A Summary of CSE 252A: Computer Vision I

Owen Jow

December 2, 2019

1 Biological Vision

Light enters the human eye through the cornea and passes through the pupil, lens, and vitreous humor before striking the retina. There it is registered by rods and cones and shallowly processed by ganglion and bipolar cells. Following this, the aggregate information exits the eye through the optic nerve and travels to the visual cortex.

Humans see with pretty high resolution relative to many other organisms, but on a different note are virtually colorblind next to the stomatopods. Stomatopods are shrimp-like animals which have 16 types of photoreceptors as compared to humans' three, allowing them to see much more of the EMR spectrum than us. They can even see UV radiation, far-red light, and polarized light. They also have eyes on stalks which are divided into three components – thus letting them see from multiple viewpoints and perform stereo with just one eye. In 252A, I learned that stomatopods are amazing.

2 Geometric Image Formation

Per similar triangles, basic pinhole perspective projection is

$$x' = \frac{fx}{z}$$
$$y' = \frac{fy}{z}$$

Perspective projection can take 3D points at infinity to *vanishing points*, finite 2D points. We use homogeneous coordinates to represent the points at infinity as part of a projective geometry space, which is just regular Euclidean points plus these points at infinity.[1] If a line is expressed in 3D as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \lambda \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix}$$

then its vanishing point is determined by computing the perspective projection of the line and taking the limit as λ goes to infinity:

$$\begin{aligned} \lim_{\lambda \to \infty} \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} f(A_x + \lambda D_x)/(A_z + \lambda D_z) \\ f(A_y + \lambda D_y)/(A_z + \lambda D_z) \end{bmatrix} \\ &= \begin{bmatrix} fD_x/D_z \\ fD_y/D_z \end{bmatrix} \end{aligned}$$

[1] There is a point at infinity for every line direction. Note that $(x, y, 0)$ is equivalent to $(\lambda x, \lambda y, 0)$.

Since the vanishing point only depends on the direction vector, all parallel lines in 3D will converge at the same vanishing point in 2D under perspective projection.

Note that perspective projection is nonlinear in Euclidean coordinates due to the division by z. We can linearize it via Taylor series expansion around an arbitrary 3D point (x_0, y_0, z_0) to create the simpler *affine camera model*:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{fx_0}{z_0} + \frac{f}{z_0}(x - x_0) - \frac{fx_0}{z_0^2}(z - z_0) \\ \frac{fy_0}{z_0} + \frac{f}{z_0}(y - y_0) - \frac{fy_0}{z_0^2}(z - z_0) \end{bmatrix}$$

As a further simplification, we can constrain the expansion point to be $(0, 0, z_0)$ (on the optical axis):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} fx/z_0 \\ fy/z_0 \end{bmatrix}$$

making projection no longer dependent on depth. This is the *scaled orthographic projection* model. Note that for validity, both the affine and the scaled orthographic models will require 3D input points to be in the neighborhood of the point of expansion.

2.1 Lenses

To avoid super-dark photographs, we want our cameras' apertures to be larger than pinhole-sized. However, navely widening an aperture will make the image blurry, as each pixel will have a bunch of points mapping to it. To avoid this, we use a lens to focus the rays through the aperture according to some design. The ideal *thin lens* maps all of the rays for a point to the intersection of (a) *its ray through the center of projection* and (b) *its ray parallel to the optical axis that goes through the lens.*

Mathematically, we can use the thin lens equation to calculate the depth z_{im} at which it'll focus all of the rays for a point at distance z_{obj} in front of the pinhole:

$$\frac{1}{z_{im}} + \frac{1}{z_{obj}} = \frac{1}{f}$$

where f is the focal length of the lens.

3 Radiometry, Reflectance

Radiometry has to do with the physical measurement of electromagnetic radiation (but since this is a vision class, we mostly just care about the part of the spectrum that is visible to us; we will refer to this part of the spectrum as *light*). We can measure (a) light flowing through a point in space in a certain direction – *radiance* $L(\mathbf{x}, \theta, \phi)$ – or (b) light arriving at a point on a surface – *irradiance* $E(\mathbf{x})$ – or (c) many other light-related quantities which aren't really a focus of this class.

Light is typically measured as energy (J) or power (W) distributed over something, e.g. *area* in the case of irradiance and *projected area perpendicular to a direction* in the case of radiance.

Radiance is defined

$$L = \frac{\text{power}}{(dA\,\cos\theta)\,d\omega}$$

An area dA_1 emitting radiance L will transfer the following amount of radiance to an area dA_2 at distance r, oriented s.t. θ_1 and θ_2 are the angles between the joining ray and the areas' normals:

$$\begin{aligned} & LdA_1 \cos\theta_1\, d\omega_{1\to 2} \\ = {} & LdA_1 \cos\theta_1 (dA_2 \cos\theta_2 / r^2) \end{aligned}$$

A BRDF ρ describes the fraction of light from an incoming direction which is reflected in an outgoing direction. It is defined as the ratio of outgoing radiance to incoming irradiance:

$$\rho(\mathbf{x}; \theta_i, \phi_i, \theta_o, \phi_o) = \frac{L_o(\mathbf{x}, \theta_o, \phi_o)}{L_i(\mathbf{x}, \theta_i, \phi_i) \cos\theta_i \, d\omega}$$

and is used in the *reflection equation*, which describes the amount of light L_r reflected in a certain direction $\omega_r = (\theta_r, \phi_r)$ at a certain point $\mathbf{x}$, as

$$L_r(\mathbf{x}, \omega_r) = \int_{H^2} \rho(\mathbf{x}; \omega_i \to \omega_r) L_i(\mathbf{x}, \omega_i) \cos\theta_i \, d\omega_i$$

In computer vision, we often make the simplification that the BRDF is constant (which is the case for purely Lambertian surfaces), meaning light is reflected equally in all directions.

We also often assume that lights are distant relative to the scene. Then, under a local illumination model, outgoing radiance at a surface point $\mathbf{x}$ is computed as

$$L(\mathbf{x}) = \rho_d(\mathbf{x})\mathbf{n}(\mathbf{x}) \cdot q\mathbf{s}$$

where ρ_d is the constant BRDF (the *albedo*), $\mathbf{n}$ is the normal at the point, q is the light intensity, and $\mathbf{s}$ is the direction to the light source (approximately the same for all points). This is also the image irradiance at the pixel corresponding to $\mathbf{x}$, since our reflectance doesn't depend on direction.

3.1 Photometric Stereo

In *photometric stereo*, we take multiple photos of a static scene under different, known lighting, then estimate the albedo and normals at each pixel and integrate them to get depth i.e. 3D.

If we have at least three images, we have at least three equations

$$\begin{aligned} L_1(\mathbf{x}) &= \rho_d(\mathbf{x})\mathbf{n}(\mathbf{x}) \cdot q_1\mathbf{s}_1 \\ L_2(\mathbf{x}) &= \rho_d(\mathbf{x})\mathbf{n}(\mathbf{x}) \cdot q_2\mathbf{s}_2 \\ L_3(\mathbf{x}) &= \rho_d(\mathbf{x})\mathbf{n}(\mathbf{x}) \cdot q_3\mathbf{s}_3 \end{aligned}$$

and can solve for the three unknowns in $\rho_d(\mathbf{x})\mathbf{n}(\mathbf{x})$. Then $\rho_d(\mathbf{x})$ is the norm of $\rho_d(\mathbf{x})\mathbf{n}(\mathbf{x})$.

$\mathbf{n}$ encodes slant and tilt, which are the partial derivatives of depth w.r.t. x and y respectively. Thus, once we have the normals we can integrate them to get the depth (perhaps using Horn integration).

4 Edge Detection

An edge is a brightness discontinuity over the spatial domain.

Since edges often signify object boundaries and/or highlight meaningful reflectance or lighting properties, it can be useful to detect them. In a sense, the edges can provide a simpler (though perhaps lossy) interpretation of what's happening in the original image.

A classically popular means of performing 2D edge detection is the Canny algorithm. In *Canny edge detection*, we curb noise in the image using a Gaussian filter, then compute the gradient direction and magnitude[2] at each point. Following this, we perform non-maximum suppression on the breadth of

[2]The gradient magnitude represents the "strength" of an edge at a point.

the edges (based on gradient magnitude) to thin them out, and trace the final edges using hysteresis thresholding. Hysteresis thresholding starts edges at locations with gradient magnitudes above τ_{high}, then traces them in a direction orthogonal to the gradient until the magnitude falls below τ_{low}.

In theory, the resulting edges are thin, strong, and properly connected.

- Note that if we only had one threshold, we would miss out on quality edges if it were too high, and retain weak or extraneous edges if it were too low.

5 Corner Detection

A corner is a location around which there are two (or more) strong gradient directions.

Owing to its distinctiveness, it is a promising choice for a feature which'll be *detected independently and matched later*. Unlike edge or low-texture regions, it won't look the same in nearby places.

To identify corners, we typically consider the second moment matrix at each point:

$$\sum_{x,y \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

for I_x and I_y the image derivatives with respect to x and y, and W a small surrounding window. Via its eigenvalues $(\lambda_{\max}, \lambda_{\min})$ and eigenvectors $(\mathbf{e}_{\max}, \mathbf{e}_{\min})$, the second moment matrix summarizes the gradient distribution in W. Namely, $\mathbf{e}_{\max}$ is the dominant gradient direction in the window, while $\lambda_{\max}$ describes the strength of the gradient in that direction. $\mathbf{e}_{\min}$ is the gradient direction orthogonal to $\mathbf{e}_{\max}$ (and $\lambda_{\min}$ the corresponding gradient "strength").

Since a corner has strong gradients in multiple directions, we want to see high and similar values of $\lambda_{\max}$ and $\lambda_{\min}$. (If one eigenvalue is much larger than the other, we probably have an edge.) There are many corner response functions based on $\lambda_{\max}$ and $\lambda_{\min}$; one, Shi-Tomasi, is simply $\lambda_{\min}$.

- The full *Shi-Tomasi corner detection* method first filters the image with a Gaussian to remove noise, then computes $\lambda_{\min}$ of the second moment matrix at each point. Finally, it takes corners as points which are local maxima in terms of $\lambda_{\min}$, under the condition that $\lambda_{\min} > \tau$.

6 Stereo Vision

The main result of stereo is depth estimation from multiple views of the same scene. This typically consists of (a) setting up epipolar geometry and using it to efficiently find correspondences between images, and (b) using these correspondences to estimate the original 3D points via triangulation.

Epipolar geometry revolves around relating points in the stereo views according to the *essential* or *fundamental matrix*. The essential matrix is for calibrated cameras, for which image coordinates are equivalent to camera coordinates. The fundamental matrix is for uncalibrated cameras, for which image coordinates are equivalent to perspective-projected pixel coordinates.

$$\mathbf{p}^T \mathbf{E} \mathbf{p}' = 0$$
$$\mathbf{p}^T \mathbf{F} \mathbf{p}' = 0$$

If we have a point in one image, the equation for the epipolar line in the other image is given by substituting the known point into the appropriate version of the *epipolar constraint* above.

This is helpful because a point's match in the other image lies on its associated epipolar line, so in finding correspondences we reduce the search space to the epipolar line (and sometimes even just a neighborhood on the epipolar line around the original location, if we have reason to believe the disparity is not too large). In most cases, we can also perform rectification [map the epipoles to horizontal infinity $(1, 0, 0)$] s.t. epipolar lines are identical scan lines across the two images.

To find matches, we just compare *windows around points on the epipolar line* to the *window around the known point* according to some metric. Of course there are also some extensions we can implement, such as adaptive window sizing or 1-to-"at most one" matching.

6.1 Estimating the Essential/Fundamental Matrix Using RANSAC

If we have a lot of correspondences, we can use RANSAC (*random sample consensus*) on top of the eight-point algorithm to estimate the essential or fundamental matrix. RANSAC is an algorithm for fitting a model in the possible presence of model-ruining outliers, in this case poor correspondences. In the case of fundamental matrix estimation, it would work by repeating the following:

1. Randomly selecting eight points, and feeding them to the eight-point algorithm to estimate $\mathbf{F}$.
2. Identifying *inlier* correspondences that agree with the model based on epipolar line distance.

After some number of iterations, we'd keep the $\mathbf{F}$ which produced the largest set of inliers.

6.2 Structure from Motion

In *structure from motion* (SfM), we again take multiple images, but estimate the camera extrinsics (and maybe intrinsics) in addition to 3D point positions. It works by (a) detecting feature points, (b) computing the essential/fundamental matrix using RANSAC, (c) *factorizing it to get camera parameters*, and (d) performing standard stereo matching & triangulation to get 3D feature positions.

7 Optical Flow

In the optical flow problem, we estimate a flow field which depicts perceived (not actual) motion in an image. In doing so, we typically make three assumptions:

1. *Color/brightness constancy*, or that objects' pixel intensities remain constant between frames.
2. *Small motion*, or that objects don't move very far between frames.
3. *Spatial coherency*, or that points in little windows all move the same way.

The *Lucas-Kanade* optical flow method starts from the brightness constancy equation

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t)$$

and based on the small motion assumption, linearizes it via Taylor expansion:

$$I(x, y, t) + \Delta x \frac{\partial I}{\partial x} + \Delta y \frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} = I(x, y, t)$$
$$\Delta x \frac{\partial I}{\partial x} + \Delta y \frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} = 0$$

which is equivalent to the following equation if Δt is infinitesimal:

$$\frac{dx}{dt}\frac{\partial I}{\partial x} + \frac{dy}{dt}\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0$$
$$uI_x + vI_y + I_t = 0$$

Note that we want to estimate u and v at each point. I_x, I_y, and I_t are initially computable.

However, given a single point, we have two unknowns and only one equation. To get around this, Lucas-Kanade brings in the spatial coherence assumption and sets up equations for everyplace in a local window around the point of interest. Then we can solve for u and v using least-squares.

$$\left(\sum_{x,y\in W}\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix} = \sum_{x,y\in W}\begin{bmatrix} -I_xI_t \\ -I_yI_t \end{bmatrix}$$

This will not work if the gradients are constant or our assumptions are violated. To help achieve small motion, we can also use an iterative (multiscale) Lucas-Kanade method. This involves running Lucas-Kanade on images from the coarsest to finest levels of a pyramid: iteratively (a) estimating a flow field using Lucas-Kanade, (b) adding it to the current flow field if there is one, and (c) if possible, upsampling it and using it to warp the next-highest-resolution image in the pyramid.

8 Recognition

A Bayesian (MAP)(optimal) classifier outputs the class with the highest posterior probability $P(\omega_i \mid \mathbf{x})$ (probability of the class given the observation). It computes the posterior probability using Bayes' rule, and so requires an estimate of the likelihood $P(\mathbf{x} \mid \omega_i)$ and the priors $P(\omega_i)$.

This is hard to approximate if/when the dimensionality of the feature space is high, so it usually helps to perform dimensionality reduction in order to condense things into "important" dimensions. We can do this using *PCA*, which projects onto a lower-dimensional space which best creates high-spread dimensions, or we can do it using *Fisher's linear discriminant*, which projects onto a lower-dimensional space that separates out the data belonging to different classes.

For classification, deep networks also usually predict posterior probabilities, but they do it based on magic (and input/output examples) instead of Bayes' rule.

9 Color

The color we perceive is based on both physical scene properties (lighting and reflectance) *and* our color response functions.[3] We usually have three types of color receptors and therefore three types of color response functions (for S/M/L $\leftrightarrow$ B/G/R wavelengths).[4]

Maybe the most well-known linear color space is the sRGB color space, and this is probably how colors are most often stored (as $[0, 1]$ sRGB components). But we can also transform an sRGB representation into other color spaces such as HSV, perhaps up to gamut limitations because different color spaces allow for the portrayal of different chromaticities. (The CIE-XYZ gamut has all human-perceivable chromaticities, though it is based on a physically impossible basis of primaries.)

[3] ...though we often experience *color constancy* (see a thing as being the same color even under different lighting).

[4] Accordingly, we usually design our cameras to filter light into R, G, and B components.

One useful color space is the SUV color space, which is nice because it separates color into one specular component and two diffuse components. Then we can perform, e.g., Lambertian photometric stereo on just the diffuse components. SUV is defined as a rotation of the RGB color space.[5]

Acknowledgments

This document was inspired by Yining Liu's "what I learned" writeups.

[5]Specifically, it turns the specular (lighting) color into the first axis of the coordinate frame.