Last class we considered smooth translations and rotations of the camera coordinate system and the resulting motions of points in the image projection plane. These two transformations were expressed mathematically in a slightly different way. A translation was expressed by a vector subtraction, and a rotation was expressed by a matrix multiplication.

Introduction to Homogeneous coordinates

Today we will learn about an alternative way to represent translations and rotations which allows them both to be expressed as a matrix multiplication. This has certain computational and notational conveniences, and eventually leads us to a wider and interesting class of transformations.

Suppose we wish to translate all points (X, Y, Z) by adding some constant vector (t_x, t_y, t_z) to all coordinates. So how do we do it? The trick is to write out scene points (X, Y, Z) as points in $\Re^4$, and we do so by tacking on a fourth coordinate with value 1. We can then translate by performing a matrix multiplication:

$$\begin{bmatrix} X + t_x \\ Y + t_y \\ Z + t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}.$$

We can perform rotations using a 4×4 matrix as well. Rotations matrices go into the upper-left 3×3 corner of the 4×4 matrix:

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

i.e. nothing interesting happens in the fourth row or column. Last lecture we looked at rotations about the X, Y, Z axes only. We will discuss other rotation matrices later today.

Notice that we can also scale the three coordinates:

$$\begin{bmatrix} \sigma_X X \\ \sigma_Y Y \\ \sigma_Z Z \\ 1 \end{bmatrix} = \begin{bmatrix} \sigma_X & 0 & 0 & 0 \\ 0 & \sigma_Y & 0 & 0 \\ 0 & 0 & \sigma_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}.$$

This scaling doesn't come up so much in 3D, but it does come up when we do 2D transformations, as we will see later today.

We have represented a 3D point (X, Y, Z) as a point $(X, Y, Z, 1)$ in $\Re^4$. We now generalize this by allowing ourselves to represent (X, Y, Z) as *any* 4D vector of the form (wX, wY, wZ, w) where $w \neq 0$. Note that the set of points

$$\{ \ (wX, wY, wZ, w) \ : \ w \neq 0 \ \}$$

is a line in $\Re^4$ that passes through the origin and through the point $(X, Y, Z, 1)$ in $\Re^4$. That is, we are associating each point in $\Re^3$ with a line in $\Re^4$, in particular, a line that passes through the origin. This representation is called *homogeneous coordinates*.

Is this generalization consistent with the 4×4 rotation, translation, and scaling matrices which we introduced above? Yes, it is. For any $\mathbf{X} = (X, Y, Z, 1)$ and indeed for any 4×4 matrix $\mathbf{M}$, we have

$$w(\mathbf{MX}) \equiv \mathbf{M}(w\mathbf{X}),$$

Thus, multiplying each component of the vector $\mathbf{X}$ by w and transforming by $\mathbf{M}$ yields the same vector as transforming $\mathbf{X}$ itself by $\mathbf{M}$ and then muliplying each component of $\mathbf{M}$ $\mathbf{X}$ by w. But multiplying each component of a 4D vector $\mathbf{X}$ by w doesn't change how that vector is transformed.

Points at infinity

We have considered points (wX, wY, wZ, w) under the condition that $w \neq 0$, and such a point is associated with $(X, Y, Z) \in \Re^3$. What happens if we let $w = 0$? Does this mean anything? Consider (X, Y, Z, ϵ) where $\epsilon > 0$, and so

$$(X, Y, Z, \epsilon) \equiv (\frac{X}{\epsilon}, \frac{Y}{\epsilon}, \frac{Z}{\epsilon}, 1).$$

If we let $\epsilon \to 0$, then the corresponding 3D point $(\frac{X}{\epsilon}, \frac{Y}{\epsilon}, \frac{Z}{\epsilon})$ goes to infinity, and stays along the line from the origin through the point $(X, Y, Z, 1)$. We thus identify the limit $(X, Y, Z, 0)$ as $\epsilon \to 0$ with a 3D point at infinity, namely a point in direction (X, Y, Z).

What happens to a point at infinity when we perform a rotation, translation, or scaling? Since 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 for 1D or 2D or higher dimensional spaces as well. For example, take 2D. 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 $(x, y, 1)$, 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:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and we can also scale the two coordinates:

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

or perform a shear *(not mentioned in class)*

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

The previous arguments about points at infinity are exactly the same for 2D points (x, y). Consider a point (x, y, ϵ) and ask what happens as $\epsilon \to 0$. We get $(\frac{x}{\epsilon}, \frac{y}{\epsilon}, 1)$ which is in the direction of (x, y) but which goes to an infinite distance from the origin as $\epsilon \to 0$. Thus, $(x, y, 0)$ represents a 2D point at infinity in direction of vector (x, y).

World vs. camera coordinates

Last lecture, we defined 3D points in the camera coordinate system. However, 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.

Rotation matrices [Note: change this in 2010 – consider reflections too!]

Earlier this lecture we mentioned rotation matrices but didn't spell out any details. Now we consider a rotation matrix whose rows define the directions of the camera's canonical axes in world coordinates. The three rows are **u**,**v**,**n** are orthonormal unit vectors. The third row $\mathbf{n} = (n_x, n_y, n_z)$ is the direction of the optical axis ("Z axis") of the camera, represented in world coordinates. The other two rows are the camera's x and y axes which are perpendicular to the n vector. These define the "left-right" and "up-down" directions of the camera. We can write the 3×3 rotation matrix as:

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{bmatrix}.$$

Note that this matrix maps **u**, **v**, **n** to (1,0,0), (0,1,0), (0,0,1) respectively.

What if we write this rotation using a 4×4 matrix ?

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

This matrix now maps the direction vectors $\mathbf{u}$, $\mathbf{v}$, $\mathbf{n}$ (written with a fourth coordinate which is 0) to the direction vectors $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, respectively.

Camera extrinsic (or external) parameters

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

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

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

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

Alternatively, we could use homogeneous coordinates, and write

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

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

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

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

The rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{C}$ define the camera's extrinsic coordinates, namely its orientation and position, respectively, in world coordinates.

Projection matrix (3D to 2D)

Recall from lecture 1 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 now rewrite this 2D point in homogeneous coordinates:

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

and note that 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:

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

The projection matrix is not invertible. It is of rank 3. In particular, because the matrix above maps a 4D space to a 3D space, there must be some non-zeo vector that maps to $(0, 0, 0)$. This vector defines the *null space* of the projection. What is this null vector? Answer: by inspection, it is the center of projection (0,0,0,1).

Note that the point $(0, 0, 0)$ is undefined, where interpreted as a 2D point in homogeneous coordinates. It is neither a finite point $(x, y, 1)$, nor a point at infinity $(x, y, 0)$, where I am assuming here that at least one of x, y is non-zero.

Camera intrinsic (or internal) parameters

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 want to index points by pixel numbers. Transforming from real physical positions to pixel positions involves a scaling (since the width of a pixel is typically not a standard unit like 1 mm) and a translation (since the principal point[1] typically doesn't correspond exactly to center of the image sensor).

The sensor is made up of a square grid of sensor elements (pixels). We can talk about the position which is the center of this grid. For example, if the grid had 1536 rows and 2048 columns, then we would be talking about the position on the sensor that is halfway between row 767 and 768 and column 1023 and 1024 (assuming we start our count from 0).

We define a coordinate system on the grid in whatever units we want, say millimeters. We define the coordinate system such that the origin is at center of the grid and the (u, v) axes are parallel to the camera's x, y axes, respectively.

We have expressed the position on the sensor plane in physical coordinates, in units of millimeters. What we want instead is to express it in terms of pixel units. We can do this by scaling. We multiply the x, y coordinates by the *number of pixels per mm* in the x and y direction. These scale factors are denoted m_x, m_y. Surprisingly, these are not always exactly the same value, though they are usually very close. The projection and scaling transformation is now:

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

[1]The *principal point* is the intersection of the optical axis with the image plane

The transformation gives positions in terms of pixel units. However, it does not correspond to the pixel indices used in an image. Typically the index (0,0) is in the corner of the image. Thus we need another transformation, namely a translation, that gives the correct pixel coordinates of the principal point. Note: the principal point might not correspond exactly to the center of the sensor. You would think it should, but keep in mind that pixels are very small and the placement of the sensor relative to the lens (which defines the optical axis and hence principal point) might not be as accurate as you hope.

Let (p_x, p_y) be the position of the principal point *in pixel coordinates*. It may indeed be at the center of the image, i.e. at position (*,*) as mentioned above. But it may be at a slightly different position too. Then, to get the position of a projected point (x, y) in pixel coordinates, we need to add a translation component which is the pixel position of the principal point:

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

A few more details...

- While the above model typically describes all the parameters needs to transform from a scene point in camera coordinates into the projected image point in pixel coordinates, it is common to add yet one more parameter called the *skew parameter* s. So a more general projection transformation is:

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

- It is common to give the 3×3 matrix on the left side a special name

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

  is called the *camera calibration matrix*. In part 3 of this course, 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

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

  where α_x and α_y is the focal length, expressed in horizontal and vertical pixel units, respectively.

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

$$\mathbf{P} = \mathbf{KR}[\ \mathbf{I}\ |\ -\mathbf{C}]$$

  where $\mathbf{K}$, $\mathbf{R}$, and $\mathbf{I}$ are 3×3, and where the matrix on the right is 3×4. This transformation is sometimes called a *finite projective camera*.