Last lecture we introduced the homogeneous coordinate representation for 3D points. At first glance, this representation seems to be just a trick for treating translations and rotations and scaling in a common format, namely multiplcation by a 4×4 matrix. But homogenous coordinates representations allow one to do much more than that. (If you have taken a 3D computer graphics course, then you have seen a much wider set of applications.)

We next consider how homogeneous coordintes allow one to represent points that are infinitely far away, and in some direction. Thus turns out to be useful for representing direction vectors.

Points at infinity

We have considered points (wX, wY, wZ, w) under the condition that $w \neq 0$, and such a 4D homogeneous coordinate point is always associated with the same $(X, Y, Z) \in \Re^3$.

Consider any $(X, Y, Z) \neq 0$, that is, not the origin. Let $s > 0$ Then, $(sX, sY, sZ, 1)$ represents a point on the line from the origin through (X, Y, Z) that is s times as far away from the origin as the point (X, Y, Z). We can write this point as:

$$(sX, sY, sZ, 1) \equiv (X, Y, Z, \frac{1}{s}).$$

If we let $s \to \infty$, then the corresponding 3D point stays along the line from the origin through the point (X, Y, Z) but gets infinitely far from the origin. In the limit, we have

$$lim_{s \to \infty} \ (sX, sY, sZ, 1) = (X, Y, Z, 0)$$

and so we can identify homogeneous coordinate point $(X, Y, Z, 0)$ with a 3D point at infinity in direction (X, Y, Z).

What happens to a point at infinity when we perform a rotation, translation, or scaling? Recall that the bottom row of each of these 4×4 matrices is (0,0,0,1), it is easy to see that these transformations map points at infinity to points at infinity. In particular,

- a translation matrix does not affect a point at infinity; i.e. it behaves the same as the identity matrix;
- a rotation matrix maps a point at infinity in exactly the same way it maps a finite point, namely, $(X, Y, Z, 1)$ rotates to $(X', Y', Z', 1)$ if and only if $(X, Y, Z, 0)$ rotates to $(X', Y', Z', 0)$.
- a scale matrix maps a point at infinity in exactly the same way it maps a finite point, namely, $(X, Y, Z, 1)$ maps to $(\sigma_X X, \sigma_Y Y, \sigma_Z Z, 1)$ if and only if $(X, Y, Z, 0)$ scales to $(\sigma_X X, \sigma_Y Y, \sigma_Z Z, 0)$.

We sometimes interpret points at infinity as *direction vectors*, that is, they have a direction but no position. One must be careful in refering to them in this way, though, since vectors have a length whereas points at infinity $(X, Y, Z, 0)$ do not have a length.

Homogenous coordinates in 2D

One can use homogenous coordinates to represent points (and points at infinity) in any n-D space. For example, we will often use homogeneous coordinates for 2D spaces. Suppose we wish to translate all points (x, y) by adding some constant vector (t_x, t_y). This transformation cannot be achieved by a 2×2 matrix, so we tack on a third coordinate with value 1 – i.e. $(x, y, 1)$ – and translate by performing a matrix multiplication:

$$\begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

We can perform rotations using a 3×3 matrix as well, namely a 2D rotations matrix goes into the upper-left 2×2 corner:

$$\begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

and we can also scale the two coordinates:

$$\begin{bmatrix} \sigma_x x \\ \sigma_y y \\ 1 \end{bmatrix} = \begin{bmatrix} \sigma_X & 0 & 0 \\ 0 & \sigma_Y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

And we can also perform a shear

$$\begin{bmatrix} x + sy \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

The arguments above about points at infinity are exactly the same for 2D points (x, y). Consider a point $(sx, sy, 1)$ in homogeneous coordinate representation, and ask what happens as $s \to \infty$. We get:

$$lim_{s\to\infty}(sx, sy, 1) = lim_{s\to\infty}\ (x, y, \frac{1}{s}) = (x, y, 0)$$

which is in the direction of (x, y) but is infinitely far from the origin. Thus, $(x, y, 0)$ represents a 2D point at infinity in direction of vector (x, y).

World vs. camera coordinates

Up to now, we have defined 3D points in the camera coordinate system. Often we would also like to express the position of points (X, Y, Z) in a world coordinate system which is independent of the camera. This should make sense intuitively: we would like to be able to speak about the positions of objects independently of where the camera is.

If we are going to express point positions both in camera and in world coordinates, then we will need a way of transforming between these coordinate systems. We need to define a transformation from point (X_w, Y_w, Z_w) in world coordinates to its location (X_c, Y_c, Z_c) in the camera coordinates. We will use 4×4 translation and rotation matrix to do so.

Camera extrinsic (or external) parameters

Suppose the position of the camera's center in world coordinates is a 3D point $\mathbf{C}_w$. If we wish to transform any other point $\mathbf{X}_w$ into the camera's coordinate system, we first subtract off $\mathbf{C}_w$ and then we perform a rotation:

$$\mathbf{X}_c = \mathbf{R}(\mathbf{X}_w - \mathbf{C}_w) \ .$$

Use 3D vectors and 3×3 matrices, we can write this as follows,

$$\mathbf{X}_c = \mathbf{R}\mathbf{X}_w - \mathbf{R}\mathbf{C}_w.$$

In homogeneous coordinates, we would write

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\mathbf{C}_w \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} .$$

so the transformation from world to camera coordinates is the product of a 4×4 translation matrix and a 4×4 rotation matrix.

$$\begin{bmatrix} \mathbf{R} & -\mathbf{R}\mathbf{C}_w \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{C}_w \\ 0 & 1 \end{bmatrix}$$

where $\mathbf{I}$ is the 3×3 identity matrix.

The rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{C}_w$ define the camera's extrinsic coordinates, namely its orientation and position, respectively, in world coordinates. The matrix $\mathbf{R}$ transforms from world to camera coordinates, and so you can think of it as $\mathbf{R}_{c \leftarrow w}$.

Camera intrinsics (internal parameters)

We now have all scene points in camera coordinates, and we can think about how the image is formed.

Projection matrix (3D to 2D)

We next project from camera coordinates into the image plane. Recall from lecture 14 that we defined the projection from 3D (X, Y, Z) into the 2D coordinates (x, y) via

$$(x, y) = (f\frac{X}{Z}, f\frac{Y}{Z}).$$

Let's rewrite this 2D point in homogeneous coordinates:

$$(x, y, 1) = (f\frac{X}{Z}, f\frac{Y}{Z}, 1)$$

If $Z \neq 0$ then

$$(f\frac{X}{Z}, f\frac{Y}{Z}, 1) \equiv (fX, fY, Z).$$

This equivalence allows us to write the projection from a 3D point (X, Y, Z) (which is in camera coordinates) to its 2D image point (x, y), using a 3×4 matrix:

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Note that the fourth column doesn't contribute anything. For this reason it is common to use a 3×3 matrix rather than a 3×4 and to write the projection transformation as follows:

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Transforming to pixel coordinates

When we are working with real cameras, typically we do not want to index points on the projection plane by their actual position on the projection plane (measured in say millimeters). Rather we index points by pixel coordinates. Transforming from real physical positions to pixel positions involves a scaling. It also involves a translation, as we'll see below. But let's first deal with the scaling.

Let's write out the transformation from mm units on the projection plane to pixel units. We multiply the x, y values by the *number of pixels per mm* in the x and y direction, respectively. These scale factors are denoted m_x, m_y. Surprisingly, m_x and m_y are not always exactly the same value, though they are usually very close. The projection followed by scaling transformation is now:

$$\begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

which can be written

$$\begin{bmatrix} fm_x & 0 & 0 & 0 \\ 0 & fm_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Note that the 4^{th} column can be dropped if we use (X, Y, Z) for the scene point (in camera coordinates) rather than the homogeneous coordinate representation $(X, Y, Z, 1)$.

We mentioned above that it is also necessary to do a translation of the pixel positions. To understand why, note that the position of the optical center, $(x, y) = 0$, on the sensor typically doesn't correspond exactly to physical center of the image sensor. (The position of the optical center is called the *principal point*.) In particular, the principal point doesn't necessarily correspond exactly to the center of the pixel grid. For example, if the grid had 1536 rows and 2048 columns, then the center of the pixel grid would be halfway between row 767 and 768 and column 1023 and 1024 (assuming we start our count from 1). Because of manufacturing limitations, the principal point might be shifted slightly.

Let (p_x, p_y) be the position of the principal point *in pixel coordinates*. Then, in order for the principal point $(x, y) = (0, 0)$ to map to (p_x, p_y), we need to translate:

$$\begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

This gives us a sequence (projection, scaling, translation):

$$\begin{bmatrix} fm_x & 0 & p_x & 0 \\ 0 & fm_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A few more details...

- While the above model typically describes all the parameters needed to transform from a scene point in camera coordinates into the projected image point in pixel coordinates, one sometimes adds another parameter called the *skew parameter* s. So a more general transformation is:

$$\mathbf{K} \equiv \begin{bmatrix} fm_x & s & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

  This matrix $\mathbf{K}$ is commonly called the *camera calibration matrix*. In a few lectures from now, we will show how to estimate this matrix.

  Notice that the parameters f, m_x, m_y only appear as two products in this matrix. Sometimes one writes

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix}.$$

- We now have a transformation takes a point $(X, Y, Z, 1)$ in world coordinates to a pixel position (wx, wy, w). One often writes this transformation as:

$$\mathbf{P} \;=\; \mathbf{K}\,\mathbf{R}\,[\,\mathbf{I}\,|\,-\mathbf{C}\,]$$

  where $\mathbf{K}$, $\mathbf{R}$, and $\mathbf{I}$ are 3×3, and where the matrix on the right is 3×4, and the symbol $|$ is used to separate the 3×3 identify matrix $\mathbf{I}$ from the vector $-\mathbf{C}$, as opposed to a difference **I - C** which would make no sense. This transformation is sometimes called a *finite projective camera*. It describes the mapping from points in world coordinates to points in pixel coordinates.

