Chapter 7

Perspective Projection and Arbitrary Viewing

Projection is the process where by data of dimension N is reduced to dimension M, where M is less than N. Our primary interest is projecting 3-D data to 2-D. Thus far we've done this by simply ignoring one of the dimensions. We now look at this process in more detail.

The act of projecting in graphics amounts to placing a plane of projection (PoP) between the eye/camera and the scene and having the points of the scene map to a location on this plane. What we've done to date has been a simple form of *parallel* projection, which assumes that if you were to draw lines between vertices in the scene and the points they map to on the plane of projection, the lines would all be parallel. In effect, we are assuming the eye/camera is at an infinite distance from the PoP. There are two primary categories of parallel projections: orthographic and oblique.

Orthographic parallel projection assumes that the plane of projection is perpendicular to the direction of the lines between the viewer and the scene (alternately, we could say the viewer is along the normal of the plane of projection). This is the most common form of parallel projection. The views down each of the three coordinate axes all fall into this category. We are not restricted to viewing our world down one of the coordinate axes, however. *Axonometric* views set the normal of the PoP so it is not aligned with any axis. *Isometric* projections look onto the world from a 45 degree angle in each dimension (a diagonal view), while *dimetric* projections position the normal of the PoP at an equal angle between two axes while allowing variation in the third. Finally, *trimetric* allows arbitrary off-axis positioning in all three directions.

Oblique parallel projection assumes that the plane of projection is not perpendicular to the lines between the viewer and the scene. This may be hard to envision until you realize that the transformation is simply a shearing in one or more dimensions. Thus we can look down one of the axes at a box which has been aligned with all three axes and see two or three faces, depending on whether the shear is in one direction or two. A shearing amount of 1 would make the side and top of a cube project to the same size as the front (called a *cavalier* view). A less dramatic view can be obtained with a shearing amount of .5 (called a *cabinet* view). Anyone who has taken a technical drawing

course has probably been exposed to this form of projection.

Alternatively, we could have the lines through the PoP converge to a camera position which is at a finite distance from the PoP. This is known as a *perspective* projection, which mimics the way humans view the world. If we map an edge from the scene which is parallel to the plane of projection, we note that its length on the screen is proportional to the relative positions of the edge, the camera, and the plane of projection. By using similar triangles we see that if the camera is at z=0, the plane of projection is at z=d, and a vertex is at z=z', the resulting coordinates on the plane of projection (px, py) are simply scaled versions of the original 3-D vertices (x', y'). This formula is simply px = x' * (d/z') and py = y' * (d/z'). This can be incorporated into our transformation matrix by making the third row, fourth column entry be 1/d and the fourth row, fourth column be 0 (remember, if w' does not equal 1 we divide all components of the resulting vector by w'). Some text books have slightly different variations on this matrix formulation, usually because some place the plane of projection at z=0 instead of z=d. You should be able to work out the math to prove to yourself that it works correctly. There are many other variants on projections which you should read about, though they won't be crucial for your projects.

The last of the "required" transformations we are interested in is to allow viewing of our world from arbitrary locations and orientations. What we'd like to do is create a camera coordinate system and place all objects into this system prior to clipping, projection, hidden surface elimination, and so on. Up until now, we have had a very simple camera coordinate system which was aligned with one of the three world coordinate axes (with translation along the axis). We can envision this process in two ways; in the first, the camera is moved and all objects remain fixed. In the second (the one we will explore), the camera remains fixed and all objects are moved. It should be clear that both interpretations can be applied to get a given scene. Thus the viewing transformation can be defined as the transformation which maps objects from the world coordinates into the camera coordinates. We assume the origin of the camera coordinate system is at the center of the plane of projection (the view reference point - VRP). We then need 3 orthogonal vectors to correspond to the axes of the coordinate system (often referred to (U, V, N) to avoid confusion with the world coordinates (X, Y, Z)). The N axis can be defined in a few ways. I like to use a "look at" point, L, in the scene, in conjunction with the VRP, to form this vector. Thus N = L - VRP. You then need an "up vector", V, which one could envision as a head or camera tilt. This must be perpendicular to the N vector to create a valid set of axes. To insure this, we will use an approximation which isn't necessarily perpendicular and refine it to obtain the correct orientation. If V' is an approximate up vector, we can generate the U vector which is perpendicular to both V' and N by taking the cross product of V' and N. We can then get the final version of V by taking the cross product of U and N. It is critical that V' is not colinear with N, or else this won't work.

We now can use U, V, and N to create a 3 x 3 transform which we can apply to each point to put it into the new coordinate system. The only thing we have to do is subtract the VRP from each point before multiplying the matrix. Thus we can transform each point as follows:

$$P' = (P - VRP)M, \; where \; M = \begin{pmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{pmatrix} \tag{7.1}$$

Some texts provide a 4 x 4 matrix that incorporates both of these transformations. If we pre-multiply the VRP with the U, V, and N vectors we get $R = (r_x, r_y, r_z) = (-VRP \cdot U, -VRP \cdot V, -VRP \cdot N)$. Thus the final matrix is

$$P' = PA, \ where \ A = \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ r_x & r_y & r_z & 1 \end{pmatrix} \tag{7.2}$$

Another common way to derive the transformation matrix is by starting with an N axis and VRP and perform the transformations necessary to align this with the world Z axis and origin, after which we apply a rotation about the z axis to account for tilt (let us say α degrees). This method appears in some graphics texts and is a bit more intuitive to some students. We create the compound transformation in the following manner.

1. Translate to place the VRP on the origin: $T_{VRP} = (-VRP_x, -VRP_y, -VRP_z)$
2. Compute the unit vector for N: $N = (L_x - VRP_x, L_y - VRP_y, L_z - VRP_z)$, where L is the lookat point. The unit vector $= N/|N| = (a, b, c)$ where, $a = N_x/|N|$, $b = N_y/|N|$, $c = N_z/|N|$, and $|N| = \sqrt{N_x^2 + N_y^2 + N_z^2}$
3. Rotate about the x-axis until in xz plane (use projection onto yz plane (0 b c)): the disired angle β has a hypotenus $d = \sqrt{b^2 + c^2}$, and thus $\cos\beta = c/d$ and $\sin\beta = b/d$. This gives us the transformation matrix:

$$R_{x,\beta} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.3}$$

4. Rotate about the y-axis until onto the positive z axis (start with (a 0 d)): the desired angle γ has components $\cos\gamma = d$ and $\sin\gamma = -a$. This gives the transformation matrix:

$$R_{y,\gamma} = \begin{pmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.4}$$

5. Rotate about the z-axis by desired head tilt angle α. This gives the transformation matrix:

$$R_{z,\alpha} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.5}$$

6. The entire transformation for each point would thus be: $P' = PT_{VRP}R_{x,\beta}R_{y,\gamma}R_{z,\alpha}$

Once the vertices of our world have been converted into the viewing coordinate system, we can apply what is termed a *prewarping* transform, which distorts the positions of our vertices to include perspective foreshortening. However, instead of mapping all depth values to the plane of projection, we conserve the relative depth of each point (called *pseudo-depth* while modifying the u and v coordinates as in the perspective projection transformation. If e_n is the position of the eye along the N axis (this is a negative number if you assume the plane of projection is at $N = 0$), the necessary transformation matrix is as follows:

$$P'' = P'W, \; where \; W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/e_n \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.6}$$

Finally, we need to perform a *normalization* process, which converts vertices in prewarped eye coordinates into 3-D screen coordinates. To accomplish this, we need to specify a number of parameters. Let W_{left}, W_{right}, W_{top}, and W_{bottom} define the range of u and v on the plane of projection which will map to the screen. Similarly, let V_{left}, V_{right}, V_{top}, and V_{bottom} define the range of pixel coordinates in our viewport which will contain the resulting image. Finally, let F and B define the front and back planes along the N axis (both should be in front of the eye, with B set to allow easy clipping of distant objects). The normalization matrix will convert depth values so that points between F and B will fall between 0 and 1 and thus positions points to allow simple clipping to a box defined by (V_{left}, V_{bottom}, 0) and (V_{right}, V_{top}, 1). The transformation is as follows:

$$P''' = P''R, \; where \; R = \begin{pmatrix} S_u & 0 & 0 & 0 \\ 0 & S_v & 0 & 0 \\ 0 & 0 & S_n & 0 \\ r_u & r_v & r_n & 1 \end{pmatrix} \tag{7.7}$$

Where
$S_u = (V_{left} - V_{right})/(W_{left} - W_{right})$
$S_v = (V_{top} - V_{bottom})/(W_{top} - W_{bottom})$
$S_n = [(e_n - B)(e_n - F)]/e_n^2(B - F)$
$r_u = (V_{right}W_{left} - V_{left}W_{right})/(W_{left} - W_{right})$
$r_v = (V_{bottom}W_{top} - V_{top}W_{bottom})/(W_{top} - W_{bottom})$
$r_n = F(e_n - B)/e_n(F - B)$

We can now compose these three transformations and apply them to all vertices with the following equation:

$$P' = PAWR \tag{7.8}$$

The objects are now in position to be clipped and rendered with hidden surfaces removed. This completes the graphics rendering pipeline!

Reading Topics: Projections and arbitrary viewing, Hill Chapter 7.

Project 7: This final "required" project has several steps. I strongly recommend that you finish them one at a time rather than trying to incorporate all of the functionality in it at once. First, expand your clipping algorithm from Module 6 to clip the 6 sides of a user-specified view volume and test it on your model from Module 5. You should be able to move your camera along the axis so that it is in among your objects and generate correct views. Now implement a perspective projection; this can be specified by simply entering the distance to the plane of projection. The x and y bounds (in camera coordinates) on the plane of projection can be used to generate 4 of the plane equations for clipping. The user can then specify near and far clipping planes for the camera's z axis. Finally, integrate a viewing transformation which allows views to be generated from arbitrary locations and orientations. This can be done by having the user enter a view reference point (the center of the plane of projection), a look-at point, and an up vector.