Rotation

Last lecture we addressed camera translation. Today we will look at camera rotation. We will first look at continuous rotation. For example, moving cameras may pan (side to side), pitch (up and down), or roll (rotate about the optical axis). We will then look at discrete finite rotations, for example when a camera rotates to a different orientation and we have two images – one from the original orientation and one from the new orientation.

To understand the distinction between discrete and continuous rotation, it is helpful to recall rotation in two dimensions, say (X, Y). A discrete counter-clockwise rotation by θ can be expressed:

$$\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}.$$

If we write $\theta = \Omega t$ to be a continuous rotation (with angular velocity Ω) we get:

$$\begin{bmatrix} X(t) \\ Y(t) \end{bmatrix} = \begin{bmatrix} \cos(\Omega t) & -\sin(\Omega t) \\ \sin(\Omega t) & \cos(\Omega t) \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}.$$

For any point (X_0, Y_0), this traces out a circle over time.

Continous rotations

Let's now extend this idea to continous 3D rotations. Consider the camera rotation about the Z axis. Let the angular velocity be Ω. Then,

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} \cos(\Omega t) & -\sin(\Omega t) & 0 \\ \sin(\Omega t) & \cos(\Omega t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

The point (X_0, Y_0, Z_0) travels on a circle, centered on the Z axis.

If we rotate about the X axis, then any point (X_0, Y_0, Z_0) will sweep out a circle centered on the X axis:

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Omega t) & \sin(\Omega t) \\ 0 & -\sin(\Omega t) & \cos(\Omega t) \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

If we rotate about the Y axis, then any point (X_0, Y_0, Z_0) will sweep out a circle centered on the Y axis:

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} \cos(\Omega t) & 0 & \sin(\Omega t) \\ 0 & 1 & 0 \\ -\sin(\Omega t) & 0 & \cos(\Omega t) \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

One can write a rotation about an arbitary axis too, but we will not do so.

Next consider what happens when we project the 3D scene into the image. Again we are assuming that the camera is rotating about the X, Y, or Z axis, and the scene is fixed. We can express the image position (x, y) of the projected point as a function of time, using the relationship from last lecture:

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

 ©Michael Langer

We can then compute the (instantaneous) motion field at time $t = 0$ by taking the derivative with respect to time:

$$(v_x, v_y) = \frac{d}{dt}(x(t), y(t))|_{t=0}$$

These derivations use basic calculus, and I will leave it to you as an exercise if you wish.

Rotation about Z axis: roll

In the case of rotation with angular velocity Ω_Z about the camera Z axis (called *roll*), we have:

$$(v_x, v_y) = \Omega_Z(-y, x)$$

This defines a vector field such that the motion is along circles. The image speed of a point is proportional to the radius of the circle on which an image point lies. See the vector field in the figure below left.

Rotation about X axis: pitch

Rotating about the X axis is called *pitch*. The motion field in this case is:

$$(v_x, v_y) = \Omega_X(\frac{xy}{f}, f(1 + (\frac{y}{f})^2)).$$

Near the center of the image projection plane $(x, y) = (0, 0)$, the velocity field is approximately $(0, f\Omega_X)$, namely it is approximately constant and in the y direction only. As (x, y) vary away from 0, second order terms become more important. See the vector field in the figure below center.

Rotation about Y axis: pan

Rotating about the Y axis (called *pan*) gives the

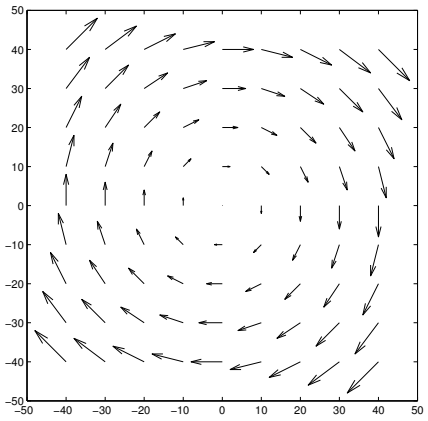
$$(v_x, v_y) \quad = \quad \Omega_Y(f(1 + (\frac{x}{f})^2), \ \frac{xy}{f})$$

This is similar to the pitch field, but now the x and y variables are swapped.

A key difference between the motion fields that are caused by camera translation versus camera rotation is that the translation fields depend on the depth of points in the scene, but *the rotation fields do not*. This is interesting and perhaps surprising. Panning or rolling the camera gives you motion, but the motion tells you *nothing* about how far away the scene points are. To get information about the distance to scene points from motion, you would need to translate the camera.

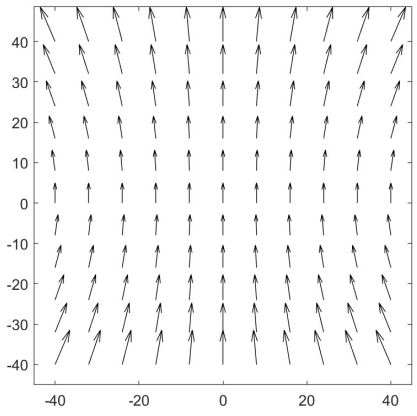
 ©Michael Langer

roll



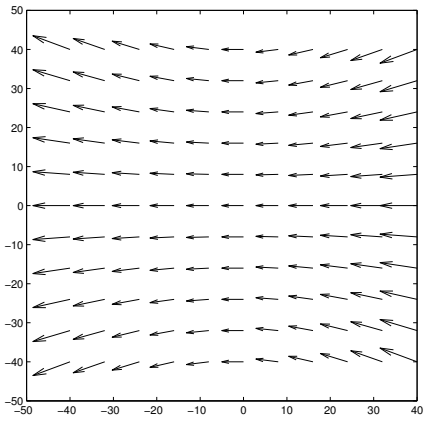
$\boldsymbol{\Omega} = (0, 0, 1)$

tilt



$\boldsymbol{\Omega} = (1, 0, 0)$

pan



$\boldsymbol{\Omega} = (0, 1, 0)$

Cross Product (review from linear algebra)

The **cross product** of two 3D vectors $\mathbf{a}$ and $\mathbf{b}$ is written $\mathbf{a} \times \mathbf{b}$. The cross product is a 3D vector that is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$, and hence to the plane spanned by $\mathbf{a}$ and $\mathbf{b}$ in the case that these two vectors are not parallel.

Given two vectors $\mathbf{a}$ and $\mathbf{b}$, the cross product $\mathbf{a} \times \mathbf{b}$ can be defined as follows. Define the cross product on the unit vectors $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ as follows:

$$\hat{\mathbf{z}} = \hat{\mathbf{x}} \times \hat{\mathbf{y}}$$

$$\hat{\mathbf{x}} = \hat{\mathbf{y}} \times \hat{\mathbf{z}}$$

$$\hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{x}}$$

and we obtain six more equations out of these unit vectors by assuming (as part of the definition) that the following properties hold for all vectors $\mathbf{a}$ and $\mathbf{b}$:

$$\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$$

$$\mathbf{a} \times \mathbf{b} = \mathbf{0} \ .$$

For example, $\hat{\mathbf{x}} \times \hat{\mathbf{x}} = 0$ and $\hat{\mathbf{y}} \times \mathbf{x} = -\hat{\mathbf{z}}$. Finally, we assume that, for any $\mathbf{a}$ and $\mathbf{b}$, the cross product $\mathbf{a} \times \mathbf{b}$ is a linear transformation on $\mathbf{b}$. Writing

$$\mathbf{a} = a_x \hat{\mathbf{x}} + a_y \hat{\mathbf{y}} + a_z \hat{\mathbf{z}}$$

and similarly for $\mathbf{b}$, we use the linearity to expand $\mathbf{a} \times \mathbf{b}$ into nine terms ($9 = 3^2$), only six of which are non-zero. If we group these terms, we get:

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

 ©Michael Langer

Notice that this matrix is not invertible. It has rank 2, not 3. In particular, when $\mathbf{a} = \mathbf{b}$, we get $\mathbf{a} \times \mathbf{b} = \mathbf{0}$.

Later in the course, we will use the notation

$$\mathbf{a} \times \mathbf{b} \equiv [\mathbf{a}]_\times \mathbf{b}$$

where $[\mathbf{a}]_\times$ is the above matrix.

Finite rotations

In general, by a *3D rotation matrix*[1], we will just mean a real 3×3 invertible matrix $\mathbf{R}$ such that $\det \mathbf{R} = 1$ and

$$\mathbf{R}^T = \mathbf{R}^{-1}$$

i.e.

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. Note that the rows and columns of $\mathbf{R}$ are orthogonal to each other and of unit length, i.e. *orthonormal*.

Rotation matrices preserve the length of vectors, as well as the angle between two vectors. To see this, let $\mathbf{p}_1$ and $\mathbf{p}_2$ be two vectors, possibly the same. Then

$$(\mathbf{R}\mathbf{p}_1) \cdot (\mathbf{R}\mathbf{p}_2) = \mathbf{p}_1^T\mathbf{R}^T\mathbf{R}\mathbf{p}_2 = \mathbf{p}_1^T\mathbf{p}_2 = \mathbf{p}_1 \cdot \mathbf{p}_2.$$

If $\mathbf{p}_1 = \mathbf{p}_2$ then we see vector length is preserved. More generally, since the dot product of unit vectors is just the cosine of the angle between them, we see that the angle between vectors is preserved.

Why did we require that $\det \mathbf{R} = 1$? An example of an orthonormal matrix that does *not* have this property is:

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

This matrix performs a mirror reflection of the scene about the yz plane. The length of vectors is preserved, as is the angle between any two vectors. But the matrix is not a rotation.

Another orthonormal matrix that fails the "$\det \mathbf{R} = 1$" condition is

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

which swaps the first two variables. This is a reflection (about the $x = y$ plane) rather than a rotation.

Consider a matrix $\mathbf{R}$ whose rows define the directions of the camera's XYZ axes, written in some other coordinate system. For example, we might have a "world coordinate system" defined by gravity and two other directions, as we discussed last class in the context of vanishing points.

[1] You may recall from your linear algebra background that such matrices are special cases of *unitary* matrices (which can have complex elements and can have determinant -1).

 ©Michael Langer

Let the three rows of $\mathbf{R}$ be $\mathbf{u}$,$\mathbf{v}$,$\mathbf{n}$. The third row $\mathbf{n} = (n_x, n_y, n_z)$ is the unit vector in the direction of the optical axis ("Z axis") of the camera, represented in world coordinates. The other two rows are unit vectors in the direction of 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 $\mathbf{u}$, $\mathbf{v}$, $\mathbf{n}$ to (1,0,0), (0,1,0), (0,0,1) respectively.

Homogeneous coordinates

We next consider 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.

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.

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) as long as $w \neq 0$. Note that the set of points

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

 ©Michael Langer

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 $\mathbf{M}$ and any scalar w, we have

$$w(\mathbf{M}\mathbf{x}) = \mathbf{M}(w\mathbf{x}),$$

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 multiplying 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

$$lim_{\epsilon \to 0}\ (X, Y, Z, \epsilon) = (X, Y, Z, 0)$$

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 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

 ©Michael Langer

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).

 ©Michael Langer