# CS 280 Computer Vision
# Spring 2018 Efros, Malik, Yu Lecture 17

## 1 Lecture

### Stereo Correspondence

How do we find points in our images which correspond to the projection of the same point out in the scene? Given a point in the first view, we have to search along the epipolar line in the second view and pick the best match.

We know that corresponding points lie on epipolar lines. Generally, the epipolar lines can be in various configurations in an image, and this depends on how the cameras are aligned with respect to each other. But in the special case where the cameras' optical axes are parallel to each other, the epipolar lines will be horizontal. Note: if we know the transformation between the two cameras, we can *rectify* the images (project the *same* rays through the *same* pinholes onto parallel planes) and create this result ourselves.

Accordingly, we will henceforth assume that we are solving the problem with these horizontal epipolar lines.

How do we determine which point in the second image (along the epipolar line) is the best match? Let's look at little windows surrounding each point, and compare each of those to the window surrounding the original point. The idea that they should be the same is based on **photo-consistency**. We assume that the point in the world is Lambertian and therefore should have the same radiance emitted in the direction of the two cameras. So the brightness of the original point should be equal to the brightness of the corresponding point.

However, since comparing only single pixels can be rather noisy, we'll want to compare all of the surrounding points as well. (Similarly to the single-pixel case, the brightness values of a whole window of points should be equal to the brightness values of the corresponding window of points.)

We can reshape each window of, say, 25 pixels into a column vector (i.e. we can vectorize). The two vectors should then be similar. So if we have

$$v = \begin{bmatrix} I_1 \\ \vdots \\ I_{25} \end{bmatrix} \quad \text{and} \quad w_i = \begin{bmatrix} I'_{i,1} \\ \vdots \\ I'_{i,25} \end{bmatrix}$$

denoting the windows of the original point $p$ and its [proposed] correspondence $q_i$, respectively, then we want to minimize

$$\|v - w_i\|_2$$

(i.e. the length of $v - w$). Specifically, we want to choose the $i$ that minimizes this distance. Note: the L2-norm equates to the sum of squared differences (SSD), technically after a squaring.

Alternatively, we can maximize the overlap (the cosine of the angle between the vectorized windows):

$$\arg\max_i \frac{v \cdot w_i}{\|v\|\|w\|}$$

This is sometimes called the cosine distance – it's just the normalization of the dot product. Technically it's the cosine of the angle between the two vectors, so maximum similarity (where the angle is 0) gives the highest cosine value 1. Historically, this objective equates to normalized cross-correlation (NCC).

NCC works slightly better, because it doesn't depend on things like gain control (which could affect brightness in different shots).

*Once we have the the corresponding points in the rectified images, we can compute the disparity $x - x'$ and set the depth to $Bf/(x - x')$ where $B$ is the baseline.*

### Limitations of Our Naive Algorithm

- *Textureless surfaces*: imagine trying to compare viewpoints of a white wall or any other surface with uniform brightness. The windows will look the same at many shifts!
- *Repetition*: imagine a grille. Many different views of the bars will look pretty similar.
- *Occlusion*: it's possible that an object one camera sees is something that the other camera doesn't see. Note: when an object can be seen by one eye and not the other, it's called **half occlusion**. When an object can be seen by neither eye, it's called **full occlusion**.
- *Non-Lambertian surfaces*: maybe we're looking at a mirror or a shiny surface, and the brightness is *not* the same in all directions.

It's also heavily dependent on the window size. Larger windows are generally better for smoothing away the effect of noise. However, if they're too big and cover regions of multiple depths at once, we're going to get disparities that are kind of an average – the depth will be smoothed as well! Basically, if the window is large we'll get less detail and less noise, and if the window is small we'll get more detail and more noise.

More modern techniques involve things like graph cuts.

## Optical Flow

Optical flow is related to stereo disparity. In stereo disparity, there are two cameras at slightly different positions with a disparity between corresponding points. To tie into optical flow, we can think of this as taking one camera and moving it. (Instead of having two cameras looking simultaneously at the scene, we'll have one camera moving around the scene.) And whenever we have one moving camera, the corresponding concept is optical flow. How do the points in the image appear to move – by how many pixels?

Before, we didn't discuss how to actually measure optical flow. But now it's time. Let's examine algorithms for computing optical flow given some video sequence.

**Aperture problem:** if we see just a little bit of the display, we might think the optical flow is something other than what it really is. On the other hand, in some conditions we actually will see the true optical flow. We really want the full image (i.e. global information) in order to compute the correct optical flow, although sometimes local information at the right places (like corners) will do the job as well.

But there are also fake corners (T-junctions) created by one surface occluding another surface. These will often also give false signals.

### The Math

We have a point $P = (x_1, y_1, t_1)$ which moves to $P' = (x_2, y_2, t_2)$. The $t$s are times. Optical flow has two components $u$ and $v$, representing to the $x$-component and $y$-component respectively. They are defined as

$$u = \frac{x_2 - x_1}{t_2 - t_1} = \frac{dx}{dt} \quad \text{and} \quad v = \frac{y_2 - y_1}{t_2 - t_1} = \frac{dy}{dt}$$

The units of optical flow are pixels per second.

So optical flow is based on correspondence over time. Note that we are measuring these locations in the image plane (not out in the 3D world).

Again we will assume that the surface in the world is Lambertian (so the brightness of the point will remain the same – the **brightness constancy assumption**). Therefore it should be the case that $I(x, y, t) \approx I(x + \Delta x, y + \Delta y, t + \Delta t)$. By extension, the total derivative should be 0.

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \Delta x I_x(x, y, t) + \Delta y I_y(x, y, t) + \Delta t I_t(x, y, t) + \text{higher-order terms}$$

$I_x$ is the partial derivative of $I$ with respect to $x$. This is just the Taylor series expansion of $I$. Now, because of the brightness constancy assumption, $I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t)$ and

$$\begin{aligned} I(x, y, t) &= I(x, y, t) + \Delta x I_x(x, y, t) + \Delta y I_y(x, y, t) + \Delta t I_t(x, y, t) \\ 0 &= \Delta x I_x(x, y, t) + \Delta y I_y(x, y, t) + \Delta t I_t(x, y, t) \\ 0 &= \frac{\Delta x}{\Delta t} I_x(x, y, t) + \frac{\Delta y}{\Delta t} I_y(x, y, t) + I_t(x, y, t) \end{aligned}$$

Now, since $\Delta x / \Delta t$ is simply the $x$-component of the optical flow,

$$I_x u + I_y v + I_t = 0$$

and *this* is the grand equation for optical flow: the **optical flow constraint equation**.

We can calculate $I_x$ and $I_y$ by taking the difference between neighboring pixels' $x$- or $y$-components. We can calculate $I_t$ by taking the brightness difference between the same pixel in two neighboring frames. So if we have a video clip, then $I_x$, $I_y$, and $I_t$ are known.

And where $I_x$, $I_y$, and $I_t$ are our knowns, $u$ and $v$ are our unknowns.

A potential problem: so far, we have only one equation but two unknowns! But there are tricks to deal with this; they will be revealed in time. First let's rewrite the optical flow constraint equation as

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -I_t$$

Then, since $\begin{bmatrix} I_x & I_y \end{bmatrix} = \nabla I$, this equation becomes

$$\nabla I \cdot \mathbf{u} = -I_t$$

where $\mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^T$. And herein lies the problem (and this is where the aperture problem arises): we're trying to determine $\mathbf{u}$, but $\mathbf{u}$ has one component along the gradient of $I$ and one component orthogonal to the gradient of $I$. (Recall that $\nabla I$ and $\mathbf{u}$ are both vectors in two-dimensional space!) And the only movement we can measure is perpendicular to the edge. Any sliding along the edge is unmeasurable. (The human visual system asserts the nonexistence of anything it cannot measure, and hence won't measure any sliding motion at all.)

*We can measure the component of* $\mathbf{u}$ *along the gradient of* $I$*. However, we cannot measure the component perpendicular to the gradient of* $I$*. It is unknown to us. Since the gradient is oriented perpendicular to an edge, this means that we can only register movement perpendicular to said edge.*

We need to extend our signal. Let's examine a neighborhood of pixels, and assume they all have the same optical flow. (We will believe in the **local constancy of optical flow**: the idea that locally, the optical flow is constant.) Now we will have an equation

$$\begin{bmatrix} I_x^i & I_y^i \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -I_t^i$$

for each pixel $i$. (These are not powers, merely superscripts.) If we have $n$ pixels, we will thus have $n$ equations. This can be rewritten as

$$\begin{bmatrix} I_x^1 & I_y^1 \\ I_x^2 & I_y^2 \\ \vdots & \vdots \\ I_x^n & I_y^n \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t^1 \\ I_t^2 \\ \vdots \\ I_t^n \end{bmatrix}$$

or $A\mathbf{u} = -\mathbf{b}$ where $A$ is $n \times 2$, $\mathbf{u}$ is $2 \times 1$, and $\mathbf{b}$ is $n \times 1$. This is an overdetermined system of equations which we can solve using least squares. The solution is $\mathbf{u} = -(A^T A)^{-1} A^T b$.

However, this depends on $A^T A$ being invertible, i.e. of rank 2. If its rank is less than 2, we cannot compute the optical flow exactly (if at all).

$$\begin{aligned} A^T A &= \begin{bmatrix} I_x^1 & \dots & I_x^n \\ I_y^1 & \dots & I_y^n \end{bmatrix} \begin{bmatrix} I_x^1 & I_y^1 \\ \vdots & \vdots \\ I_x^n & I_y^n \end{bmatrix} \\ &= \begin{bmatrix} \sum \left(I_x^i\right)^2 & \sum I_x^i I_y^i \\ \sum I_x^i I_y^i & \sum \left(I_y^i\right)^2 \end{bmatrix} \end{aligned}$$

i.e. a $2 \times 2$ matrix composed from the partial derivatives of $I$. This matrix is called the **second moment matrix**. It all comes down to the rank of this matrix. If the rank is 0 (I is constant over the image), then we can't solve the equations and can't say anything about the optical flow. If the rank is 1 (all of the $\nabla I$ vectors are scalar multiples of the same thing, e.g. in the neighborhood of an edge), we *still* can't solve the equations. But if the rank is 2 (e.g. we are at a corner), we can at last compute the optical flow.