CS 280 Computer Vision
Spring 2018 Efros, Malik, Yu Lecture 4

1 Lecture

Projective Transformations

There is now a notion of points in Euclidean form, and points in projective form. The latter points include a homogeneous coordinate.

Example: representing lines with homogeneous coordinates. Say we have the line $x = 1$ in Euclidean form. We can transform it into projective form $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ as

$$\begin{aligned} x &= 1 \\ x - 1 &= 0 \\ \frac{x_1}{x_3} - 1 &= 0 \\ x_1 - x_3 &= 0 \end{aligned}$$

Similarly, we can represent the line $x = 2$ as $x_1 - 2x_3 = 0$. We can see that the lines $x = 1$ and $x = 2$ now intersect at $x_3 = 0$ (infinity), where formerly they were parallel.

We can represent affine transforms as projective transformations (i.e. with homogeneous coordinates):

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Perspective projection can also be represented in this fashion:

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

Euclidean Transformation: Rotations

Rotations are orthogonal transformations with det = +1. How do we characterize a rotation in 3D space? Rotation matrices have two properties: each column vector should have length 1, and the dot product of any two different columns should be 0. These six constraints leave room for only three degrees of freedom, meaning there are exactly three numbers which characterize a rotation. (Any matrix with nine numbers must be equivalent in some way to these three numbers.) For example, we can represent rotations as

- Euler angles which specify rotations about three axes (roll/pitch/yaw)
- Axis plus amount of rotation
- Quaternions which generalize complex numbers from 2D to 3D (note that a complex number can represent a rotation in 3D). Quaternions reduce redundancy from nine numbers down to four

In general, we will use axis and rotation as the preferred representation of an orthogonal matrix, specifically (s, θ) where s is the unit vector of the axis of rotation (2 degrees of freedom; length doesn't matter) and θ is the amount of rotation (one degree of freedom).

Thus, we should be able to take R (a 3×3 rotation matrix) and convert it to/from an $(\hat{s}, \theta)$ representation. Again, $\hat{s}$ should be the axis as a unit vector in $\mathbb{R}^3$, while θ should be the amount of rotation in radians. **Rodrigues' formula** should enable us to do this.

Here is a trick: we can represent vector products (cross products) by a certain matrix multiplication. Specifically, we can represent them using skew-symmetric matrices. Recall that

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \wedge \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

($\wedge$ takes the cross product.) We can define $\hat{a}$ as

$$\hat{a} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

This is a skew-symmetric matrix (by definition, B is skew-symmetric if $B^T = -B$)! We can see that multiplying $\hat{a}$ by any vector gives

$$\hat{a} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} -a_3 b_2 + a_2 b_3 \\ a_3 b_1 - a_1 b_3 \\ -a_2 b_1 + a_1 b_2 \end{bmatrix} = a \wedge b$$

Now, the equation of motion of a point q is given by $\dot{q} = \omega \wedge q(t)$, where $\dot{q}$ is the time-derivative of q, ω's direction is the axis of rotation, and $\|\omega\|$ specifies the speed of rotation (in rad/s). Rewriting with $\hat{\omega}$, we have $\dot{q}(t) = \hat{\omega} q(t)$. Then, solving, we see that $q(t) = e^{\hat{\omega} t} q(0)$. This solution is the exponential of a matrix, as $\hat{\omega}$ is a matrix.

Note that $e^{\hat{\omega} t} = I + \hat{\omega} t + \frac{(\hat{\omega} t)^2}{2!} + \frac{(\hat{\omega} t)^3}{3!} + \ldots$

The magical result, then, is that rotation matrices are merely the exponentials of skew-symmetric matrices, where each skew-symmetric matrix corresponds to the axis of rotation. If we simplify things out, we get Rodrigues' formula for a rotation matrix R:

$$R = e^{\phi \hat{s}} = I + \sin \phi \hat{s} + (1 - \cos \phi) \hat{s}^2$$

where s is a unit vector along ω and $\phi = \|\omega\| t$ is the total amount of rotation. Given an axis of rotation s and amount of rotation ϕ, we can construct $\hat{s}$ and plug it in.

Dynamic Perspective / Optical Flow

So far, what we've covered is **static perspective**. In static perspective, nothing in our visual world moves. Everything is stationary; our camera is not moving. The world coordinate system (the XYZ-coordinate system) is positioned at the optical center (otherwise known as the center of projection). There's also an xy-coordinate system at our image plane. (There is no z because it is fixed at $-f$ in world coordinates.) Also, we move the camera, the world coordinate system will move.

In **dynamic perspective**, the world doesn't move. The camera moves. If we move the camera in a certain way, the image will change; in optical flow, we study how the image changes as the camera moves.

(Having the camera move with respect to the world is more flexible than having the world move with respect to the camera. In the visual world, the relative movement is what matters.)

Formally: when a point (X, Y, Z) in the world moves relative to the camera, its projection in the image (x, y) moves as well. This movement in the image plane is called **optical flow**. If the point (x, y) moves to $(x + \Delta x, y + \Delta y)$ in time Δt, then

$$u = \frac{(x + \Delta x) - x}{\Delta t} = \frac{\Delta x}{\Delta t}, \;\; v = \frac{(y + \Delta y) - y}{\Delta t} = \frac{\Delta y}{\Delta t}$$

are the two components of the optical flow at (x, y). Each is a function of the location in the image plane, such that we really have $u(x, y)$ and $v(x, y)$. (This is a vector field. Optical flow is sometimes known more precisely as an *optical flow field*!)

In optical flow, the projection of the world points cast a trajectory on the film (in a sense). We want to figure out the direction and amount of displacement for any given point on the image. This can be complex. For instance, points farther away will move at a different speed than points nearby.

Important: it's all a relative motion for the points!

Generally, pixels closer to us (smaller z) have a bigger optical flow, while pixels further from us (larger z) have a smaller optical flow.

Right now, we're trying to derive the equation relating optical flow field to scene depth $Z(x, y)$ (depth does depend on the location in the scene) and the motion of the camera t, ω (translation and rotation). The translational component of the flow field is more important, as it is what tells us what $Z(x, y)$ and t are. On the other hand, the rotational component of the flow field reveals information about ω (how the camera moves), but nothing about the scene.

If the camera moves with translational velocity $t = (t_x, t_y, t_z)$ and angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$, the movement of X can be characterized as

$$\dot{X} = -t - \omega \wedge X$$

which can be written out in coordinates as

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = - \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} - \begin{bmatrix} \omega_y Z - \omega_z Y \\ \omega_z X - \omega_x Z \\ \omega_x Y - \omega_y X \end{bmatrix}$$

This equation relates camera motion to the scene point movement. The first term t refers to the fact that when the camera moves left, the scene point appears to move to the right. And the second part is just rotating, but in the opposite direction.

Even with our world coordinate system where the world doesn't move, we can still use the projection relationship

$$x = -fX/Z$$
$$y = -fY/Z$$