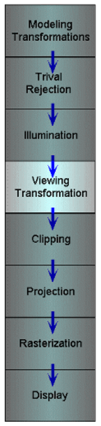
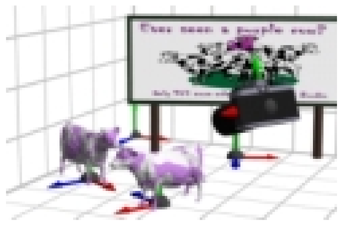
Viewing and Projection

- An Intuitive Camera Specification
- The Algebra of Projection
- A Canonical Viewing Volume

Viewing Transformations

Strictly speaking, there is no real need for a special set of viewing transformations. We can always compose one using a combination of rotations and translations. However, they occur so frequently that it is useful to develop a special class of transformations specifically for the purpose of mapping points from *world space* to *eye space*.

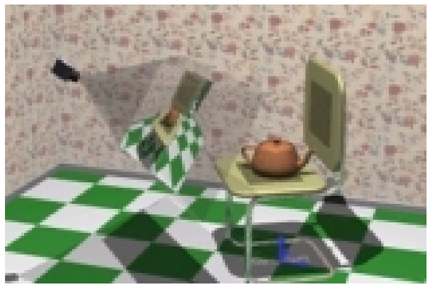




Some Motivation

Let's consider an example.

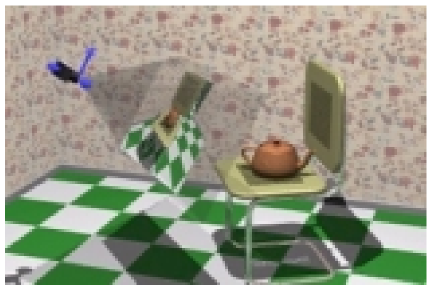
In the scene shown the origin of world space is shown as a blue coordinate frame located under the chair. This was a convenient coordinate system for orienting the floor and walls (Notice how the axes are aligned with the floor tiles). The chair and teapot could also be easily placed and oriented in this coordinate system.



The goal of a viewing transformation is to specify the position and orientation of our *3-D graphics camera* in the scene. However, its effect is almost the opposite.

Camera-Centered View

We will find that it is much more useful to reorient the entire scene such that the camera is located at the origin. Moreover, if we align the scene so that its optical axis (the direction it is looking) is along one of the coordinate axes and twist the scene so that the desired *up* direction is aligned with our camera's up direction **we can greatly simplify the clipping and projection steps that follow**.

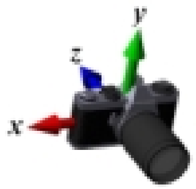


Once more, the mechanics of this specification can be expressed using the rigid body transformations discussed before. First, we need to perform the rotations needed to align the two coordinate frames.

Viewing Steps

First, we need to perform the rotations needed to align the two coordinate frames.

Then we need to perform a translation that will move the origin of our world space to the camera's origin. (Why did I rotate first and then translate?)



A More Intuitive Approach

It is more natural to position the camera in the world space as if it was another object.

Suppose that we identify a point where the camera is located (in world space), and call it the *eye point*. Then we identify some other world-space point in the scene that we wish to appear in the center of our view. We'll call this point the *look-at* point. Next, we identify a world space vector that we wish to be oriented upwards in our final image, and this point we'll call the *up-vector*.

This specification allows us to specify an arbitrary camera path by changing only the eye point and leaving the look-at and up vectors untouched. Or we could pan the camera from object to object by leaving the eye-point and up-vector fixed and changing only the look-at point.

Deriving a Natural Viewing Transform

As discussed before, we will compute the rotation part of the viewing transformation first. Fortunately, we know some things about the rotation matrix that we are looking for.

For instance, consider a vector along the look-at direction:

$$\begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} = \begin{bmatrix} lookat_x \\ lookat_y \\ lookat_z \end{bmatrix} - \begin{bmatrix} eye_x \\ eye_y \\ eye_z \end{bmatrix}$$

After normalizing it

$$\hat{l} = \frac{\bar{l}}{\sqrt{l_x^2 + l_y^2 + l_z^2}}$$

We expect our desired rotation matrix to map it to the vector *[0, 0, -1]*^{T} (Why?).

$$\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \mathbf{V}\hat{l}$$

Deriving a Natural Viewing Transform

There is another special vector that we can compute. If we find the cross product between the look-at vector with our up vector, we will get a vector that points to the right.

$$\bar{r} = \bar{l} \times \overline{up}$$

We expect that this vector when normalized will transform to the vector *[1, 0, 0]*^{T}.

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{V}\frac{\bar{r}}{\sqrt{r_x^2 + r_y^2 + r_z^2}}$$

One More Vector

Finally, from these two vectors we can synthesize a third vector in the up direction.

$$\bar{u} = \bar{r} \times \bar{l}$$

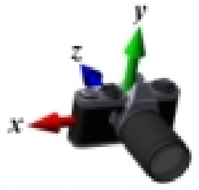
And this vector, when normalized, should transform to *[0, 1, 0]^T*.

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \mathbf{V} \frac{\bar{u}}{\sqrt{u_x^2 + u_y^2 + u_z^2}}$$

Now we know everything that our viewing matrix must do; we need only to assemble it.

Rotations are Easy to Invert

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{V}\begin{bmatrix} \hat{r} & \hat{u} & -\hat{l} \end{bmatrix}$$



Therefore, the rotation component of our viewing transformation is just the transpose of the matrix formed by our selected vectors.

$$\mathbf{V}_{rotate} = \begin{bmatrix} \hat{r} \\ \hat{u} \\ -\hat{l} \end{bmatrix}$$

Composing a Result

Let's consider all of these results together.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{V}\begin{bmatrix} \hat{r} & \hat{u} & -\hat{l} \end{bmatrix}$$

In order to compute the matrix, **V** we need only compute the inverse of the matrix formed by concatenating our 3 special vectors. We will instead employ a little trick from linear algebra.

Remember that each of our vectors are unit length (we normalized them). Also, each vector is perpendicular to the other two. These two conditions on a matrix makes it, **orthogonal**. Rotations are also orthogonal. Orthonormal matrices have the unique property that:

$$\mathbf{V}^{-1} = \mathbf{V}^{T} \quad \text{if V is orthonormal}$$

Now for the Translation

The rotation that we just derived is specified about the eye point. However, we are still in world space at this point. In order for the rotation to occur about the correct point if we must first subtract the eye point from any given world space point.

$$\begin{bmatrix} \hat{r} \\ \hat{u} \\ -\hat{l} \end{bmatrix}\begin{bmatrix} x - eye_x \\ y - eye_y \\ z - eye_z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Next we isolate the rotation and translation parts.

$$\begin{bmatrix} \hat{r} \\ \hat{u} \\ -\hat{l} \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} \hat{r} \\ \hat{u} \\ -\hat{l} \end{bmatrix}\begin{bmatrix} eye_x \\ eye_y \\ eye_z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

The Viewing Transformation

Finally we can compose the rotation and translation into a single 4 by 4 matrix giving **V**:

$$\begin{bmatrix} & \hat{r} & & -\hat{r}\cdot\overline{eye} \\ & \hat{u} & & -\hat{u}\cdot\overline{eye} \\ & -\hat{l} & & \hat{l}\cdot\overline{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

And the viewing transform is complete:

Projection Transformation

Our lives are greatly simplified by the fact that viewing transformations transform the eye to the origin and the look-at direction (optical axis) to a specified coordinate axis. This greatly reduces the range of possible projection matrices.

The projection transformation maps all of our 3-D coordinates onto our desired viewing plane. Thus, making our 3-D world into a 2-D image. This sort of mapping is not affine like all of the transforms we've discussed thus far. In fact, projection matrices do not transform points from our affine space back into the same space. They transform points into something different. Usually, we will use a projection matrix to reduce the dimensionality of our affine points.

Thus, we should expect projection matrices to be *less than full rank*.

Orthographic Projection

The simplest form of projection, is to simply project all points along lines parallel to the z-axis. This form of projection is called **orthographic** or **parallel**. It is the common form of projection used by draftspeople for top, bottom, and side views. The advantage of parallel projection is that the you can make accurate measurements of image features in the two dimensions that remain. The disadvantage is that the images don't appear natural (i.e. they lack perspective foreshortening).

Here is an example of a parallel projection of our scene.

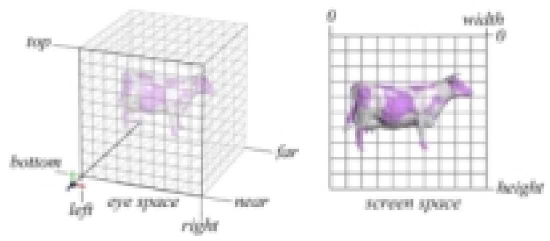
Notice that the parallel lines of the tiled floor remain parallel after orthographic projection.

Orthographic Projection

The projection matrix for orthographic projection is very simple

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

There are some problems with this simple form, however. To begin with the units of the transformed points are still the same as the model. This is great for drafting, but in our case we'd like for the units to be in pixels.



Mapping to Pixel Coordinates

We can incorporate this change of units, and perform the flip of the y-axis required for raster coordinates into our projection matrix as follows.

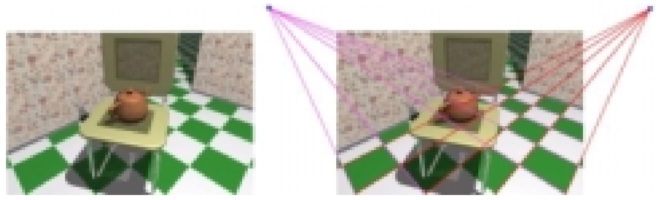
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{width}{right-left} & 0 & 0 & \frac{-left \times width}{right-left} \\ 0 & \frac{height}{bottom-top} & 0 & \frac{-top \times height}{bottom-top} \\ 0 & 0 & \frac{z_{\max}}{far-near} & \frac{-near \times z_{\max}}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

This process is often called the viewport transformation. The variables, *left*, *right*, *top* and *bottom* refer to the extents of the viewing frustum in modeling units. The values *width* and *height* are in unit of pixels.

This transformation is little more than a scale and a translation.

Perspective Projection

Artists (Donatello, Brunelleschi, and Da Vinci) during the Renaissance discovered the importance of perspective for making images appear realistic. This outdates mathematicians by more than 300 years. Perspective causes objects nearer to the viewer to appear larger than the same object would appear farther away. Another reason for introducing homogenous coordinates to computer graphics was to accomplish perspective projections using linear operators.



Note how lines known to be parallel in image space appear to converge to a single point when viewed in perspective. This is an important attribute of lines in projective spaces; they always intersect at a point.

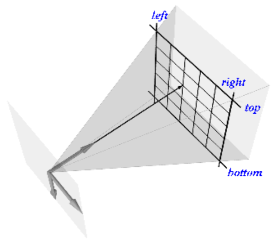
Perspective Projection

The matrix for perspective projection is:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Notice how similar this transform is to the original parallel projection. Once more the units of the transform are those of the model and not pixels.

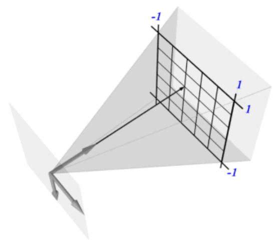
We need to decide *where* (at what depth) we will specify the values of left, right, top, and bottom. Our convention will be to specify these at the *near plane*.



Perspective Projection

The following transformation accomplishes the projection and the conversion to pixels in a single transform.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{width \times near}{right-left} & 0 & \frac{-left \times width}{right-left} & 0 \\ 0 & \frac{height \times near}{bottom-top} & \frac{-height \times top}{bottom-top} & 0 \\ 0 & 0 & \frac{z_{\max} \times far}{far-near} & \frac{-z_{\max} \times far \times near}{far-near} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



For the purpose of clipping we can modify this transformation so that it is mapped into a canonical space.

Canonical Perspective Projection

Following is a canonical space mapping for orthographic projections.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & \frac{-(right+left)}{right-left} \\ 0 & \frac{2}{bottom-top} & 0 & \frac{-(bottom+top)}{bottom-top} \\ 0 & 0 & \frac{2}{far-near} & \frac{-(far+near)}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Next is a canonical space mapping for perspective projections.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2 \times near}{right-left} & 0 & \frac{-(right+left)}{right-left} & 0 \\ 0 & \frac{2 \times near}{bottom-top} & \frac{-(bottom+top)}{bottom-top} & 0 \\ 0 & 0 & \frac{far+near}{far-near} & \frac{-2 \times far \times near}{far-near} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Next Time