Lecture 4

Up to now we have defined points in a 3D world coordinate space and we have considered how to transform into a 3D camera coordinate space. Our next step is to project the points into an image. Images are 2D, not 3D, and so we need to map 3D points to a 2D surface (a plane).

Orthographic projection

A first guess on how to do this is to just throw away the z coordinate and leave the x and y values as they are. This is called the *orthographic* projection in the z direction. We can write this as a mapping to the $Z = 0$ plane. This mapping can be represented by a 4×4 matrix:

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

This is called an orthographic projection.

More generally, an *orthographic projection* is a mapping from 3D to a plane such that each point (x, y, z) is mapping in a direction that is parallel to the normal to the plane. The "view from above" and "view from the side" sketches shown later in this lecture can be thought of an orthographic projection in the y and x directions, respectively.

In class (see the slides), I presented a number of visual examples of orthographic projections, including the easy-to-think about x, y, and z orthographic projections. I also presented a projection onto the plane $x + y + z = 0$. This projection has a special name: an *isometric* projection. The name comes from the fact that the projected unit vectors (x, y, z axes of some object *e.g.* a cube) all have the same length in the image.

Parallel Projection

A slightly more general way to project the world onto the $z = 0$ plane is to project in direction $\mathbf{p} = (p_x, p_y, p_z)$. We assume $p_z \neq 0$ in order for this to make sense.

How do we project in direction $\mathbf{p}$? Consider any 3D point (x, y, z). Suppose this point projects to $(x^*, y^*, 0)$, i.e.

$$(x, y, z) - t(p_x, p_y, p_z) = (x^*, y^*, 0)$$

for some t. To compute x^*, y^*, we solve for t first:

$$z - tp_z = 0$$

so $t = \frac{z}{p_z}$. Thus,

$$x^* = x - (p_x/p_z)z$$
$$y^* = y - (p_y/p_z)z$$

In homogeneous coordinates, we can represent this projection as follows:

$$\begin{bmatrix} x^* \\ y^* \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -p_x/p_z & 0 \\ 0 & 1 & -p_y/p_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Note we really don't *need* homogeneous coordinates here. i.e. the fourth row and fourth column don't do anything interesting. But I want to be consistent since pretty much everything we do from here on is going to use homogeneous coordinates.

Note that orthographic projection is the special case of parallel projection in which $p_x = p_y = 0$. Any other case of parallel projection is called *oblique* projection. In class, I discussed briefly two common types of oblique projections, namely *cavalier* and *cabinet*. The latter in particular often are used in furniture illustrations e.g. instructions for assembling Ikea furniture.

Let's now turn to a model which more closely approximates the image formation that we experience with our eyes and with cameras.

Perspective Projection

The standard model of image formation that is used in computer graphics is that a 3D scene is projected towards a single point – called the *center of projection*. This center of projection is just the position of the camera.

The image is not defined at the projection point, but rather it is defined on a plane, called the *projection plane*. The projection plane is perpendicular to the camera z axis. For real cameras, the projection plane is the sensor plane which lies behind the camera lens. The same is true for eyes, although the projection surface on the back of your eye (the retina) isn't a plane but rather it is the spherically curved surface. In both of these cases, the image of the scene is upside down and backwards on the projection surface.

In computer graphics we define a projection plane to lie in front of the center of projection. (See illustrations on next page.) In camera coordinates, the center of projection is $(0,0,0)$ and the projection plane is $z = f$ where $f < 0$ since we are using right handed coordinates. A 3D point (x,y,z) is projected to (x^*, y^*, f). Using similar triangles, we observe that the projection should satisfy:

$$\frac{x}{z} = \frac{x^*}{f} \qquad \frac{y}{z} = \frac{y^*}{f}$$

and so the projection maps the 3D point (x,y,z) to a point on the projection plane,

$$(x,y,z) \rightarrow (f\ \frac{x}{z}, f\ \frac{y}{z}, f).$$

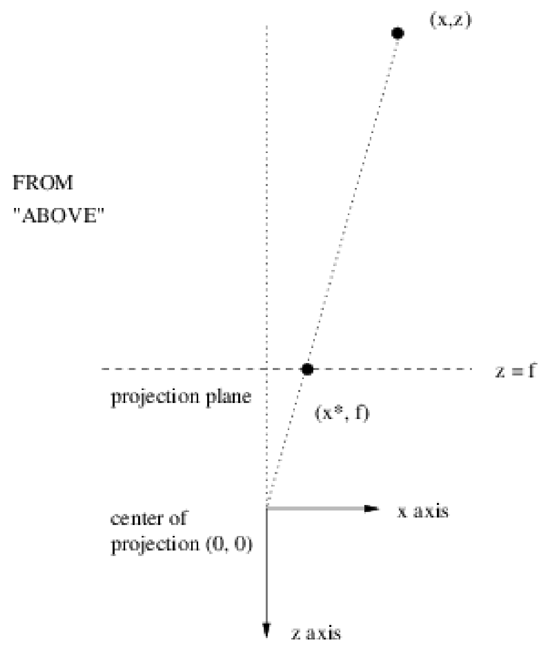
Using homogeneous coordinates to define a projection

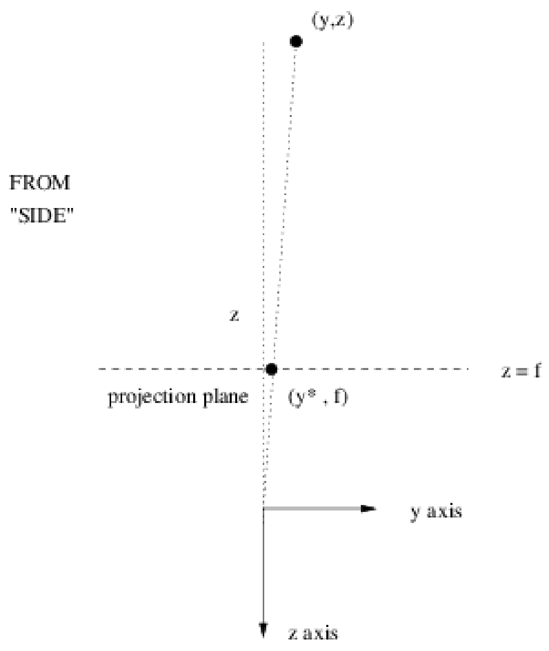
The homogeneous coordinate representation allows us to write projection mapping. For any $z \neq 0$, we can re-write our projected points as:

$$(f\frac{x}{z}, f\frac{y}{z}, f, 1) \equiv (xf, yf, fz, z).$$

This allows us to represent the projection transformation by a 4×4 matrix, i.e. :

$$\begin{bmatrix} fx \\ fy \\ fz \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$





A few observations can be made. First, since we are now treating all 4D points (wx, wy, wz, w) as representing the same 3D point (x, y, z), we can multiply any 4×4 transformation matrix by a constant, without changing the transformation that is carried out by the matrix. For example, the above matrix can be divided by the constant f and written instead as:

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

This matrix, like the one above, projects the 3D scene onto the plane $z = f$, with center of projection being the origin $(0, 0, 0)$.

A second observation (which I neglected to mention in class) is that, whereas the 4×4 translation, rotation, and scaling matrices were invertible (and hence of rank 4), the two projection matrices are not. They are of rank 3 since the third and fourth rows are linearly dependent.

Vanishing points

One well known concept from perspective geometry is that parallel lines in the scene intersect at a common point in the image plane. The most familiar example is two train tracks converging at the horizon. The point where the lines intersect is called a *vanishing point*.

How can we express vanishing points mathematically? Consider a family of lines of the form,

$$\{(x_0, y_0, z_0) + t(v_x, v_y, v_z)\}$$

where (v_x, v_y, v_z) is some non-zero constant vector. For any point (x_0, y_0, z_0), we get a line in the direction of this vector (v_x, v_y, v_z). So for a bunch of different points, (x_0, y_0, z_0), we get a bunch of parallel lines. What happens along each of the lines as we let $t \to \infty$?

We write points in homogeneous coordinates as:

$$\{(x_0 + tv_x, y_0 + tv_y, z_0 + tv_z, 1)\}$$

Now dividing by t and taking the limit gives

$$\lim_{t\to\infty}(\frac{x_0}{t} + v_x, \frac{y_0}{t} + v_y, \frac{z_0}{t} + v_z, \frac{1}{t}) = (v_x, v_y, v_z, 0)$$

so all the lines go to the same point at infinity. Thus, when we project the lines into the image, the lines all intersect at the image projection of the point at infinity.

What happens when we use perspective projection to project a point at infinity $(v_x, v_y, v_z, 0)$ onto a projection plane?

$$\begin{bmatrix} fv_x \\ fv_y \\ fv_z \\ v_z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

The projected point is called the *vanishing point* in the image. If $v_z \neq 0$, then the projected point written as 3D vector is $(\frac{v_x}{v_z}f, \frac{v_y}{v_z}f, f)$. This is clearly a finite point. However, if $v_z = 0$, then the projected point itself is a point at infinity.

If you took an art class in high school, you may have learned about *n point perspective.* You may remember this has something to do with vanishing points. (See the slides for example images.) With the above discussion, we can now define n-point perspective in terms of points at infinity.

N-point perspective refers to an image of a manmade scene or object that contains multiple sets of parallel lines. Typically the object or scene has a natural xyz coordinate system and the lines in the object/scene are aligned with these coordinate axes, but this is not a requirement. The requirement is just that the scene contains at least one set of parallel lines.

We say an image has n-point perspective if there are *n finite* vanishing points in the image projection plane. The 'finite vanishing point' condition is equivalent to saying that the parallel lines have a direction vector (v_x, v_y, v_z) such that $v_z \neq 0$. Parallel lines in directions $(1, 0, 0)$ and $(0, 1, 0)$, in particular, do *not* have this property. These vectors produce vanishing points at infinity. See slides for examples of 1–, 2–, and 3– point perspective.

View Volume (Frustum)

We have considered how, in perspective projection, points in a 3D scene project onto a 2D plane. We considered all 3D points. Note, however, that only part of the scene needs to be considered. For example, points behind the camera are not visible and hence need not be considered. In addition, many points in front of the camera also need not be considered. For example, typical cameras have an angular field of view of between 30 and 60 degrees only (depending on whether the lens is telephoto, regular, or wide angle). Let's next formalize this notion of field of view of an image and show how to express it using projective transformations and homogeneous coordinates.

To make an image of a scene, we want to consider only scene points that (1) project toward the origin, and (2) project to a finite rectangular *window* in the projection plane, and (3) lie over a given range of z values, i.e. "depths". This set of points is called the *view volume* or *view frustum.*[1]

[1]According to wikipedia, a "frustum" is the portion of a solid normally a cone or pyramid which lies between two parallel planes cutting the solid. Note the word is "frustum", not "frustrum".

The shape of the view volume is a truncated pyramid. (See the slides for illustrations.) We will only want to draw those parts of the scene that lie within this view volume. Those points that lie outside the view volume are not part of our picture. They are "clipped." I will say more about how clipping works next lecture.

Let's now define the view frustrum. Let xyz be camera coordinates. Define the sides of the view volume by four sloped planes,

$$\frac{x}{z} = \pm \tan(\frac{\theta_x}{2})$$

$$\frac{y}{z} = \pm \tan(\frac{\theta_y}{2})$$

where θ_x and θ_y define angular field of view in the x and y directions. We also restrict ourselves to a range of depths lying between some near and far z plane. Keep in mind that we care only about points where $z < 0$ i.e. the camera is looking in the $-z$ direction. Let near and far be the (positive) distances to the planes. Then

$$0 \leq \text{near} \leq -z \leq \text{far}$$

or if you prefer

$$-\text{far} \leq z \leq -\text{near} \leq 0.$$

In OpenGL, rather than defining such a view volume by the parameters $\{\theta_x, \theta_y, \text{near}, \text{far}\}$, the view volume is defined with parameters as follows:

$$\texttt{gluPerspective}(\texttt{theta\_y}, \texttt{aspect}, \texttt{near}, \texttt{far}).$$

The third and fourth parameters do as expected. It is the first and second parameters that require some thought. The first parameter specifies θ_y in degrees. The second parameter specifies the *aspect ratio*

$$\text{aspect} \equiv \frac{\tan(\theta_x/2)}{\tan(\theta_y/2)}.$$

Note that this is *not* the ratio of the angles, but rather it is the ratio of x size to y size of the rectangle that is defined by any depth slice through the frustrum. (Typically – but not necessarily – the aspect ratio is defined to match that of the window on a display where the user knows the x and y size of the display in pixels.)

An equivalent way to define the above view volume is to choose a depth plane, say the near plane, and define a square on the plane which corresponds to the angles θ_x and θ_y. This square is also defined by the intersection of the four sloped planes with the near plane, so

$$x = \pm\text{near} \tan(\frac{\theta_x}{2})$$

$$y = \pm\text{near} \tan(\frac{\theta_y}{2})$$

The x values are the left and right edge of the square, and the y values here are the bottom and top of the square. (Of course, the far value is also needed to complete the definition of the view volume.)

OpenGL allows a slightly more general definition that this. One may define any rectangle in the near plane such that the sides of the rectangle are aligned with the x and y axes. One uses:

$$\texttt{glFrustum}(\texttt{left}, \texttt{right}, \texttt{bottom}, \texttt{top}, \texttt{near}, \texttt{far}).$$

Here the variables `left` and `right` refer to the limits of x on the near plane ($z = f_0 = -\texttt{near}$) and `bottom` and `top` refer to the limits of y on the near plane. Note that this is a more general definition than `gluPerspective`, which essentially required that `left = -right` and `bottom = -top`,

You may wonder why anyone would need the generality of the `glFrustum`. One nice application is to create 3D desktop displays such as stereo displays (like 3D movies). For stereo displays, you need to consider that there are two eyes looking at (through) the same image window, and so the position of the window differs in the coordinate systems of the two eyes. See the slides for an illustration. Another nice example is shown in this interesting video: `https://www.youtube.com/watch?v=Jd3-eiid-Uw`. Again, the basic idea is to consider the frame of your display monitor to be a window through which you view a 3D world. As you move, the position of the display changes with respect to your own position. If you could get a external camera or some other "motion capture" system to track your position when viewing the display, then the frustum could be defined by the relationship between the display frame and your (changing) viewing position. [I don't expect you to understand the mathematical details of this method from my brief summary here, or even from watching the video. I just want you to get the flavor.]