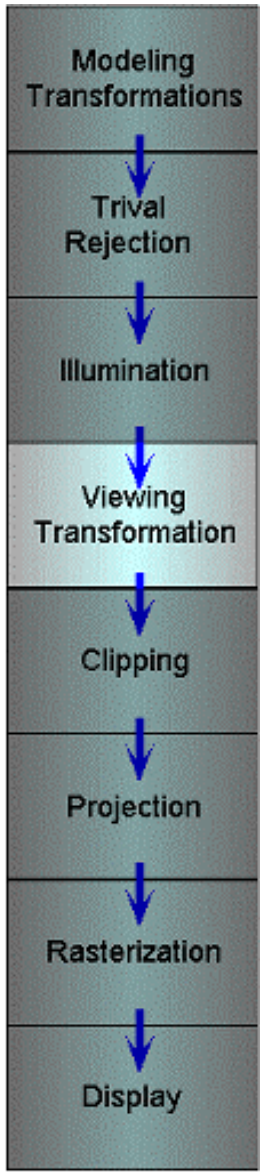
Viewing and Projection

Camera Specification

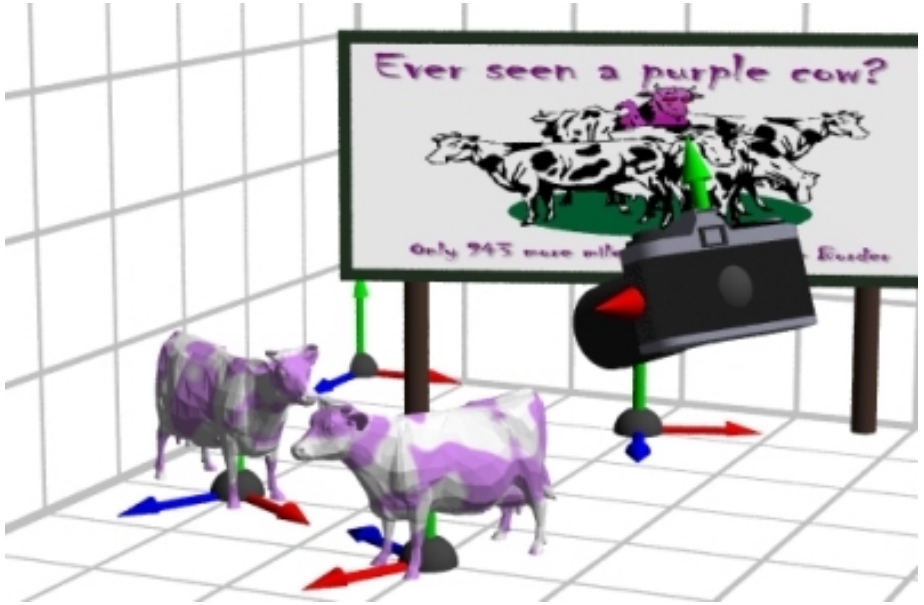
Projection Matrices

Viewing Frustum

Viewing Transformations



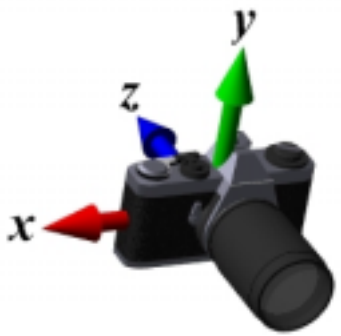
Strictly speaking, there is no real need for a special set of viewing transformations. We can always compose one using a combination of rotations and translations. However, they occur so frequently that it is useful to develop a special class of transformations specifically for the purpose of mapping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



Motivation

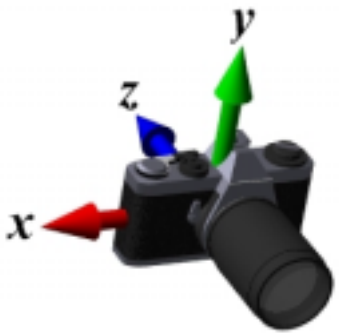
In the scene shown, the origin of world space is shown as a blue coordinate frame located under the chair. This was a convenient coordinate system for orienting the floor and walls (Notice how the axes are aligned with the floor tiles). The chair and teapot could also be easily placed and oriented in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



We will find that it is beneficial to reorient the entire scene such that the camera is located at the origin. If we align the scene so that the camera's optical axis is along one of the coordinate axes and twist the scene so that the desired up direction is aligned with the camera's up direction we can greatly simplify the clipping and projection steps that follow. Once more, the viewing transformation 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



$$\vec{e}^t = \vec{w}^t E$$
$$\vec{e}^t V = \vec{w}^t$$
$$V = E^{-1}$$

Viewing transformation should align the world and camera coordinate frames.

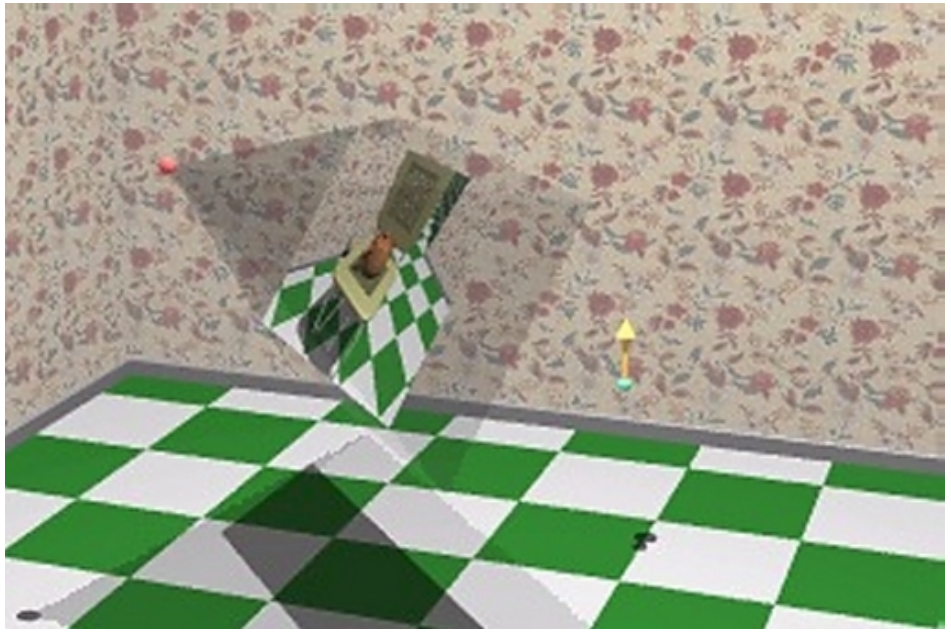
Rotate

Translate

Intuitive Camera Specification

We define the camera (eye) location by identifying the origin of the camera frame in the world space. Viewing direction can be specified with another world-space point, the look-at point, that should appear in the center of the camera image. Last, an up-vector specifies the camera orientation by defining a world space vector that should be oriented upwards in the final image.

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.



Look-At 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:

$$l = \begin{bmatrix} \text{lookat}_x \\ \text{lookat}_y \\ \text{lookat}_z \end{bmatrix} - \begin{bmatrix} \text{eye}_x \\ \text{eye}_y \\ \text{eye}_z \end{bmatrix}, \quad \hat{l} = \frac{l}{\|l\|} = \frac{l}{\sqrt{l_x^2 + l_y^2 + l_z^2}}$$

The viewing transform should map the normalized vector $\hat{l}$

$$\vec{w}^t \begin{bmatrix} \hat{l} \\ 0 \end{bmatrix} \mapsto \vec{e}^t \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} = \vec{w}^t E \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}$$

Look-At Viewing Transform

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

$$r = l \times \text{up}$$

The viewing transform should map this normalized vector

$$\vec{w}^t \begin{bmatrix} \hat{r} \\ 0 \end{bmatrix} \mapsto \vec{e}^t \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \vec{w}^t E \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Look-At Viewing Transform

Last, from the look-at vector and the right vector we can synthesize a third vector in the up direction.

$$\hat{u} = \hat{\mathrm{r}} \times \hat{l}$$

The viewing transform should map this normalized vector

$$\vec{w}^t \begin{bmatrix} \hat{u} \\ 0 \end{bmatrix} \mapsto \vec{e}^t \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \vec{w}^t E \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Rotation: Composing Results

Now lets consider all of these results together.
In order to compute the rotation part of the viewing matrix, we need only compute the inverse of the matrix whose columns are formed by our special vectors.

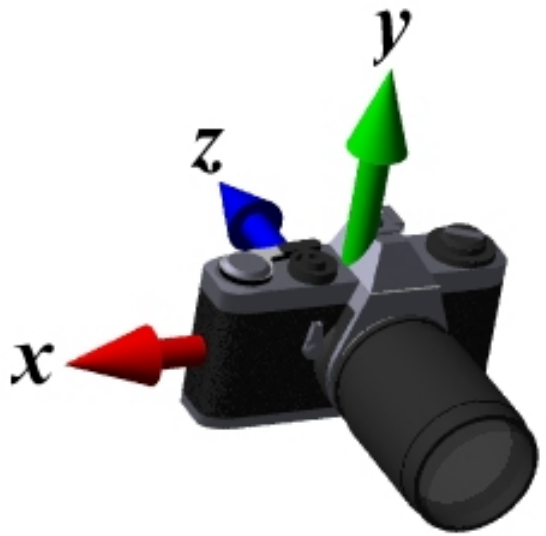
$$E_R = \begin{bmatrix} \hat{r} & \hat{u} & -\hat{l} \end{bmatrix}, \quad E = \begin{bmatrix} E_R & ? \\ 0 & 1 \end{bmatrix}$$

We will employ a little trick from linear algebra. Remember that each column vector is unit length (we normalized them). Also, each vector is perpendicular to the other two. These two conditions make the matrix orthogonal. Rotations are also orthogonal. Orthogonal matrices have a unique property:

$$E_R^{-1} = E_R^T$$

Rotations are Easy to Invert

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



$$V_R = E_R^{-1} = \begin{bmatrix} \hat{r}^T \\ \hat{u}^T \\ -\hat{l}^T \end{bmatrix}$$

Translation

The rotation that we just derived aligns the world axes with the eye axes. However, the origins of the coordinate frames still do not match. We match the origins by subtracting the eye point from any given world space point. We isolate the rotation and translation parts:

$$\begin{bmatrix} \hat{r}^T \\ \hat{u}^T \\ -\hat{l}^T \end{bmatrix} \begin{bmatrix} x - \text{eye}_x \\ y - \text{eye}_y \\ z - \text{eye}_z \end{bmatrix} = \underbrace{\begin{bmatrix} \hat{r}^T \\ \hat{u}^T \\ -\hat{l}^T \end{bmatrix}}_{E_R^{-1}} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \underbrace{\begin{bmatrix} \hat{r}^T \\ \hat{u}^T \\ -\hat{l}^T \end{bmatrix} \begin{bmatrix} \text{-eye}_x \\ \text{-eye}_y \\ \text{-eye}_z \end{bmatrix}}_{E_T^{-1}}$$

Translation

Or we can derive the translation component by observing how the eye point should transform:

$$\vec{w}^t \begin{bmatrix} \text{eye}_x \\ \text{eye}_y \\ \text{eye}_z \\ 1 \end{bmatrix} \mapsto \vec{e}^t \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \vec{w}^t E \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

This directly defines the last column of the E matrix whose inverse is the viewing matrix:

$$E = \begin{bmatrix} \hat{r} & \hat{u} & -\hat{l} & \text{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How do we invert this transformation matrix?

Inverting Homogenous Matrix

We decompose the matrix into a rotation and translation and invert both separately:

$$\begin{aligned} \vec{e}^t &= \vec{w}^t E \\ &= \vec{w}^t E_T E_R \\ \vec{e}^t E_R^{-1} &= \vec{w}^t E_T \\ \vec{e}^t E_R^{-1} E_T^{-1} &= \vec{w}^t \end{aligned}$$

Recomposing the matrices with multiplication computes the viewing transform:

$$V = E_R^{-1} E_T^{-1} = \begin{bmatrix} & \hat{r}^{\mathrm{T}} & & 0 \\ & \hat{u}^{\mathrm{T}} & & 0 \\ & -\hat{l}^{\mathrm{T}} & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & -\mathrm{eye} \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation

$$\vec{w}^t \mathbf{c} = \vec{e}^t V \mathbf{c} = \vec{e}^t \begin{bmatrix} & \hat{r}^{\mathrm{T}} & & -\hat{r}^{\mathrm{T}}\mathrm{eye} \\ & \hat{u}^{\mathrm{T}} & & -\hat{u}^{\mathrm{T}}\mathrm{eye} \\ & -\hat{l}^{\mathrm{T}} & & \hat{l}^{\mathrm{T}}\mathrm{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{c}$$

Projection Transformation

Our lives are greatly simplified by the fact that viewing transformations map 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 dimensionally 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 an 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 simple:

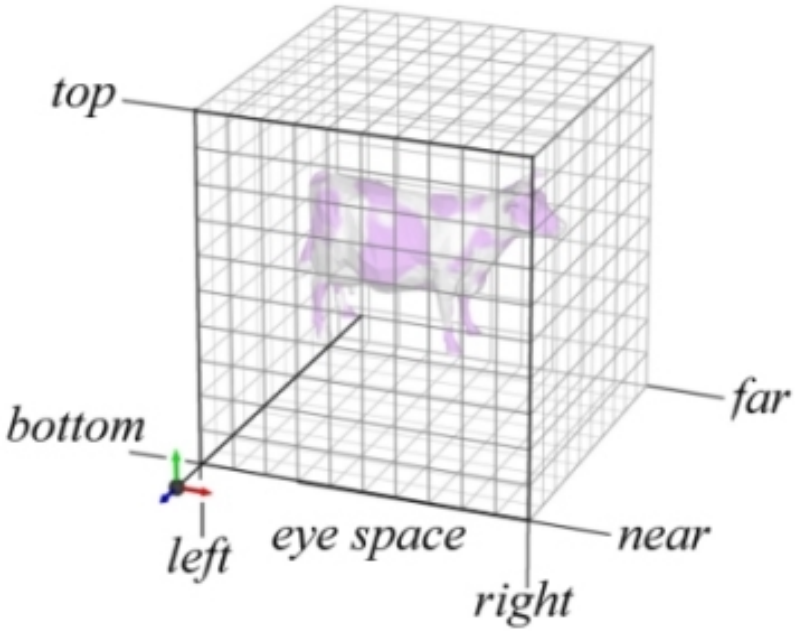
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

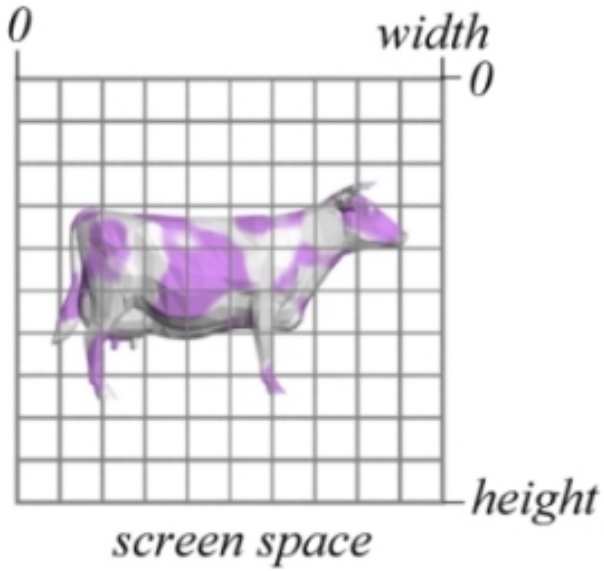
There are some problems with this simple form, however. To begin with the units of the transformed points are still the same as the model units. This is great for drafting, but in our case we'd like for the units to be in pixels.

Mapping to Pixel Coordinates

This process is often called the viewport transformation. The variables, left, right, top, bottom, near and far 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.





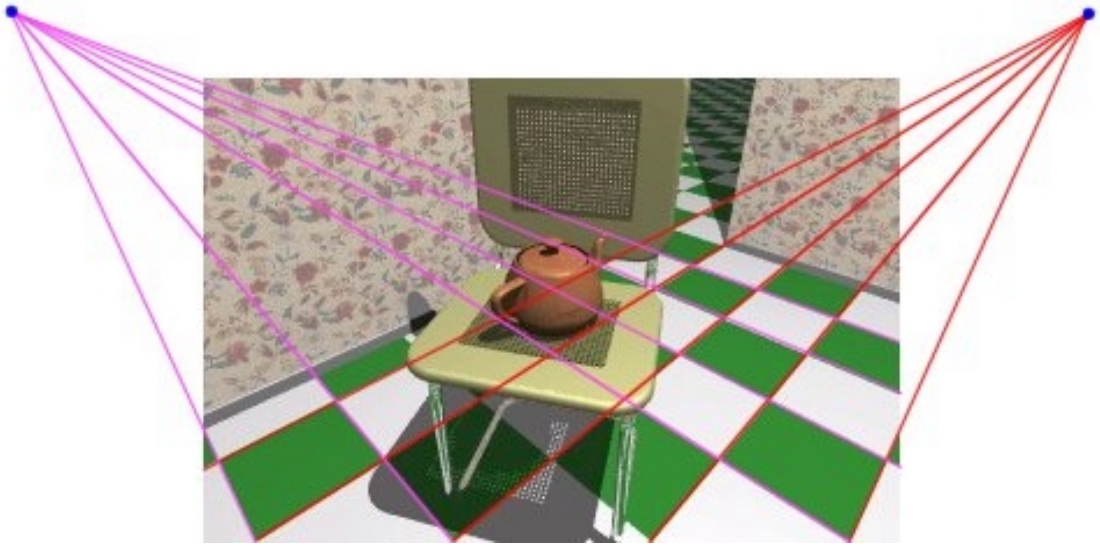
Mapping to Pixel Coordinates

We can incorporate this change of units, and perform the flip of the y-axis required for raster coordinates into our projection matrix. We can also re-map the z-coordinate such that the new coordinates preserve the original depth ordering.

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

Perspective Projection

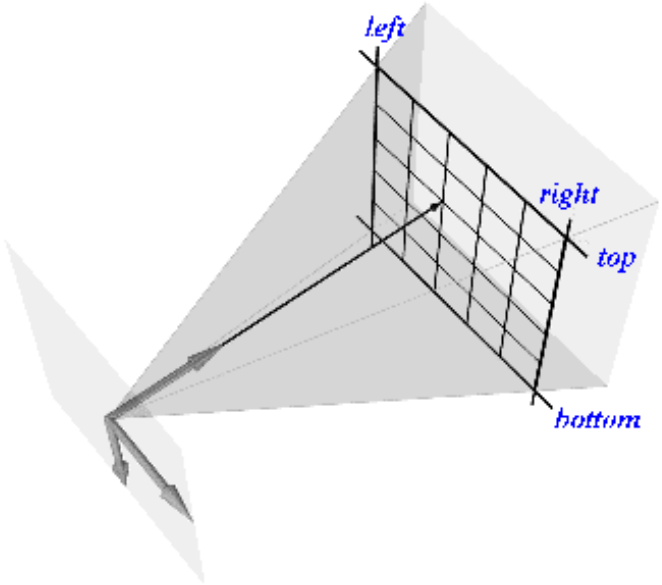
Artists (Donatello, Brunelleschi, and Da Vinci) during the renaissance discovered the importance of perspective for making images appear realistic. This outdates mathematicians by more than 300 years. Perspective causes objects nearer to the viewer to appear larger than the same object would appear farther away. 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 projection matrix for perspective projection matrix is:

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



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.

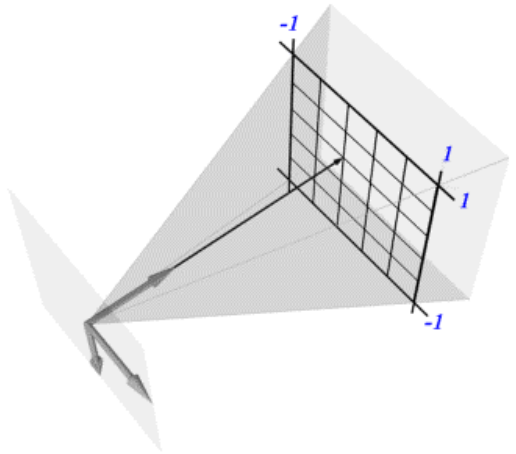
We need to decide where (at what depth) we will specify the values of left, right, top, and bottom. Our convention will be to specify these at the near plane.

Perspective Projection

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

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

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



Canonical Perspective Projection

Orthographic

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

Perspective

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

Next Time