016$$

Figure 9. Which regions map to where.



This really starts to push the resolvable limits of single-precision floating-point numbers. And using double-precision numbers is the cowardly approach.

We can do everything in single precision if we know the approximate range of Z values (relative to the eye) for objects in our scene. We just scale and translate the postperspective Z_s values to spread more uniformly over the range 0 to 1. The easiest way to specify this scale and translation is in terms of two Z values in preperspective (3D) space. We'll call these Z_n for the near value and Z_f for the far value. We then calculate the scale and transformation that maps these values to 0 and 1, respectively, in 3DP space. A scale and translation in Z will only modify the third column of the matrix, so let's solve for the matrix elements that do the desired mapping. A point on the near plane maps to

$$(0, 0, Z_n, 1)\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & 1/D \\ 0 & 0 & B & 0 \end{bmatrix} = (0, 0, AZ_n + B, Z_n/D)$$

We want this to map to $Z_s = 0$, so we must have

$$B = -AZ_n$$

Next, a point on the far plane maps to

$$(0, 0, Z_f, 1)\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & 1/D \\ 0 & 0 & -AZ_n & 0 \end{bmatrix} = (0, 0, A(Z_f - Z_n), Z_f/D)$$

We want this to map to $Z_s = 1$, so its z and w components must be equal. This gives us

$$A = \frac{Z_f}{D(Z_f - Z_n)}$$

The resultant matrix maps Z_n to 0, Z_f to 1, and $Z = \infty$ to $Z_f/(Z_f - Z_n)$.

In my column "Nested Transformations and Blobby Man" (*CG&A*, October 1987, pp. 61-66), I showed this same matrix, but I wrote it a bit differently. To get to the form used in that article, just multiply the above matrix by the constant factor $D \sin(fov/2) = \cos(fov/2)$ to get

$$\begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & Q & s \\ 0 & 0 & -QZ_n & 0 \end{bmatrix}$$

where

$$s = \sin(fov/2)$$
$$c = \cos(fov/2)$$
$$Q = \frac{s}{1 - Z_n/Z_f}$$

There are two practical reasons for this way of specifying the matrix. First, if the user specifies a field of view of zero degrees, the program will not blow up by attempting to calculate an infinite cotangent. (Admittedly, though, the matrix will be a bit weird.) Second, if the user wants to use an infinite value for Z_f (a perfectly reasonable thing to do), the expression for Q above reduces nicely to $Q = s$.

Now that we have the handles Z_n and Z_f to play with, let's see how they solve our resolution problem. If we use the fairly loose bounds around our object of $Z_n = 400$ and $Z_f = 600$, the perspective matrix gives us

$$Z_s = \frac{3Z - 1{,}200}{Z}$$

You can check that $Z = 400$ maps to $Z_s = 0$ and $Z = 600$ maps to $Z_s = 1$. The Z values of 500 and 501 map respectively into Z_s values of 0.6 and 0.6048, which are much more readily distinguishable.

The location of Z_n and Z_f are usually associated with near and far clipping planes. The problem is that users typically don't want anything clipped off in the near and far directions, so they make Z_n very small and Z_f very large. Making Z_f large, or even infinite, doesn't really cause problems, but making Z_n small does. It basically defeats the purpose of our new formulation, cramming the objects in the scene into a depth range very close to $Z_s = 1$. The depth resolution available to the rendering algorithm is then completely lost. I have seen a lot of people struggle with this. My advice is to place Z_n as far away as you can manage. In fact, for some projects I have had to animate the value of Z_n to track an object as it flies around on the screen.

Clipping implications

In an earlier article ("A Trip Down the Graphics Pipeline: Line Clipping," *CG&A*, January 1991, pp. 98-105), I wrote about a line-clipping algorithm that operated with the perspective transformation we derived here. Now that we have a better understanding of homogeneous perspective, I want to clear up a few statements made in that article.

First, near and far clipping are optional. We must still, however, specify a value for Z_n and Z_f. After all, *some* value of Z is going to map to $Z_s = 0$, and *some* other value of Z is going to map to $Z_s = 1$. We can't do anything about that. But it's not

necessary to *clip* to these planes. You do, however, need to be prepared to detect and avoid overflow for objects that come close to the eye and produce large negative values of Z_s, but this is actually rather rare. You might as well do the clipping, though. The overhead of clipping at Z_n and Z_f as described in the earlier column is pretty negligible.

Second, we can clip properly *after* doing the w division, that is, in 3DP space rather than in 3DHP space. I don't recommend it, but it's *possible*. The reason we usually think it's impossible is as follows. Consider a line that has one endpoint in front of the eye and one endpoint behind the eye. That is, the line runs off the top of the screen, as in one of the long edges of the wide rectangle in Figure 9. After the perspective multiplication, the endpoint behind the eye will have a negative value for w_s. If you clip in 3DHP space, the endpoint will have $y_s > w_s$, and the line will be clipped properly. When you look at the line, you will see it disappear off the top of the screen. Now suppose you did the homogeneous division before clipping. The point behind the eye will have wrapped around infinity and may reappear with Y coordinates on the visible portion of the screen, generating an external line segment. This situation has traditionally been thought to be indistinguishable from an internal segment with both endpoints visible in front of the eye. There is a difference, however. The behind-the-eye point has wrapped around infinity and has a Z coordinate greater than $Z_f/(Z_f - Z_n)$. To determine if a line segment that has two visible endpoints is internal (requiring no clipping) or external (requiring clipping), just test if the Z coordinates of its two endpoints straddle $Z_f/(Z_f - Z_n)$. This is more trouble than it's worth perhaps, but it's possible.

Summary

The perspective transform turns space inside out. Flat polygons might have their outlines distorted but they remain flat. You can therefore use rendering algorithms in 3DP space that linearly interpolate Z values across polygons. The plane $Z = 0$ (the eye plane) becomes the plane at infinity; the plane at infinity becomes $Z_s = Z_f/Z_f - Z_n$. Shapes that start out straddling the eye plane split in two.

From a user's point of view, the important message here is to make the value of Z_n as big as possible. This makes it easier for the renderer you are using to do depth comparisons. ❑

Topology references

1. S. Barr, *Experiments in Topology*, Thomas Y. Crowell, New York, 1964.
2. R.F. Riesenfeld, "Homogeneous Coordinates and Projective Planes in Computer Graphics," *J. ACM*, Vol. 1, No. 1, Jan. 1981 pp. 50-55.