Lecture 5

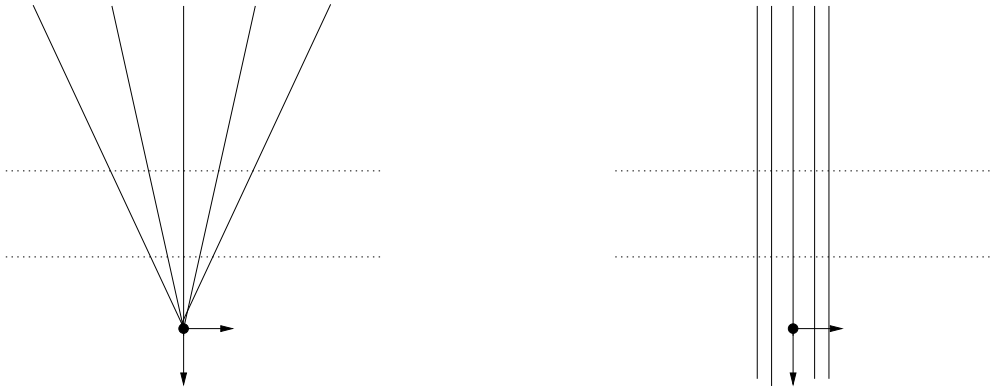
Projective transformation (from eye to clipping coordinates)

Last lecture we considered how to project 3D points to the 2D projection plane. The problem with doing this is that we would lose the information about the depth of the 3D points. We need to preserve depth information in order to know which surfaces are visible in an image, i.e. if surface A lies between the camera and surface B, then A is visible but B is not.

Specifically, recall the projection matrix we saw last class, where $f < 0$,

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

This matrix is non-invertible. You can easily see this mathematically by noting that the 3rd and 4th rows are linearly dependent. To keep information about the depth (z) ordering of points, we will choose a slightly different 4×4 transformation that it *is* invertible. This is called a *projective transformation.* The intuitive idea is shown in this figure. We are going to transform the scene by taking lines that pass through the camera position and make these lines parallel. (In the figure on the left, the lines stop at the origin – but this is just for illustration purposes.) Such a transformation will take lines that *don't* pass through the camera position and transform them too, but we don't care about those lines since they play no role in our image.



Define the near and far planes as before to be at $z = f_0 = -near$ and $z = f_1 = -far$, respectively. To get an invertible matrix, we change the third row of the above projection matrix:

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and require only that $\beta \neq 0$ which forces the matrix to be invertible since the four rows are linearly independent. We choose α and β such that

- the z value of points *on the near and far planes* ($z = -near$ and $z = -far$, respectively) is preserved under this transformation. That is, $z = f_0$ maps to $z = f_0$, and $z = f_1$ maps to $z = f_1$.
- the (x, y) values are preserved within the plane $z = f_0$ (and by the previous point, it follows that the (x, y, z) values in the plane $z = f_0$ are *all* preserved under the transformation.)

As I show in the Appendix, the values are $\alpha = f_0 + f_1$ and $\beta = -f_0 f_1$. I verified in the lecture slides that the following matrix has the properties just defined, but you should do it for yourself now as an exercise.

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & f_0 + f_1 & -f_0 f_1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

This matrix maps a scene point from eye coordinates to a new coordinate system called *clipping coordinates*. The term 'clipping coordinates' refers to a 4D homogeneous coordinate representation of a point, although keep in mind that the 4D point represents a 3D point. We'll discuss the term "clipping" in more detail next lecture. For now, let's better understand what this transformation does.

As sketched above, this transformation maps lines that pass through the origin to lines that are parallel to the z direction. Let's examine the points that are outside the view volume in more detail. How are they transformed by the projective transformation above? This analysis turn out to be useful next lecture when we ask how to discard ("clip") points that are outside the view volume.

First, how are points at infinity transformed? These points are outside the (finite) view volume, but it is not clear where they map to. Applying the projective transform gives:

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & f_0 + f_1 & -f_0 f_1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} f_0 x \\ f_0 y \\ (f_0 + f_1) z \\ z \end{bmatrix} = \begin{bmatrix} f_0 \frac{x}{z} \\ f_1 \frac{y}{z} \\ f_0 + f_1 \\ 1 \end{bmatrix}$$

Thus, all points at infinity map to the plane $z = f_0 + f_1$, which is beyond the far plane. Keep in mind that f_0 and f_1 are both negative.

Next, where are points at $z = 0$ mapped to? Note that $z = 0$ is the plane containing the camera, since the context is that we are mapping *from* camera coordinates *to* clip coordinates.

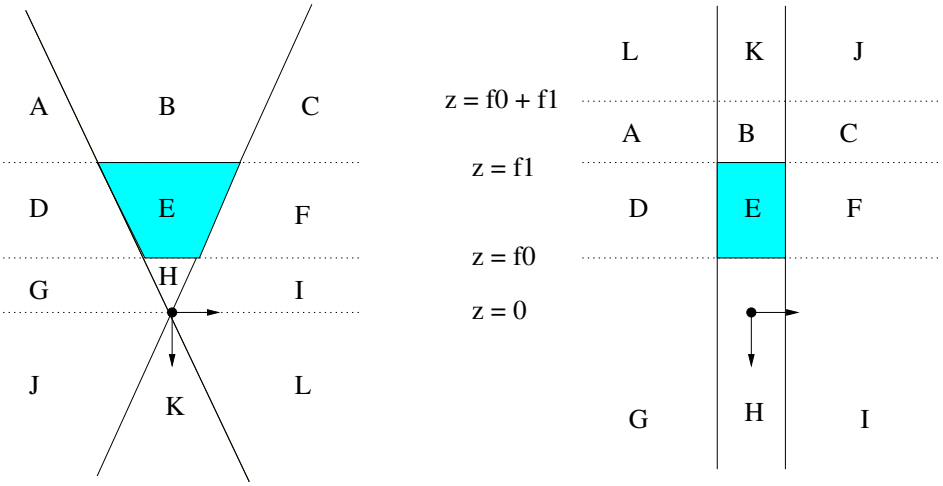
$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & f_0 + f_1 & -f_0 f_1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} f_0 x \\ f_0 y \\ -f_0 f_1 \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -f_1 \\ 0 \end{bmatrix} = \begin{bmatrix} -x \\ -y \\ f_1 \\ 0 \end{bmatrix}$$

Thus, $z = 0$ is mapped to points at infinity.

For a general scene point, we have:

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & f_0 + f_1 & -f_0 f_1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f_0 x \\ f_0 y \\ (f_0 + f_1) z - f_0 f_1 \\ z \end{bmatrix} = \begin{bmatrix} \frac{f_0}{z} x \\ \frac{f_0}{z} y \\ (f_0 + f_1) - \frac{f_0 f_1}{z} \\ 1 \end{bmatrix}$$

Let's compare the space in front of the camera ($z < 0$) to the space behind the camera ($z > 0$). For the space behind the camera, $\frac{f_0}{z} < 0$ since $f_0 < 0$ and $z > 0$ for these points. Thus this transformation changes the sign (and magnitude) of the x and y coordinates. Moreover, since $\frac{f_0 f_1}{z} > 0$, the half space $z > 0$ (JKL in figure below) is mapped to $z < f_0 + f_1$, that is, it maps to a space *beyond* the far plane. Combining the above two observations, we see that the regions labelled JKL on the left are mapped to the regions LKJ on the right.



Since we are only going to draw those parts of the scene that lie within the view volume (region E), you might think that it doesn't matter that the regions JKL behave so strangely under the projective transformation. However, *all* points gets transformed to clipping coordinates. In order to reason about which points can be discarded (or "clipped"), we need to keep all the regions in mind.

In particular, note that $(x, y, z, 1)$ maps to a 4D point (wx', wy', wz', w) such that $w = z$. (Verify this above!) Since points in the view volume always have $z < 0$, it follows that if we were to use this projection matrix above then we would have a negative w coordinate. OpenGL's avoids this by defining the $\mathbf{M}_{projective}$ matrix to be the negative of the above matrix, namely OpenGL uses:

$$\begin{bmatrix} -f_0 & 0 & 0 & 0 \\ 0 & -f_0 & 0 & 0 \\ 0 & 0 & -(f_0+f_1) & f_0 f_1 \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} \text{near} & 0 & 0 & 0 \\ 0 & \text{near} & 0 & 0 \\ 0 & 0 & \text{near} + \text{far} & \text{near} * \text{far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Taking the negative of the matrix doesn't change the mapping. But the new matrix instead has the property that $w > 0$ and $z < 0$ for points in the view volume. As I'll mention briefly again below, this slightly simplifies the reasoning about whether a point *could* be in the view volume.

Normalized view volume

After performng the perspective transformation which maps the truncated pyramid frustum to a rectanguloid volume, it is common to transform once more, in order to normalize the rectanguloid volume to a cube centered at the origin.

$$[\text{left}, \text{right}] \times [\text{bottom}, \text{top}] \times [-\text{far}, -\text{near}] \quad \rightarrow \quad [-1, 1] \times [-1, 1] \times [-1, 1],$$

This transformation is achieved by a translation, followed by a scaling, followed by a translation:

$$\mathbf{M}_{normalize} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & \frac{-2}{far-near} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -left \\ 0 & 1 & 0 & -bottom \\ 0 & 0 & 1 & near \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The first translation lines up the left, bottom, and near planes with $x = 0, y = 0, z = 0$ respectively. The second transformation scales the x, y, z dimensions of the rectanguloid to a cube of width and height and depth of 2. The final translation shifts the cube so it is centered at the origin.

Notice the -2 factor instead of 2 in the (3,3) component of the scaling matrix. The z value is not merely scaled by 2, but it is also negated (i.e. reflection about the $z = 0$ plane). This flipping of the z axis switches from the usual right handed coordinate system to a left handed coordinate system. With the back of the hand facing toward you, x (thumb) is to the right, y (index) is up, and z (middle) points *away* from the camera. i.e. visible points that are farther from the camera now have a greater z values.

Clip Coordinates

The $\mathbf{M}_{normalize}$ transform does not affect the 4th coordinate (w) so if $w > 0$ prior to the normalization then it will be so after normalization as well. Do not confuse this with what happens to the sign of z. The normalization flips z (the factor -2 above) and maps it to $[-1, 1]$ so it can easily happen that the sign of z changes. But the sign of w does not change.

After normalization, the 3D points that lie within the view volume will all be normalized to the 4D points (wx, wy, wz, w) such that the x, y, z values all lie in $[-1, 1]$ and $w > 0$. Thus, these values all satisfy all of the following inequalities:

$$w > 0$$
$$-w \leq wx \leq w$$
$$-w \leq wy \leq w$$
$$-w \leq wz \leq w.$$

It is easy to check if these inequalities hold for a given point (wx, wy, wz, w) and hence if the point needs to be considered further or discarded (or "clipped"). That is the reaons such coordinates are referred to as *clip coordinates*.

[ASIDE: After the lecture, I was asked why we need to do the projective transform. It seems easy enough to do clipping in the original space, no? In particular, testing that z is between $-near$ and $-far$ is easy enough in the original space. So why make things so complicated? The answer is that, although the z condition is about equally easy to check in the original (camera) coordinates as in the clip coordinates, checking whether a point lies beyond the slanted side walls of the view volume is clearly more complicated in the camera coordinates than in clip coordinates. Bottom line: we don't *need* to do all these tests in clip coordinates, but it is simple to do.]

Summary (thus far) of the OpenGL pipeline

The last few lectures we have developed a sequence of transformations:

$$\mathbf{M}_{normalize}\mathbf{M}_{projective}\mathbf{M}_{camera \leftarrow world}\mathbf{M}_{world \leftarrow object}$$

where the transformations are applied from right to left.

In OpenGL, the transformation defined by the product $\mathbf{M}_{normalize}\mathbf{M}_{projective}$ is represented by a 4×4 `GL_PROJECTION` matrix. This matrix is defined by the commands `glFrustrum` or alternatively `gluPerspective` which I introduced last lecture. Note that *near* and *far* only play a role in $\mathbf{M}_{projective}$ whereas, as we saw last page, *top, bottom, left, right* are used for $\mathbf{M}_{normalize}$

The product $\mathbf{M}_{camera \leftarrow world}\mathbf{M}_{world \leftarrow object}$ is represented by the 4×4 `GL_MODELVIEW` matrix. The matrix $\mathbf{M}_{camera \leftarrow world}$ is specified by the OpenGL command `gluLookAt`. The matrix $\mathbf{M}_{world \leftarrow object}$ is specified by a sequence of `glRotate`, `glTranslate`, and `glScale` commands which are used to put objects into a position and orientation in the world coordinate system. These transformations are also used to position and orient and scale the parts of an object relative to each other.

Perspective division, normalized device coordinates

In order to explicitly represent the 4-tuple (wx, wy, wz, w) as a 3-tuple (x, y, z), one needs to divide the first three components of (wx, wy, wz, w) by the fourth component i.e. the w value. This step is called *perspective division*. It transforms from 4D clip coordinates (wx, wy, wz, w) to a 3D normalized view coordinates (x, y, z), also known as *normalized device coordinates* (NDC). Note that NDC is in left handed coordinates, with z increasing away from the camera.

In OpenGL, perspective division step occurs only after it has been determined that the point indeed lies within the view volume. That is, perspective division occurs after clipping. This ordering is not necessary, but it's just how OpenGL has done it historically.

Addendum: more on projective transformations

Projective transformation of a plane

When we use the term "polygon" we have in mind a set of vertices that all lie in a plane. It should be intuitive that a polygon remains a polygon under a modelview transformation since this transformation only involves rotation, translation, and scaling. It is less intuitive, however, whether our projective transformation also maps polygons to polygons (planes to planes). So, let us show why it does.

Consider any 4×4 invertible matrix $\mathbf{M}$ and consider the equation of a plane in $\Re^3$,

$$ax + by + cz + d = 0$$

which can be written as

$$[a, b, c, d]\ [x, y, z, 1]^T = 0.$$

We are assuming the matrix $\mathbf{M}$ is invertible, so we can insert $\mathbf{M}^{-1}\mathbf{M}$ as follows:

$$[a, b, c, d]\ \mathbf{M}^{-1}\mathbf{M}\ [x, y, z, 1]^T = 0.$$

Define

$$[w'x', w'y', w'z', w'] = \mathbf{M}\ [x, y, z, 1]^T$$

where we have assumed $w' \neq 0$. Similarly, define

$$[a', b', c', d'] \equiv [a, b, c, d]\ \mathbf{M}^{-1}.$$

Then,

$$[a', b', c', d']\ [w'x', w'y', w'z', w']^T = 0$$

which is the equation of a plane in (x', y', z'). Thus, $\mathbf{M}$ maps planes to planes.

Projective transformation of surface normal

What about the normal to a plane? Does a projective transformation map the normal of a plane to the normal of the transformed plane? It turns out that it does *not*. Why not?

Consider projective transformation matrix $\mathbf{M}$. Let $\mathbf{e} = (e_x, e_y, e_z)$ be a vector that is perpendicular to the surface normal vector $\mathbf{n} = (n_x, n_y, n_z)$. For example, $\mathbf{e}$ could be defined by some edge of the polygon, i.e. the vector difference $\mathbf{p}_1 - \mathbf{p}_0$ of two vertices . By definition of "surface normal", the vectors $\mathbf{n}$ and $\mathbf{e}$ must satisfy:

$$n_x e_x + n_y e_y + n_z e_z = \mathbf{n}^T \mathbf{e} = 0.$$

Write $\mathbf{e}$ and $\mathbf{n}$ as direction vectors $(e_x, e_y, e_z, 0)$ and $(n_x, n_y, n_z, 0)$. When the scene is transformed to clip coordinates, the normal $\mathbf{n}$ and vector $\mathbf{e}$ are transformed to $\mathbf{M}\ \mathbf{n}$ and $\mathbf{M}\ \mathbf{e}$, respectively. Note there is no reason why the projective transformation *should* preserve angles (e.g. 90 degree angles) and so there is no reason why $\mathbf{Mn}$ should remain perpendicular to $\mathbf{Me}$. So there is no obvious reason why $\mathbf{Mn}$ *should* be the normal to the corresponding polygon in the clip coordinates.

What *is* the normal vector to the polygon in clip coordinates? The normal must be perpendicular to any transformed edge $\mathbf{Me}$, so

$$0 = \mathbf{n}^T \mathbf{e} = \mathbf{n}^T \mathbf{M}^{-1} \mathbf{M} \mathbf{e} = ((\mathbf{M}^{-1})^T \mathbf{n})^T (\mathbf{M}\mathbf{e})$$

Thus, the vector $(\mathbf{M}^{-1})^T \mathbf{n}$ is perpendicular to all the transformed edges $\mathbf{M}\ \mathbf{e}$ of the polygon. Thus $(\mathbf{M}^{-1})^T \mathbf{n}$ must be the normal to the polygon in clip coordinates. Note that in general

$$(\mathbf{M}^{-1})^T \mathbf{n} \neq \mathbf{Mn}$$

and indeed you need special circumstances for this to hold e.g. $\mathbf{M}$ could be a rotation matrix.

Appendix

We first take a point (x, y, f_0) that lies on the near plane. Then,

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ f_0 \\ 1 \end{bmatrix} = \begin{bmatrix} f_0 x \\ f_0 y \\ f_0\ \alpha + \beta \\ f_0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \alpha + \frac{\beta}{f_0} \\ 1 \end{bmatrix}$$

Since we require the transformation to preserve the z values for points on the f_0 plane, we must have:

$$\alpha + \frac{\beta}{f_0} = f_0.$$

Note that the x and y values will be preserved for these points!

What about points on the $z = f_1$ plane?

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ f_1 \\ 1 \end{bmatrix} = \begin{bmatrix} f_0 x \\ f_0 y \\ f_1\ \alpha + \beta \\ f_1 \end{bmatrix} = \begin{bmatrix} \frac{f_0}{f_1} x \\ \frac{f_0}{f_1} y \\ \alpha + \frac{\beta}{f_1} \\ 1 \end{bmatrix}$$

Unlike for $z = f_0$ plane, points on the $z = f_1$ plane undergo a change in their (x, y) values. In particular, since $\frac{f_0}{f_1} < 1$ the values of (x, y) are scaled down.[1]

Also notice that since we are also requiring that these points stay in the $z = f_1$ plane, we have

$$\alpha + \frac{\beta}{f_1} = f_1.$$

The two cases $(z = f_0, f_1)$ give us two linear equations in two unknowns (α and β). Solving, we get $\alpha = f_0 + f_1$ and $\beta = -f_0 f_1$, and this defines the following transformation:

$$\begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & f_0 + f_1 & -f_0 f_1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Verify for yourself that it maps points on the plane $z = f_0$ to points on $z = f_0$, and that it maps points on $z = f_1$ to $z = f_1$.

[1] This reflects the familiar projective phenomenon that objects that are further away from the camera appear smaller in the image.