

Viewing (9/14/6)

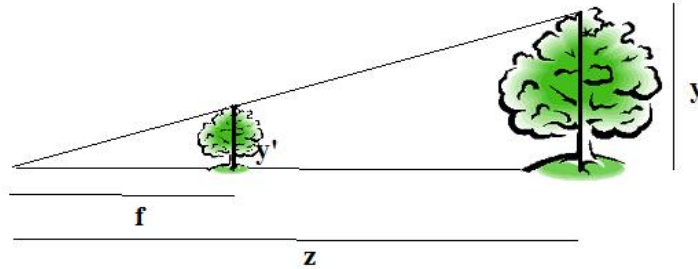
Lecturer: James Kuffner

Scribes: Rosen Diankov

1 Perspective

Perspective transformations convert 3D world coordinates to 2D screen coordinates. This can be thought of as projecting all 3D geometry on a plane then defining a subrectangle of that region and mapping it to the screen. Imagine we're looking at only the y-axis and z-axis and defining the screen to be f away from the eye (Figure 1). Then we can solve for the projected height y' by:

$$\frac{y}{z} = \frac{y'}{f}$$



So the final coordinates of the projected point will be $\bar{\mathbf{v}} = (\frac{xf}{z}, \frac{yf}{z}, f)$. However, we would like a 3x3 linear transformation M such that $M\mathbf{v} = \bar{\mathbf{v}}$ where $\bar{\mathbf{v}} = (x, y, z)$. This is impossible because the operation is non-linear. Instead we can work in 4D and use a 4th homogeneous coordinate to get the implicit division. Remember that $(x, y, z, w) = (\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1)$. So all we have to do is create some matrix M such that

$$M \begin{pmatrix} x & y & z & 1 \end{pmatrix} = \begin{pmatrix} x & y & z & \frac{z}{f} \end{pmatrix}$$

Since the final vector is linear in the parameters, such a linear transformation exists and is given by:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{pmatrix}$$

The rank of M is 3 because the last column is all zeros, therefore projections cannot be inverted. Traditionally, the transformation pipeline for an object looks like:

- Vector Data - object coordinate system
- Model/View Transform - `glMatrixMode(GL_MODELVIEW)`, produces eye coordinates
- Projection - `glMatrixMode(GL_PROJECTION)`, produces clip coordinates (or projection coordinates)
- Viewport Transform - `glViewport(x, y, width, height)`. maps the clip coordinates (usually $[-1, 1] \times [-1, 1]$) to $[0, width - 1] \times [0, height - 1]$

The real screen is only a subrectangle of the whole clip coordinate space, therefore all the geometry out of it must be rejected. Sometimes a polygon can intersect one or more planes, so the clipping cannot just reject or accept, it must compute the exact intersection between the polygon and the subrectangle. The concept is simple: for every edge, compute the part of the polygon that is in front of it. Repeat this for all edges or until the polygon is reduced to a line or a point. If all the polygon's points are behind even one edge, the whole polygon can be rejected. If some points are in front and some points are behind the polygon is clipped and passed to the next edge. Repeating this for all edges generates the correct view clipping (also called Cohen-Sutherland algorithm). There are many ways to speed this up by keeping track of clip regions.

One detail we omitted is the depth value of the pixel, which is necessary for the z-buffer to work correctly. Instead of ignoring the z-coordinate after projection, it is saved and passed to the clipping stage. The view frustum defines near and far planes that define clipping on the z direction. Any points behind the near plane or farther than the far plane get clipped. With depth combined, the clip algorithm clips against a box, so it uses planes rather than edges; however, all the math works the same.

2 Normals

Normals are not transformed the same way as points. For example, normals are not translated, but points are. Usually the homogeneous coordinate of normals is 0, while that of positions is 1. This helps the translation component, but there are still cases where the angles between normals and the real normals to the faces differ. In math, this means $((M\mathbf{n}) \cdot (M(\mathbf{p}_2 - \mathbf{p}_1))) \neq \mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1)$ where M is the linear transformation, \mathbf{p}_2 and \mathbf{p}_1 are positions, and \mathbf{n} is a normal vector. To derive the correct equation for normal transformations, we compute

$$\begin{aligned} \mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1) &= \\ \mathbf{n}^T (\mathbf{p}_2 - \mathbf{p}_1) &= \\ \mathbf{n}^T (M^{-1}M)(\mathbf{p}_2 - \mathbf{p}_1) &= \\ ((M^{-1})^T \mathbf{n}) \cdot M(\mathbf{p}_2 - \mathbf{p}_1) & \end{aligned}$$

So whenever transforming points by M , transform normals by $(M^{-1})^T$. For rigid transformations, you can use M instead since $R^{-1} = R^T$ for rotation matrices.