6.801/6.866: Machine Vision Fall 2020

Lecture Notes

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Contents

6.801/6.866: Machine Vision, Lecture Notes

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
MIT Department of Electrical Engineering and Computer Science
Fall 2020

1 Lecture 2: Image Formatio, Perspective Projection, Time Derivative, Motion Field

1.1 Motion from Perspective Projection, and FOE

Definition of perspective projection:

$$\frac{x}{f} = \frac{X}{Z}, \frac{y}{f} = \frac{Y}{Z} \textbf{ (component form)}$$

$$\frac{1}{f}\mathbf{r} = \frac{1}{\mathbf{R} \cdot \hat{\mathbf{z}}}\mathbf{R} \textbf{ (vector form)}$$

If we differentiate these perspective projection equations:

$$\frac{1}{f}\frac{dx}{dt} = \frac{1}{Z}\frac{dX}{dt} - \frac{X}{Z^2}\frac{dZ}{dt}$$

What are these derivatives? They correspond to **velocities**. Let's define some of these velocities:

- $u \triangleq \frac{dx}{dt}$
- $v \triangleq \frac{dy}{dt}$
- $U \triangleq \frac{dX}{dt}$
- $V \triangleq \frac{dY}{dt}$
- $W \triangleq \frac{dZ}{dt}$

Now, rewriting the differentiated perspective projection equations with these velocity terms, we first write the equation for the x component:

$$\frac{1}{f}u = \frac{1}{Z}U - \frac{X}{Z^2}W$$

Similarly, for y:

$$\frac{1}{f}v = \frac{1}{Z}V - \frac{Y}{Z^2}W$$

Why are these equations relevant? They allow us to find parts of the image that don't exhibit any motion - i.e. stationary points. Let's find where $U = V = 0$. Let the point (x_0, y_0) correspond to this point. Then:

$$\frac{x_0}{f} = \frac{U}{W}, \frac{y_0}{f} = \frac{V}{W}$$

Focus of Expansion (FOE): Point in image space given by (x_0, y_0). This point is where the 3D motion vector intersects with the line given by $z = f$.

Why is FOE useful? If you know FOE, you can derive the **direction of motion** by drawing a vector from the origin to FOE.

Additionally, we can rewrite the differentiated perspective projection equations with FOE:

$$\frac{1}{f}u = \frac{x_0 - x}{f}\frac{W}{Z}\ \textbf{(x comp.)},\quad \frac{1}{f}v = \frac{y_0 - y}{f}\frac{W}{Z}\ \textbf{(y comp.)}$$

Cancelling out the focal length (f) terms:

$$u = (x_0 - x)\frac{W}{Z}\ \textbf{(x comp.)},\quad v = (y_0 - y)\frac{W}{Z}\ \textbf{(y comp.)}$$

A few points here:

- You can draw the vector diagram of the motion field in the image plane.
- All vectors in the motion field expand outward from **FOE**.
- Recall that perspective projection cannot give us absolute distances.

For building intuition, let's additionally consider what each of these quantities mean. The inverse term $\frac{Z}{W} = \frac{Z}{\frac{dZ}{dt}}$ has units of $\frac{\textbf{meters}}{\frac{\textbf{meters}}{\textbf{second}}}$ = **seconds** - i.e. **Time of Impact**.

Let's now revisit these equations in vector form, rather than in the component form derived above:

$$\frac{1}{f}\frac{d\mathbf{r}}{dt} = \frac{1}{\mathbf{R}\cdot\hat{\mathbf{z}}} - \frac{R}{(\mathbf{R}\cdot\hat{\mathbf{r}})^2}\frac{d}{dt}(\mathbf{R}\cdot\hat{\mathbf{r}})$$

Let's rewrite this with dots for derivatives. Fun fact: The above notation is Leibniz notation, and the following is Newtonian notation:

$$\frac{1}{f}\dot{\mathbf{r}} = \frac{1}{\mathbf{R}\cdot\hat{\mathbf{z}}}\dot{\mathbf{R}} - \frac{R}{(\mathbf{R}\cdot\hat{\mathbf{z}})^2}(\dot{\mathbf{R}}\cdot\hat{\mathbf{z}})$$

$$\frac{1}{f}\dot{\mathbf{r}} = \frac{1}{Z}(\dot{\mathbf{R}} - W\frac{1}{f}\mathbf{r})$$

One way for reasoning about these equations is that **motion is magnified by the ratio of the distance terms**.

Next, we'll reintroduce the idea of **Focus of Expansion**, but this time, for the vector form. FOE in the vector form is given at the point where $\dot{\mathbf{r}} = 0$:

$$\frac{1}{f}\dot{\mathbf{r}} = \frac{1}{W}\dot{\mathbf{R}}$$

We can use a dot product/cross product identity to rewrite the above expression in terms of cross products. The identity is as follows for any $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^n$

$$\mathbf{a}\times(\mathbf{b}\times\mathbf{c}) = (\mathbf{c}\cdot\mathbf{a})\mathbf{b} - (\mathbf{a}\cdot\mathbf{b})\mathbf{c}$$

Using this identity, we rewrite the expression above to solve for FOE:

$$\frac{1}{f}\dot{\mathbf{r}} = \frac{1}{(\mathbf{R}\cdot\hat{\mathbf{z}})^2}(\hat{\mathbf{z}}\times(\dot{\mathbf{R}}\times\mathbf{R}))$$

What is this expression? This is **image motion** expressed in terms of **world motion**. Note the following identities/properties of this motion, which are helpful for building intuition:

- $\dot{\mathbf{r}}\cdot\hat{\mathbf{z}} = 0 \implies$ Image motion is perpendicular to the z-axis. This makes since intuitively because otherwise the image would be coming out of/going into the image plane.
- $\dot{\mathbf{r}} \perp \hat{\mathbf{z}}$
- $\dot{\mathbf{R}} \parallel \mathbf{R} \implies \dot{\mathbf{r}} = 0$ (this condition results in there being no image motion).

1.2 Brightness and Motion

Let's now consider how brightness and motion are intertwined. Note that for this section, we will frequently be switching between continuous and discrete. The following substitutions/conversions are made:

- **Representations of brightness functions**: $E(x, y) \leftrightarrow E[x, y]$
- **Integrals and Sums**: $\int_x \int_y \leftrightarrow \sum_x \sum_y$
- **Brightness Gradients and Finite Differences**: $(\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}) \leftrightarrow (\frac{1}{\delta x}(E[k, e+1] - E[k, e])$

1.2.1 1D Case

$$\frac{dx}{dt} = U \implies \delta x = U\delta$$

By taking a linear approximation of the local brightness:

$$\delta E = E_x \delta x = uE_x \delta t \quad \text{(note here that } E_x = \frac{\partial E}{\partial x}\text{)}$$

Dividing each side by δt, we have:

$$uE_x + E_t = 0 \implies U = -\frac{E_x}{E_t} = -\frac{\frac{\partial E}{\partial t}}{\frac{\partial E}{\partial x}}$$

A couple of points about this:

- This 1D result allows us to recover motion from brightness.
- We can infer motion from a single point. However, this is only true in the 1D case.
- We can estimate from 1 pixel, but frequently, we have much more than 1 pixel, so why use just 1? We can reduce noise by estimating motion from many pixels through regression techniques such as Ordinary Least Squares (OLS).
- From statistics, the standard deviation of the motion estimates will be reduced by a factor of $\frac{1}{\sqrt{N}}$, where N is the number of pixels sampled for estimating motion.

Finite Difference approximation for E is given by:

$$E \approx \frac{1}{\delta x}(E(x + \delta x, t) - E(x, t))$$

Motion estimation can be done through unweighted averaging:

$$\bar{u}_{\text{unweighted}} = \frac{1}{N}\sum_{i=1}^{N} \frac{-E_{t_i}}{E_{x_i}}$$

As well as weighted averaging:

$$\bar{u}_{\text{weighted}} = \frac{\sum_{i=1}^{N} w_i \frac{-E_{t_i}}{E_{x_i}}}{\sum_{i=1}^{N} w_i}$$

A quick check here: take $w_i = 1 \; \forall \; i \in \{1, ..., N\}$. Then we have that $\bar{u}_{\text{weighted}} = \frac{1}{N}\sum_{i=1}^{N} \frac{-E_t}{E_x} = \bar{u}_{\text{unweighted}}$.

Note that in the continuous domain, the sums in the weighted and unweighted average values are simply replaced with integrals.

1.2.2 2D Case

While these results are great, we must remember that images are in 2D, and not 1D. Let's look at the 2D case. First and foremost, let's look at the brightness function, since it now depends on x, y, and t: $E(x, y, t)$. The relevant partial derivatives here are thus:

- $\frac{\partial E}{\partial x}$ - i.e. how the brightness changes in the x direction.
- $\frac{\partial E}{\partial y}$ - i.e. how the brightness changes in the y direction.
- $\frac{\partial E}{\partial t}$ - i.e. how the brightness changes w.r.t. time.

As in the previous 1D case, we can approximate these derivatives with finite forward first differences:

- $\frac{\partial E}{\partial x} = E_x \approx \frac{1}{\delta x}(E(x + \delta x, y, t) - E(x, y, t))$
- $\frac{\partial E}{\partial y} = E_y \approx \frac{1}{\delta y}(E(x, y + \delta y, t) - E(x, y, t))$
- $\frac{\partial E}{\partial t} = E_t \approx \frac{1}{\delta t}(E(x, y, t + \delta t) - E(x, y, t))$

Furthermore, let's suppose that x and y are parameterized by time, i.e. $x = x(t), y = y(t)$. Then we can compute the First-Order Condition (FOC) given by:

$$\frac{dE(x, y, t)}{dt} = 0$$

Here, we can invoke the chain rule, and we obtain the result given by:

$$\frac{dE(x, y, t)}{dt} = \frac{dx}{dt}\frac{\partial E}{\partial x} + \frac{dy}{dt}\frac{\partial E}{\partial y} + \frac{\partial E}{\partial t} = 0$$

Rewriting this in terms of u, v from above:

$$uE_x + vE_y + E_t = 0$$

Objective here: We have a time-varying sequence of images, and our goal is to find and recover motion.

To build intuition, it is also common to plot in velocity space given by (u, v). For instance, a linear equation in the 2D world corresponds to a line in velocity space. Rewriting the equation above as a dot product:

$$uE_x + vE_y + E_t = 0 \leftrightarrow (u, v) \cdot (E_x, E_y) = -E_t$$

Normalizing the equation on the right by the magnitude of the brightness derivative vectors, we obtain the **brightness gradient**:

$$(u, v) \cdot \left(\frac{E_x}{\sqrt{E_x^2 + E_y^2}}, \frac{E_y}{\sqrt{E_x^2 + E_y^2}}\right) = -\frac{E_t}{\sqrt{E_x^2 + E_y^2}}$$

What is the **brightness gradient**?

- A unit vector given by: $\left(\frac{E_x}{\sqrt{E_x^2+E_y^2}}, \frac{E_y}{\sqrt{E_x^2+E_y^2}}\right) \in \mathbb{R}^2$.
- Measures spatial changes in brightness in the image in the image plane x and y directions.

Isophotes: A curve on an illuminated surface that connects points of equal brightness (source: Wikipedia).

As we saw in the previous case with 1D, we don't want to just estimate with just one pixel. For multiple pixels, we will solve a system of N equations and two unknowns:

$$\begin{aligned} uE_{x_1} + vE_{y_1} + E_{t_1} &= 0 \\ uE_{x_2} + vE_{y_2} + E_{t_2} &= 0 \end{aligned}$$

Rewriting this in matrix form:

$$\begin{bmatrix} E_{x_1} & E_{y_1} \\ E_{x_2} & E_{y_2} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} -E_{t_1} \\ -E_{t_2} \end{bmatrix}$$

Solving this as a standard $Ax = b$ problem, we have:

$$\begin{bmatrix} U \\ V \end{bmatrix} = \frac{1}{(E_{x_1}E_{y_2} - E_{y_1}E_{x_2})} \begin{bmatrix} E_{y_2} & -E_{y_1} \\ -E_{x_2} & E_{x_1} \end{bmatrix} \begin{bmatrix} -E_{t_1} \\ -E_{t_2} \end{bmatrix}$$

Note that the expression given by $\frac{1}{(E_{x_1}E_{y_2} - E_{y_1}E_{x_2})}$ is the determinant of the partial derivatives matrix, since we are taking its inverse (in this case, simply a 2x2 matrix).

When can/does this fail? It's important to be cognizant of edge cases in which this motion estimation procedure/algorithm fails. Some cases to consider:

- When brightness partial derivatives / brightness gradients are parallel to one another ↔ The determinant goes to zero ↔ This corresponds to linear dependence in the observations. This occurs when $E_{x_1}E_{y_2} = E_{y_1}E_{x_2} \implies \frac{E_{y_1}}{E_{x_1}} = \frac{E_{y_2}}{E_{x_2}}$.

This issue can be mitigated by weighting the pixels as we saw in the 1D case above. However, a more robust solution is to search for a minima of motion, rather than the point where it has zero motion. The intuition here is that even if we aren't able to find a point of zero motion, we can still get as close to zero as possible. Mathematically, let us define the following objective:

$$J(u, v) \triangleq \int_{x \in \mathbb{X}} \int_{y \in \mathbb{Y}} (uE_x + vE_y + E_t)^2 dxdy$$

Then we now seek to solve the problem of:

$$u^*, v^* = \arg\min_{u,v} J(u, v) = \arg\min_{u,v} \int_{x \in \mathbb{X}} \int_{y \in \mathbb{Y}} (uE_x + vE_y + E_t)^2 dxdy$$

Since this is an unconstrained optimization problem, we can solve by finding the minimum of the two variables using two First-Order Conditions (FOCs):

- $\frac{\partial J(u,v)}{\partial u} = 0$
- $\frac{\partial J(u,v)}{\partial v} = 0$

Here, we have two equations and two unknowns. When can this fail?

- When we have linear independence. This occurs when:
  - E = 0 everywhere
  - E = constant
  - $E_x = 0$
  - $E_y = 0$
  - $E_x = E_y$
  - $E_x = kE_y$
- When $E = 0$ everywhere (professor's intuition: "You're in a mine.")
- When $E_x, E_y = 0$ (constant brightness).
- Mathematically, this fails when: $\int_x \int_x E_x^2 \int_y \int_y E_y^2 - (\int_x \int_y E_x E_y)^2 = 0$

When is this approach possible? **Only when isophotes are not parallel straight lines - i.e. want isophote curvature/rapid turning of brightness gradients.**

Noise Gain: Intuition - if I change a value by this much in the image, how much does this change in the result?

2 Lecture 3: Time to Contact, Focus of Expansion, Direct Motion Vision Methods, Noise Gain

2.1 Noise Gain

Example/motivation: **Indoor GPS**. Rather than using localization of position with satellites, use indoor cellular signals.

Fun fact: EM waves travel at 1 ns/foot.

Dilution of Precision:

- How far off GPS is w.r.t. your location.
- Important to note that your dilution of precision can vary in different directions - e.g. horizontal precision is oftentimes greater than vertical position.

2.2 Forward and Inverse Problems of Machine Vision

2.2.1 Scalar Case

One way to conceptualize the goals of machine vision, as well as to highlight why noise gain is important, is by considering the following problems with some one-dimensional input x and an output $y = f(x)$:

- The **forward problem**: $x \rightarrow y \stackrel{\Delta}{=} f(x)$
- The **inverse problem***: $y \stackrel{\Delta}{=} f(x) \rightarrow x$
  *(this term comes up a lot in machine vision, computer graphics, and robotics)

In machine vision, we oftentimes focus on solving the **inverse problem**, rather than the **forward problem**. Intuitively, we usually observe some $y = f(x)$, and from this infer the latent parameters x using our model f.

When it is possible to express the inverse of a function in closed form or via a matrix/coefficient, we can simply solve the **inverse problem** using: $x = f^{-1}(y)$.

More importantly, to build a robust machine vision system to solve this inverse problem, it is critical that small perturbations in $y = f(x)$ do not need to large changes in x. Small perturbations need to be taken into account in machine vision problems because the sensors we use exhibit **measurement noise**. The concept of **noise gain** can come in to help deal with this uncertainty.

Consider a perturbation δy that leads to a perturbation δx when we solve the inverse problem. In the limit, as $\delta \in \mathbb{R} \rightarrow 0$, then we arrive at the definition of noise gain:

$$\text{noise gain} = \frac{\delta x}{\delta y} = \frac{1}{f'(x)} = \frac{1}{\frac{df(x)}{dx}} \tag{1}$$

Like other concepts/techniques we've studied so far, let's understand when this system fails. Below are two cases; we encourage to consider why they fail from both a mathematical and intuitive perspective (hint: for the mathematical component, look at the formula above, and for the intuitive component, think about how the effect on x from a small change in y in a curve that is nearly flat):

- $f'(x) = 0$ (flat curve)
- $f'(x) \approx 0$ (nearly flat curve)

2.2.2 Vector Case

Now that we've analyzed this problem in the scalar case, let us now consider it in the vector/multi-dimensional case. Since images are inherently multidimensional, this more general vector case is where we will find ourselves.

First, we can restate these problems:

- **Forward Problem**: $\mathbf{x} = \mathbf{M}\mathbf{b}, \mathbf{M} \in \mathbb{R}^{m \times n}$ for $m, n \in \mathbb{N}$
- **Inverse Problem**: $\mathbf{b} = \mathbf{M}^{-1}\mathbf{x}, \mathbf{M} \in \mathbb{R}^{m \times n}$ for $m, n \in \mathbb{N}$

But how good is this answer/approach? If $\mathbf{x}$ changes, how much does $\mathbf{b}$ change? We quantify noise gain in this case as follows:

$$\text{“noise gain”} \rightarrow \frac{||\delta \mathbf{b}||}{||\delta \mathbf{x}||}, \delta \in \mathbb{R} \tag{2}$$

***NOTE**: This multidimensional problem is more nuanced because we may be in the presence of an **anisotropic** (spatially non-uniform) noise gain - e.g. there could be little noise gain in the x_1 direction, but a lot of noise gain in the x_2 direction.

As in the previous case, let's analyze when this approach fails. To do this, let's consider M^{-1} to help build intuition for why this fails:

$$M^{-1} = \frac{1}{\det|M|}\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \tag{3}$$

Let's ignore the specific entries of M^{-1} for now, and focus on the fact that we need to compute the determinant of M. When is this determinant zero? This will be the case whenever there exists **linear dependence** in the columns of **M**. As we saw before, two cases that can yield to poor performance will be:

- $\det|M| = 0$: This corresponds to a non-invertible matrix, and also causes the noise term to blow up.
- $\det|M| \approx 0$: Though this matrix may be invertible, it may cause numerical instability in the machine vision system, and can also cause the noise term to blow up.

Let's also revisit, just as a refresher, the inverse of a 2 × 2 matrix:

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{4}$$

$$\tag{5}$$

$$\mathbf{A}^{-1} = \frac{1}{\det\mathbf{A}}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{6}$$

Now let's verify that this is indeed the inverse:

$$\mathbf{A}^{-1}\mathbf{A} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \frac{1}{ad-bc}\begin{bmatrix} ad-bc & -ab+ab \\ cd-cd & ad-bc \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}_2 \tag{7}$$

2.3 Review from Lecture 2

Before we dive into the next set of concepts, let's also revisit some of the concepts discussed in the previous lectures.

2.3.1 Two-Pixel Motion Estimation, Vector Form

First, let's recall the two pixel motion estimation set of equations:

$$uE_{x_1} + vE_{y_1} + E_{t_1} = 0 \tag{8}$$

$$uE_{x_2} + vE_{y_2} + E_{t_2} = 0 \tag{9}$$

Rewritten in matrix form:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{E_{x_1}E_{y_2} - E_{y_1}E_{x_2}}\begin{bmatrix} E_{y_2} & -E_{y_1} \\ -E_{x_2} & E_{x_1} \end{bmatrix} \tag{10}$$

Take note of the denominator on the right-hand side. Does this term look familiar to the determinant term above? We were indeed solving an instance of the **inverse problem**. If this determinant-like quantity $(E_{x_1}E_{y_2} - E_{y_1}E_{x_2})$ is small, the noise is greatly amplified. We saw that this happens when brightness gradients are similar to one another.

2.3.2 Constant Brightness Assumption, and Motion Equation Derivation

Recall the constant brightness assumption's mathematical formulation:

$$\textbf{Constant Brightness Assumption} \implies \frac{dE}{dt} = 0 \tag{11}$$

*(Please note the quantity above is a total derivative.)

Intuition Behind This: As the object/camera moves, the physical properties of the camera do not change and therefore the total derivative of the brightness w.r.t. time is 0. From the chain rule, we can rewrite this total derivative assumption:

$$\frac{dE}{dt} = 0 \implies uE_x + vE_y + E_t = 0 \tag{12}$$

*(Recall this is for when x and y are parameterized w.r.t. time, i.e. $x = x(t), y = y(t)$.)

The above constraint is known as the **Brightness Change Constraint Equation (BCCE)**.

2.3.3 Optical Mouse Problem

Recall our motion estimation problem with the optical mouse, in which our objective is no longer to find the point where the **BCCE** is strictly zero (since images are frequently corrupted by noise through sensing), but to minimize the LHS of the BCCE, i.e:

$$\min_{u,v}\{J(u,v) \triangleq \iint (uE_x + vE_y + E_t)^2 dxdy\} \tag{13}$$

We solve the above using unconstrained optimization and by taking a "least-squares" approach (hence why we square the LHS of the BCCE). Solve by setting the derivatives of the two optimizing variables to zero:

$$\frac{dJ(u,v)}{du} = 0, \frac{dJ(u,v)}{dv} = 0 \tag{14}$$

2.3.4 Perspective Projection

Recall the perspective projection equations in the scalar form:

$$\frac{x}{f} = \frac{X}{Z} \textbf{ (x-component)}, \frac{y}{f} = \frac{Y}{Z} \textbf{ (y-component)} \tag{15}$$

*(Note that capital coordinates are in the world space, and lowercase coordinates are in the image space.)

What if these quantities are changing in the world w.r.t. time? Take time derivatives:

$$\frac{1}{f}\frac{dx}{dt} = \frac{1}{Z}\frac{dX}{dt} - \frac{1}{Z^2}X\frac{dZ}{dt} \tag{16}$$

Writing these for x and y:

- **x**: $\frac{1}{f}u = \frac{1}{Z}U - \frac{W}{Z}\frac{X}{Z}$
- **y**: $\frac{1}{f}v = \frac{1}{Z}V - \frac{W}{Z}\frac{Y}{Z}$

Note again the following definitions:

- $u \rightarrow$ image velocity in the x direction
- $v \rightarrow$ image velocity in the y direction
- $U \rightarrow$ world velocity in the X direction
- $V \rightarrow$ world velocity in the Y direction

When are these points in (u, v) space interesting? When $u = v = 0$ - this is the **Focus of Expansion (FOE)**. The **FOE** given by (x_0, y_0) in two dimensions:

$$(x_0, y_0) = (\frac{f}{Z}\frac{U}{W}, \frac{f}{Z}\frac{V}{W}) \tag{17}$$

2.4 Time to Contact (TTC)

In the previous lecture, we discussed the derivation of Time to Contact (TTC) in terms of meters:

$$\frac{Z}{W} \triangleq = \frac{Z}{\frac{dZ}{dt}} = \frac{\text{meters}}{\frac{\text{meters}}{\text{seconds}}} = \text{seconds} \tag{18}$$

Let us express the inverse of this **Time to Contact (TTC)** quantity as C:

$$C \triangleq \frac{W}{Z} = \frac{1}{\text{TTC}} \tag{19}$$

Let us now suppose we parameterize our spatial velocities u and v according to $u = Cx, v = Cy$, where C is the inverse TTC value we introduced above. Then, substituting these into the BCCE equation, we have:

$$\textbf{Recall BCCE:} \quad uE_x + vE_y + E_t = 0 \tag{20}$$

$$\textbf{Substitute:} \quad C(xE_x + yE_y) + E_t = 0 \tag{21}$$

$$\textbf{Solve for } C\textbf{:} \quad C = -\frac{E_t}{xE_x + yE_y} \tag{22}$$

The denominator in the derivation of C is the **"radial gradient"**:

$$g = xE_x + yE_y = (x, y) \cdot (E_x, E_y) \tag{23}$$

Building Intuition: If we conceptualize 2D images as topographic maps, where brightness is the third dimension of the surface (and the spatial dimensions x and y comprise the other two dimensions), then the brightness gradient is the direction of steepest **ascent** up the brightness surface.

Another note: (x, y) is a radial vector, say in a polar coordinate system. Hence why the above dot product term is coined the name "radial gradient". This gradient is typically normalized by its L_2/Euclidean norm to illustrate the multiplication of the brightness gradients with a radial unit vector):

$$g = \sqrt{x^2+y^2}(\frac{x}{\sqrt{x^2+y^2}}, \frac{y}{\sqrt{x^2+y^2}}) \cdot (E_x, E_y) \tag{24}$$

This g quantity can be thought of as: "How much brightness variation is in an outward direction from the center of the image?"

For a more robust estimate, let us again employ the philosophy that estimating from more data points is better. We'll again take a least-squares approach, and minimize across the entire image using the parameterized velocities we had before. In this case, since we are solving for inverse Time to Contact, we will minimize the error term over this quantity:

$$\min_C \{J(C) \triangleq \iint (C(xE_x + yE_y) + E_t)^2 dxdy\} \tag{25}$$

Without the presence of measurement noise, the optimal value of C gives us an error of zero, i.e. perfect adherence to the **BCCE**. However, as we've seen with other cases, this is not the case in practice due to noise corruption. We again will use unconstrained optimization to solve this problem.

Taking the derivative of the objective $J(C)$ and setting it to zero, we obtain:

$$\frac{dJ(C)}{dC} = 0 \implies 2\iint (C(xE_x + yE_y) + E_t)(xE_x + yE_y)dxdy = 0 \tag{26}$$

This in turn gives us:

$$C\iint (xE_x + yE_y)^2 dxdy + \iint (xE_x + yE_y)E_t dxdy = 0 \tag{27}$$

$$\frac{1}{\text{TTC}} = C = -\frac{\iint (xE_x + yE_y)E_t dxdy}{\iint (xE_x + yE_y)^2 dxdy} \tag{28}$$

A few notes here:

- E_t can be computed by taking differences between a pixel at different points in time.
- To implement a framework like this, we can do so with **accumulators**.
- This is an instance of **"direct computation"**.
- When objects/important components of a scene are far away, we can combine image pixels and run these algorithms at **multiple scales** - this allows us to compute/analyze motion velocities over different timescales. This can also be helpful for building more computationally-tractable motion estimation implementations, since the number of pixels over which computations must occur can be reduced quadratically with the scale.
- Note one problem we run into with this approach - each pixel we apply this estimation approach to introduces one equation, but two unknowns.
- Note that neighboring pixels typically exhibit similar behavior to one another.

Let's briefly return to the concept of the radial gradient. When is this zero?

- $E = 0$ everywhere (coal mine)
- $(x, y) \cdot (E_x, E_y) = 0$ (radial gradient is zero)

Now, let's take a more general case when we have world motion, i.e. $U \neq 0, V \neq 0$. For our two components x and y:

- **x-component**:

$$\frac{u}{f} = \frac{U}{Z} - \frac{X}{f}\frac{W}{Z} \tag{29}$$

$$u = \frac{fU}{Z} - X\frac{W}{Z} \tag{30}$$

$$u = A - XC \tag{31}$$

$$\text{Where: } A \triangleq \frac{fU}{Z}, C \triangleq \frac{W}{Z} \tag{32}$$

- **y-component**:

$$\frac{v}{f} = \frac{V}{Z} - \frac{Y}{f}\frac{W}{Z} \tag{33}$$

$$v = \frac{fV}{Z} - Y\frac{W}{Z} \tag{34}$$

$$v = B - YC \tag{35}$$

$$\text{Where: } B \triangleq \frac{fV}{Z}, C \triangleq \frac{W}{Z} \tag{36}$$

Note that for the world quantities A and B, we also have the following identities (note that the Focus of Expansion (FOE) is given by the point (x_0, y_0):

- $A = \frac{fU}{Z} = Cx_0$
- $B = \frac{fV}{Z} = Cy_0$

Building Intuition: "As I approach the wall, it will loom outward and increase in size."

Now returning to our **BCCE**, this time with A, B, and C:

$$AE_x + BE_y + C(xE_x + yE_y) + E_t = 0 \tag{37}$$

We can again use least-squares to minimize the following objective enforcing the BCCE. This time, our optimization aim is to minimize the objective function $J(A, B, C)$ using the quantities A, B, and C:

$$\min_{A,B,C}\{J(A,B,C)\} \triangleq \iint (AE_x + BE_y + C(xE_x + yE_y) + E_t)^2 dxdy\} \tag{38}$$

Use unconstrained optimzation with calculus and partial derivatives to solve. Since we have three variables to optimize over, we have three first-order conditions (FOCs):

- $\frac{dJ(A,B,C)}{dA} = 0$
- $\frac{dJ(A,B,C)}{dB} = 0$
- $\frac{dJ(A,B,C)}{dC} = 0$

Using the Chain rule for each of these FOCs, we can derive and rewrite each of these conditions to obtain 3 equations and 3 unknowns. Note that $G \triangleq xE_x + yE_y$.

- **A** variable:

$$2\iint (AE_x + BE_y + C(xE_x + yE_y))E_x = 0 \tag{39}$$

$$A\iint E_x^2 + B\iint E_xE_y + C\iint GE_x = -\iint E_xE_t \tag{40}$$

- **B** variable:

$$2\iint (AE_x + BE_y + C(xE_x + yE_y))E_x = 0 \tag{41}$$

$$A\iint E_yE_x + B\iint E_y^2 + C\iint GE_y = -\iint E_yE_t \tag{42}$$

- **C** variable:

$$2\iint (AE_x + BE_y + C(xE_x + yE_y))E_x = 0 \tag{43}$$

$$A\iint GE_x + B\iint GE_y + C\iint G^2 = -\iint GE_t \tag{44}$$

Putting all of these equations together:

$$\begin{aligned}
A\iint E_x^2 + B\iint E_xE_y + C\iint GE_x &= -\iint E_xE_t\\
A\iint E_yE_x + B\iint E_y^2 + C\iint GE_y &= -\iint E_yE_t\\
A\iint GE_x + B\iint GE_y + C\iint G^2 &= -\iint GE_t
\end{aligned}$$

This can be compactly written as a matrix-vector product equation:

$$\begin{bmatrix} \iint E_x^2 & \iint E_xE_y & \iint E_xG \\ \iint E_yE_x & \iint E_y^2 & \iint E_yG \\ \iint GE_x & \iint GE_y & \iint G^2 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = -\begin{bmatrix} \iint E_xE_t \\ \iint E_yE_t \\ \iint GE_t \end{bmatrix}$$

As in the time-to-contact problem above, this can again be implemented using **accumulators**.

Let's end on a fun fact: Did you know that optical mice have frame rates of 1800 fps?

3 Lecture 4: Fixed Optical Flow, Optical Mouse, Constant Brightness Assumption, Closed Form Solution

3.1 Review

Let's frame the topics today by briefly reviewing some concepts we've covered in the previous lectures. Feel free to skip this section if you already feel comfortable with the material.

- Image formation
  - Where in the image? Recall **perspective projection**:

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

    Differentiating this expression gives:

    $$\frac{u}{f} = \frac{U}{Z} - \frac{X}{Z}\frac{W}{Z}, \frac{v}{f} = \frac{V}{Z} - \frac{Y}{Z}\frac{W}{Z}$$

    From these, we can find the **Focus of Expansion (FOE)**, or, more intuitively: "The point in the image toward which you are moving."

    How long until we reach this point? This is given by **Time to Contact (TTC)**:

    $$\text{Time to Contact} = \frac{Z}{W} = \frac{1}{C}$$

  - How bright in the image? For this, let us consider an image solid, where the brightness function is parameterized by x, y, and t: $E(x, y, t)$.

3.1.1 Constant Brightness Assumption Review with Generalized Isophotes

Recall the **constant brightness** assumption, which says that the **total derivative** of brightness with respect to time is zero: $\frac{dE(x,y,t)}{dt} = 0$. By chain rule we obtain the **BCCE**:

$$\frac{dx}{dt}\frac{\partial E}{\partial x} + \frac{dy}{dt}\frac{\partial E}{\partial y} + \frac{\partial E}{\partial t} = 0$$

Recall our variables: $u \triangleq \frac{dx}{dt}, v \triangleq \frac{dy}{dt}$. Then BCCE rewritten in the standard notation we've been using:

$$uE_x + yE_y + E_t = 0$$

Recall our method of using **least-squares regression** to solve for optimal values of u, v that minimize the total computed sum of the LHS of the BCCE over the entire image (note that integrals become discrete in the presence of discretized pixels, and derivatives become differences):

$$u^*, v^* = \arg\min_{u,v} \iint (uE_x + yE_y + E_t)dxdy$$

The first-order conditions (FOCs) of this optimization problem give:

$$u\iint E_x^2 + v\iint E_xE_y = -\iint E_xE_t \quad (45)$$
$$u\iint E_yE_x + v\iint E_y^2 = -\iint E_yE_t \quad (46)$$

Written in matrix-vector form, our equations become:

$$\begin{bmatrix} \iint E_x^2 & \iint E_xE_y \\ \iint E_yE_x & \iint E_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \iint E_xE_t \\ \iint E_yE_t \end{bmatrix}$$

(**New**) Now, to introduce a new variation on this problem, let us suppose we have the following spatial parameterization of brightness (you'll see that this brightness function creates **linear isophotes**) for **linear f**:

$$E(x,y) = f(ax+by)$$

If f is differentiable over the domain, then the spatial derivatives E_x and E_y can be computed as follows, using the chain rule:

- $E_x = f'(ax+by)a$
- $E_y = f'(ax+by)b$

Where f' is the derivative of this scalar-valued function (i.e, we can define the input to be $z = ax + by$, and the derivative f' is therefore equivalent to $\frac{df(z)}{dz}$).

Isophote Example: If $E(x,y) = ax + by + c$, for $a, b, c \in \mathbb{R}_+$, then the **isophotes** of this brightness function will be linear.

3.1.2 Time to Contact (TTC) Review

Recall the following symbols/quantities from TTC:

- $C = \frac{Z}{w}$
- TTC $= \frac{w}{Z} = \frac{1}{Z}\frac{dZ}{dt} = \frac{d}{dt}log_e(z)$, therefore we can simply take the slope of the line corresponding to the logarithm of Z to compute TTC.

Now, let's suppose that objects are moving both in the world and in the image. Let's denote s as our image coordinate and S as our world coordinate. Then:

$$\frac{s}{f} = \frac{S}{Z}$$

Then we can write:

$$sZ + sf = 0$$

Differentiating:

$$Z\frac{ds}{dt} + s\frac{dZ}{dt} = 0 \implies \frac{\frac{ds}{dt}}{S} = \frac{\frac{dZ}{dt}}{Z}$$

The above relationship between derivative ratios can be interpreted as: "The change in the image's size is the same as the change in distance."

3.2 Increasing Generality of TTC Problems

Let us now consider adding some additional generality to the Time to Contact (TTC) problem. We've already visited some of these cases before:

- **Simple case**: Solving for C
- **More General case**: Solving for A, B, C
- **Even More General case**: (Motivation) What if the optical axis isn't perpendicular to the wall? What if the camera plane is tilted, e.g. $Z = aX + bY + C$ for some $a, b, C \in \mathbb{R}$? In this case, we can solve the problem **numerically** rather than through a **closed-form expression**.

Another motivating question for developing TTC methods: What if the surface is non-planar? This is a common scenario for real-world TTC systems. In this case, we have two options:

- Parameterize the geometric models of these equations with **polynomials**, rather than **planes**.
- Leave the planar solution, and look for other ways to account for errors between the modeled and true surfaces.

In practice, the second option here actually works better. The first option allows for higher modelling precision, but is less robust to local optima, and can increase the sensitivity of the parameters we find through least-squares optimization.

If you want to draw an analog to machine learning/statistics, we can think of modeling surfaces with more parameters (e.g. polynomials rather than planes) as creating a model that will **overfit** or not generalize well to the data it learns on, and create a problem with too many unknowns and not enough equations.

3.3 Multiscale and TTC

If you recall from last lecture, we saw that TTC and FOE estimation "fell apart" as we got really close. This is due to measurement sensitivity and the fact that the pixels occupy increasingly more space as we get closer and closer. This is where **multiscale** can help us - it enables us to use more coarse resolutions for these estimation problems. The implicit averaging done through downsampling allows us to "smoooth out" any measurement noise that may be present, and will consequently reduce the magnitide of pixel brightness gradients.

Additionally, multiscale is computationally-efficient: Using the infinite geometric series, we can see that downsampling/downscaling by a factor of 2 each time and storing all of these smaller image representations requires only 33% more stored data than the full size image itself:

$$\sum_{n=0}^{\infty}((\frac{1}{2})^2)^n = \frac{1}{1-\frac{1}{4}} = \frac{4}{3} = 1 + \frac{1}{3}$$

More generally, for any downsampling factor $r \in \mathbb{N}$, we only add $\frac{1}{r^2-1} \times 100\%$ amount of additional data:

$$\sum_{n=0}^{\infty}(\frac{1}{r^2})^n = \frac{1}{1-\frac{1}{r^2}} = \frac{r^2}{r^2-1} = \frac{(r^2-1)+1}{r^2-1} = 1 + \frac{1}{r^2-1}$$

(Note that we have r^2 rather than r in the denominator because we are downsampling across both the x and y dimensions.)

3.3.1 Aliasing and Nyquist's Sampling Theorem

Though multiscale is great, we also have to be mindful of **aliasing**. Recall from 6.003 (or another Signals and Systems course) that aliasing causes overlap and distortion between signals in the frequency domain, and it is required that we sample at a spatial frequency that is high enough to not produce aliasing artifacts.

Nyquist's Sampling Theorem states that we must sample at twice the frequency of the highest-varying component of our image to avoid aliasing and consequently reducing spatial artifacts.

3.3.2 Applications of TTC

A few more applications of TTC:

- Airplane Wing Safety - Using TTC to make sure wings don't crash into any objects at airports.
- NASA TTC Control - TTC systems were used to ensure that NASA's payload doesn't crash into the surface of earth/other planets/moons. In this application, feedback control was achieved by setting a nominal "desired TTC" and using an amplifier, imaging, and TTC estimation to maintain this desired TTC.
- Autonomous Vehicles - e.g. a vehicle is coming out of a parking lot and approaching a bus - how do we control when/if to brake?

Let's discuss the NASA TTC Control example a little further. Using our equation for TTC:

$$\frac{Z}{W} = T$$

We can rewrite this as a first-order Ordinary Differential Equation (ODE):

$$\frac{Z}{\frac{dZ}{dt}} = T \implies \frac{dZ}{dt} = \frac{1}{T}Z$$

Since the derivative of Z is proportional to Z, the solution to this ODE will be an exponential function in time:

$$Z(t) = Z_0 e^{\frac{-t}{T}}$$

Where Z_0 depends on the initial conditions of the system.

This method requires that deceleration is not uniform, which is not the most energy efficient approach for solving this problem. As you can imagine, energy conservation is very important in space missions, so let's next consider a constant deceleration approach. Note that under constant deceleration, we have $\frac{d^2z}{dt^2} \overset{\Delta}{=} a = 0$. Then we can express the first derivative of Z w.r.t. t as:

$$\frac{dZ}{dt} = at + v_0$$

Where v_0 is an initial velocity determined by the boundary/initial conditions. Here we have the following boundary condition:

$$\frac{dZ}{dt} = a(t - t_0)$$

This boundary condition gives rise to the following solution:

$$Z = \frac{1}{2}at^2 - at_0t + c = \frac{1}{2}a(t - t_0)^2$$

Therefore, the **TTC** for this example becomes:

$$T = \frac{Z}{\frac{dZ}{dt}} = \frac{\frac{1}{2}a(t - t_0)^2}{a(t - t_0)} = \frac{1}{2}(t - t_0)$$

3.4 Optical Flow

Motivating question: What if the motion of an image is non-constant, or it doesn't move together? We have the **Brightness Change Constraint Equation (BCCE)**, but this only introduces one constraint to solve for two variables, and thus creates an under-constrained/ill-posed problem.

How can we impose additional constraints? To do this, let us first understand how motion relates across pixels, and information that they share. Pixels don't necessarily move exactly together, but they move together in similar patterns, particularly if pixels are close to one another. We'll revisit this point in later lectures.

What Else Can We Do? One solution is to divide the images into equal-sized patches and apply the **Fixed Flow Paradigm**, as we've done with entire images before. When selecting patch size, one trade-off to be mindful of is that the smaller the patch, the more uniform the brightness patterns will be across the patch, and patches may be too uniform to detect motion (note: this is equivalent to the matrix determinants we've been looking at evaluating to zero/near-zero).

3.5 Vanishing Points (Perspective Projection)

Before we dive into what vanishing points are, let's discuss why they're useful. Vanishing points can be useful for:

- Camera calibration - has applications to robotics, autonomous vehicles, photogrammetry, etc.
- Finding relative orientation between two coordinate frames/systems

Now, let's discuss what it is. Suppose we have several parallel lines in the world, and we image them by projecting them onto the 2D image plane. Then **vanishing points** are the points in the image (or, more commonly, outside of the image) where these lines converge to in the image. To discuss these mathematically, let's first discuss about defining lines in a 3D world:

- **Vector Form**: $\mathbf{R} = \mathbf{R_0} + s\hat{\mathbf{n}}$
  Here, we can express this using our standard vector form of perspective projection:

$$\begin{aligned}\frac{1}{f}\mathbf{r} &= \frac{1}{\mathbf{R}\cdot\hat{\mathbf{z}}}\mathbf{R} \\ &= \frac{1}{(\mathbf{R_0} + s\hat{\mathbf{n}})\cdot\hat{\mathbf{n}}}(\mathbf{R_0} + s\hat{\mathbf{n}})\end{aligned}$$

- **Parametrically**: $(x_0 + \alpha s, y_0 + \beta s, z_0 + \gamma s)$
  Here, we can expand this to our standard Cartesian form of perspective projection to apply our transformation:

$$\begin{aligned}\frac{x}{f} &= \frac{1}{Z_0 + \gamma s}(x_0 + \alpha s) \\ \frac{y}{f} &= \frac{1}{Z_0 + \gamma s}(y_0 + \beta s)\end{aligned}$$

To build intuition, let's consider what happens when we travel far along the lines (i.e. as s gets very large) in our parametric definition of lines:

- $\lim_{x\to\infty} \frac{x}{f} = \lim_{x\to\infty} \frac{1}{Z_0+\gamma s}(x_0 + \alpha s) = \frac{\alpha s}{\gamma s} = \frac{\alpha}{\gamma}$ **(x-coordinate)**
- $\lim_{y\to\infty} \frac{y}{f} = \lim_{y\to\infty} \frac{1}{Z_0+\gamma s}(x_0 + \beta s) = \frac{\beta s}{\gamma s} = \frac{\beta}{\gamma}$ **(x-coordinate)**

The 2D point $(\frac{\alpha}{\gamma}, \frac{\beta}{\gamma})$ is the vanishing point in the **image plane**. As we move along the line in the world, we approach this point in the image, but we will never reach it. More generally, we claim that **parallel lines in the world have the same vanishing point in the image**.

3.5.1 Applications of Vanishing Points

Let's now discuss some of these applications in greater detail.

- **Protecting Pedestrians on a Road Side**: To protect pedestrians, the camera must transform its coordinate system. This transformation can be found using **vanishing points**.
- **Camera Calibration**: One way to calibrate a camera is to solve for the **Center of Projection (COP)** in the image space, using perspective projection. Calibration is typically achieved through **calibration objects**.

3.6 Calibration Objects

Let's discuss two calibration objects: **spheres** and **cubes**:

3.6.1 Spheres

:

- If image projection is directly overhead/straight-on, the projection from the world sphere to the image plane is a circle. If it is not overhead/straight on, it is elliptic.
- Relatively easy to manufacture

3.6.2 Cube

:

- Harder to manufacture, but generally a better calibration object than a sphere.
- Cubes can be used for detecting edges, which in turn can be used to find vanishing points (since edges are lines in the world).
- Cubes have three sets of four parallel lines/edges each, and each of these sets of lines are orthogonal to the others. This implies that we will have three **vanishing points** - one for each set of parallel lines.
- For each of these sets of lines, we can pick a line that goes through the Center of Projection (COP), denoted $\mathbf{p} \in \mathbb{R}^3$ (in the world plane). We can then project the COP onto the image plane (and therefore now $\mathbf{p} \in \mathbb{R}^{\nvDash}$).
- Let us denote the **vanishing points** of the cube in the image plane as $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^2$. Then, because of orthogonality between the different sets of lines, we have the following relations between our three vanishing points and $\mathbf{p}$:

  - $(\mathbf{p} - \mathbf{a}) \cdot (\mathbf{p} - \mathbf{b}) = 0$
  - $(\mathbf{p} - \mathbf{b}) \cdot (\mathbf{p} - \mathbf{c}) = 0$
  - $(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{a}) = 0$

  In other words, the difference vectors between $\mathbf{p}$ and the vanishing points are all at right angles to each other.

  To find $\mathbf{p}$, we have three equations and three unknowns. We have terms that are quadratic in $\mathbf{p}$. Using **Bézout's Theorem** (*The maximum number of solutions is the product of the polynomial order of each equation in the system of equations*), we have $(2)^3 = 8$ possible solutions for our system of equations. More generally:

  $$\text{number of solutions} = \prod_{e=1}^{E} o_e$$

  Whhere E is the number of equations and o_e is the polynomial order of the e^{th} equation in the system.

  This is too many equations to work with, but we can subtract these equations from one another and create a system of 3 linearly dependent equations. Or, even better, we can leave one equation in its quadratic form, and 2 in their linear form, and this maintains linear independence of this system of equations:

  - $(\mathbf{a} - \mathbf{p}) \cdot (\mathbf{c} - \mathbf{b}) = 0$
  - $(\mathbf{b} - \mathbf{p}) \cdot (\mathbf{a} - \mathbf{c}) = 0$
  - $(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{a}) = 0$

3.7 Additional Topics

We'll also briefly recap some of the topics discussed in our synchronous section today. These topics are not meant to introduce new concepts, but are designed to generalize the concepts we've discussed such that they can be adapted for a broader range of applications.

3.7.1 Generalization: Fixed Flow

The motivating example for this generalization is a **rotating optimal mouse**. We'll see that instead of just solving for our two velocity parameters u and v, we'll also need to solve for our rotational velocity, ω.

Suppose we are given the following parameterizations of velocity:

- $u = u_0 - wy$
- $v = v_0 + wx$

Note that we can also write the radial vector of x and y, as well as the angle in this 2D plane to show how this connects to rotation:

$$\begin{aligned}\mathbf{r} = (x, y) &= \sqrt{x^2 + y^2} \\ \theta &= \arctan 2(y, x) \\ \omega &= \frac{d\theta}{dt}\end{aligned}$$

With this rotation variable, we leverage the same least-squares approach as before over the entirety of the image, but now we also optimize over the variable for ω:

$$u_0^*, v_0^*, \omega^* = \arg \min_{u_0, v_0, \omega} \{J(u_0, v_0, \omega) \triangleq \iint (u_0 E_x + v_0 E_y + wH + E_t)^2 dxdy\}$$

Like the other least-squares optimization problems we've encountered before, this problem can be solved by solving a system of first-order conditions (FOCs):

- $\frac{dJ(u_0, v_0, \omega)}{du_0} = 0$
- $\frac{dJ(u_0, v_0, \omega)}{dv_0} = 0$
- $\frac{dJ(u_0, v_0, \omega)}{d\omega} = 0$

3.7.2 Generalization: Time to Contact (for $U = 0, V = 0, \omega \neq 0$)

Let's now revisit TTC, but with the following parameterization: $U \neq 0, V \neq 0$, image is of a tilted plane.

For this, we can write Z as a function of the world coordinates X and Y:

$$\begin{aligned}Z &= Z_0 + \frac{\partial Z}{\partial X} X + \frac{\partial Z}{\partial Y} Y \\ Z &= Z_0 + pX + qY\end{aligned}$$

Recall the following derivations for the image coordinate velocities u and v, which help us relate image motion in 2D to world motion in 3D:

- $\frac{u}{f} = \frac{U}{Z} - \frac{X}{Z}\frac{W}{Z}$
- $\frac{v}{f} = \frac{V}{Z} - \frac{Y}{Z}\frac{W}{Z}$

Some additional terms that are helpful when discussing these topics:

- **Motion Field**: Projection of 3D motion onto the 2D image plane.
- **Optical Flow**:
  - What we can sense
  - Describes motion in the image

We can transform this into image coordinates:

$$u = \frac{1}{2}(fU - xw), \;\; v = \frac{1}{2}(fV - yw)$$

Let's take $U = V = 0, u = -X\frac{w}{Z}, v = -Y\frac{W}{Z}$. Z (world coordinates) is not constant, so we can rewrite this quantity by substituting the image coordinates in for our expression for Z:

$$\begin{aligned} Z &= Z_0 + px + qy \\ &= Z_0 + p\frac{X}{f}Z + q\frac{Y}{f}Z \end{aligned}$$

Now, we can isolate Z and solve for its closed form:

$$\begin{aligned} Z(1 - p\frac{X}{f} - q\frac{Y}{f}) &= Z_0 \\ Z &= \frac{Z_0}{1 - p\frac{X}{f} - q\frac{Y}{f}} \end{aligned}$$

From this we can conclude that $\frac{1}{Z}$ is linear in x and y (the image coordinates, not the world coordinates). This is helpful for methods that operate on finding solutions to linear systems. If we now apply this to the **BCCE** given by $uE_x + vE_y + E_t = 0$, we can first express each of the velocities in terms of this derived expression for Z:

- $u = \frac{1}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})(-x\omega)$
- $v = \frac{1}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})(-y\omega)$

Applying these definitions to the BCCE:

$$\begin{aligned} 0 &= uE_x + vE_y + E_t \\ &= \frac{1}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})(-x\omega)E_x + \frac{1}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})(-y\omega)E_y + E_t \end{aligned}$$

Combining like terms, we can rewrite this constraint as:

$$0 = -\frac{\omega}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})(xE_x + yE_y) + E_t$$

Equivalently, dividing everything by -1:

$$0 = \frac{\omega}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})(xE_x + yE_y) - E_t$$

We can also express this with the "radial gradient" given by: $G \triangleq (xE_x + yE_y)$:

$$0 = \frac{\omega}{Z_0}(1 - p\frac{X}{f} - q\frac{Y}{f})G - E_t$$

For symbolic simplicity, take the following definitions:

- $R \triangleq \frac{w}{Z_0}$
- $P \triangleq -p\frac{w}{Z_0}$
- $Q \triangleq -q\frac{w}{Z_0}$

Using these definitions, the BCCE with this paramterization becomes:

$$0 = (R + Px + Qy)G - E_t$$

Now, we can again take our standard approach of solving these kinds of problems by applying least-squares to estimate the free variables P, Q, R over the entire continuous or discrete image space. Like other cases, this fails when the determinant of the system involving these equations is zero.

4 Lecture 5: TTC and FOR Montivision Demos, Vanishing Point, Use of VPs in Camera Calibration

In this lecture, we'll continue going over **vanishing points** in machine vision, as well as introduce how we can use brightness estimates to obtain estimates of a surface's orientation. We'll introduce this idea with **Lambertian surfaces**, but we can discuss how this can generalize to many other types of surfaces as well.

4.1 Robust Estimation and Sampling

We'll start by covering some of the topics discussed during lecture.

4.1.1 Line Intersection Least-Squares

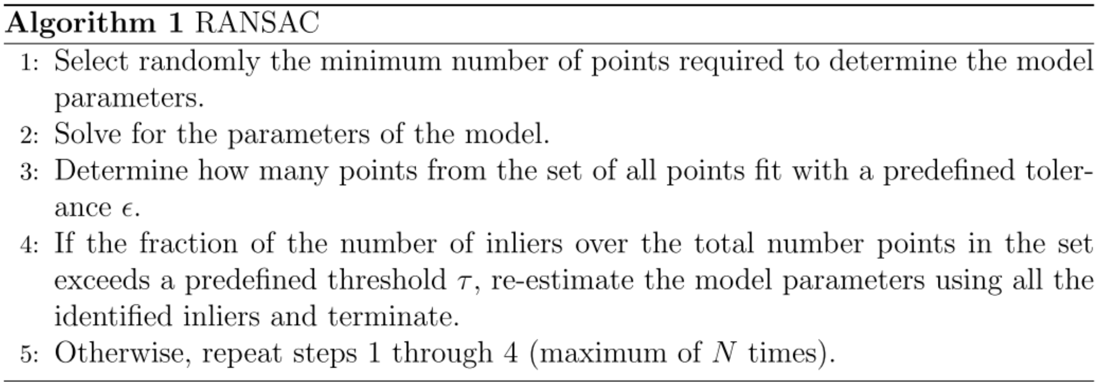
Helpful when considering multiple lines at a time - this optimization problem is given by:

$$\min_{x,y} \sum_i (x \cos\theta_i + y \sin\theta_i - \rho)^2 \tag{47}$$

4.1.2 Dealing with Outliers

Many of the approaches we've covered thus far are centered around the idea of making estimates from data. But recall that the data gathered by sensors can be noisy, and can be corrupted from phenomena such as crosstalk. Because of this noise, it is advantageous to distinguish and filter outliers from the inliers in our dataset, especially when we make estimates using Least-Squares.

A good choice of algorithm for dealing with outliers is **RANSAC**, or **Random Sample Consensus** [1]. This algorithm is essentially an iterative and stochastic variation of Least-Squares. By randomly selecting points from an existing dataset to fit lines and evaluate the fit, we can iteratively find line fits that minimize the least-squares error while distinguishing inliers from outliers.



Algorithm 1 RANSAC

```
1: Select randomly the minimum number of points required to determine the model
   parameters.
2: Solve for the parameters of the model.
3: Determine how many points from the set of all points fit with a predefined toler-
   ance $\epsilon$.
4: If the fraction of the number of inliers over the total number points in the set
   exceeds a predefined threshold $\tau$, re-estimate the model parameters using all the
   identified inliers and terminate.
5: Otherwise, repeat steps 1 through 4 (maximum of $N$ times).
```

Figure 1: Pseudocode for the RANSAC Algorithm.

4.1.3 Reprojection and Rectification

These two problems are highly important in mapmaking, and help for solving the equations:

$$\begin{aligned} R\mathbf{r} &= \mathbf{r}' \\ \mathbf{r} &= R^T\mathbf{r} \end{aligned}$$

Where $\mathbf{r}$ is the vector from the Center of Projection (COP) to the image plane.

4.1.4 Resampling

Resampling is also a valuable application in many facets of computer vision and robotics, especially if we seek to run any kind of **interpolation** or **subsampling** algorithms. Some approaches for this:

- **Nearest Neighbor**: This is a class of methods in which we interpolate based off of the values of neighboring points. This can be done spatially (e.g. by looking at adjacent pixels) as well as other image properties such as brightness and color. A common algorithm used here is **K-Nearest Neighbors (KNN)**, in which interpolation is done based off of the K-nearest points in the desired space.

- **Bilinear Interpolation**: An extension of linear interpolation used for functions in two-dimensional grids/of two variables (e.g. (x, y) or (i, j))) [2], such as the brightness or motion in images.

- **Bicubic Interpolation**: Similar to bilinear interpolation, bicubic interpolation is an extension of cubic interpolation of functions in two-dimensional grids/of two variables (e.g. (x, y) or (i, j))) [3], such as the brightness or motion in images. Bicubic interpolation tends to perform much better than bilinear interpolation, though at the cost of additional computational resources.

4.2 Magnification with TTC

- For the Montivision demo, note the following for the bars:
  - $A \rightarrow$ x-dimension for motion estimation problem
  - $B \rightarrow$ y-dimension for motion estimation problem
  - $C \rightarrow \frac{1}{\text{Time to Contact}}$
- Recall from the previous lectures that the percent change of size in the world is the percent change of size in the image. We can derive this through **perspective projection**. The equation for this is:

  $$\frac{s}{f} = \frac{S}{Z}$$

  Where s is the size in the image plane and S is the size in the world. Differentiating with respect to time gives us (using the chain rule):

  $$\frac{d}{dt}(sZ = fS) \rightarrow Z\frac{ds}{dt} + s\frac{dZ}{dt} = 0$$

  Rearranging terms:

  $$\frac{\frac{dz}{dt}}{Z}f = -\frac{\frac{ds}{dt}}{S}$$

  Recall that the intuition here is that the **rate of change of size is the same in the image and the world**.

4.2.1 Perspective Projection and Vanishing Points

Let's begin by defining a few terms. Some of these will be a review, but these review terms connect to new terms that we'll introduce shortly:

- **vanishing points**: These are the points in the image plane (or extended out from the image plane) that parallel lines in the world converge to.
- **Center of Projection (COP)**: This is where in the camera/sensor (not in the image plane) where incident projected light rays converge. An analog to the COP for human vision systems is your eyes. NOTE: COP and vanishing points are oftentimes different.
- **Principle Point**: The orthogonal projection of the Center of Projection (COP) onto the image plane.
- **f**: Similar to the focal length we've seen in other perspective projection examples, this f is the perpendicular distance from the COP to the image plane.

Recall that a common problem we can solve with the use of vanishing points is **finding the Center of Projection (COP)**. Solving this problem in 3D has 3 degrees of freedom, so consequently we'll try to solve it using three equations.

Intuitively, this problem of finding the Center of Projection can be thought of as finding the intersection of three spheres, each of which have two vanishing points along their diameters. Note that three spheres can intersect in up to two places - in this case we have defined the physically-feasible solution, and therefore the solution of interest, to be the solution above the image plane (the other solution will be a mirror image of this solution and can be found below the image plane).

Application of this problem: This problem comes up frequently in photogrammetry, in that simply having two locations as your vanishing points isn't enough to uniquely identify your location on a sphere.

4.2.2 Lines in 2D and 3D

Next, let's briefly review how we can parameterize lines in 2D and 3D:

- **2D**: (2 Degrees of Freedom)
  - $y = mx + c$
  - $ax + by + c = 0$
  - $\sin\theta x - \cos\theta y + \rho = 0$
  - If $\hat{\mathbf{n}} \triangleq (-\sin\theta, \cos\theta)^T$, then $\hat{\mathbf{n}} \cdot \mathbf{r} = \rho$.
- **3D**: (3 Degrees of Freedom)
  - $\hat{\mathbf{n}} = \rho$
  - $aX + bY + cZ + d = 0$

4.2.3 Application: Multilateration (MLAT)

This problem comes up when we estimate either our position or the position of other objects based off of Time of Arrival [4]. One specific application of this problem is localizing ourselves using distances/Time of Arrival of wifi access points inside buildings/other locations without access to GPS.

Like the other problems we've looked at, this problem can be solved by finding the intersection of 3 spheres. Let's begin with:

$$||\mathbf{r} - \mathbf{r}_i||_2 = p_i \ \forall \ i \in \{1, \cdots, N\}$$

Next, let's square both sides of this equation and rewrite the left-hand side with dot products:

$$\begin{aligned} ||\mathbf{r} - \mathbf{r}_i||_2^2 &= (\mathbf{r} - \mathbf{r}_i)^T(\mathbf{r} - \mathbf{r}_i) = p_i^2 \\ &= \mathbf{r} \cdot \mathbf{r} - 2\mathbf{r} \cdot \mathbf{r}_i + \mathbf{r}_i \cdot \mathbf{r}_i = p_i^2 \end{aligned}$$

Recall from Bezout's Theorem that this means that are 8 possible solutions here, since we have three equations of second-order polynomials. To get rid of the 2nd order terms, we simply subtract the equations:

$$\begin{aligned} &\mathbf{r} \cdot \mathbf{r} - 2\mathbf{r} \cdot \mathbf{r}_i + \mathbf{r}_i \cdot \mathbf{r}_i = \rho_i^2 \\ - \quad &\mathbf{r} \cdot \mathbf{r} - 2\mathbf{r} \cdot \mathbf{r}_j + \mathbf{r}_j \cdot \mathbf{r}_j = \rho_j^2 \quad \forall \ i, j \in \{1, 2, 3\}, i \neq j \end{aligned}$$

Subtracting these pairs of equations yields a linear equation in $\mathbb{R}$:

$$2\mathbf{r}(r_j - r_i) = (\rho_j^2 - \rho_i^2) - (R_j^2 - R_i^2)$$

(Where the scalar $R_j^2 \triangleq \mathbf{r}_j \cdot \mathbf{r}_j$.)

Putting these equations together, this is equivalent to finding the intersection of three different spheres:

$$\begin{bmatrix} (\mathbf{r}_2 - \mathbf{r}_1)^T \\ \mathbf{r}_3 - \mathbf{r}_2)^T \\ \mathbf{r}_1 - \mathbf{r}_3)^T \end{bmatrix} \mathbf{r} = \frac{1}{2} \begin{bmatrix} (\rho_2^2 - \rho_1^2) - (R_2^2 - R_1^2) \\ (\rho_3^2 - \rho_2^2) - (R_3^2 - R_2^2) \\ (\rho_1^2 - \rho_3^2) - (R_1^2 - R_3^2) \end{bmatrix}$$

However, even though we've eliminated the second-order terms from these three equations, we still have two solutions. Recall from linear algebra equations don't have a unique solution when there is redundancy or linear dependence between the equations. If we add up the rows on the right-hand side of the previous equation, we get 0, which indicates that the matrix on the left-hand side is singular:

$$A \triangleq \begin{bmatrix} (\mathbf{r}_2 - \mathbf{r}_1)^T \\ \mathbf{r}_3 - \mathbf{r}_2)^T \\ \mathbf{r}_1 - \mathbf{r}_3)^T \end{bmatrix} \in \mathbb{R}^{3 \times 3}$$

To solve this linear dependence problem, we again use **Bezout's Theorem** and keep one of the second-order equations:

$$
\begin{aligned}
&(\mathbf{r}-\mathbf{r}_1)\cdot(\mathbf{r}-\mathbf{r}_2)=0\\
&(\mathbf{r}-\mathbf{r}_2)\cdot(\mathbf{r}-\mathbf{r}_3)=0\\
&(\mathbf{r}-\mathbf{r}_2)\cdot(\mathbf{r}-\mathbf{r}_2)=0\rightarrow(\mathbf{r}-\mathbf{r}_2)\perp(\mathbf{r}_3-\mathbf{r}_1)
\end{aligned}
$$

I.e. the plane passes through $\mathbf{r}_2$ - this intersecting point is the solution and is known as the **orthocenter** or the **principal point**. Now, all we need is to find the quantity f to find the Center of Projection.

Next, note the following relations between the vanishing points in the inverse plane and $\hat{\mathbf{z}}$, which lies perpendicular to the image plane:

$$
\begin{aligned}
\mathbf{r}_1\cdot\hat{\mathbf{z}}&=0\\
\mathbf{r}_2\cdot\hat{\mathbf{z}}&=0\\
\mathbf{r}_3\cdot\hat{\mathbf{z}}&=0
\end{aligned}
$$

What else is this useful for? Here are some other applications:

- Camera calibration (this was the example above)
- Detecting image cropping - e.g. if you have been cropped out of an image
- Photogrammetry - e.g. by verifying if the explorer who claimed he was the first to make it "all the way" to the North Pole actually did (fun fact: he didn't).

Next, now that we've determined the principal point, what can we say about f (the "focal length")?

For this, let us consider the 3D simplex, which is triangular surface in $\mathbb{R}^3$ given by the unit vectors:

$$
\begin{aligned}
e_1&\triangleq\begin{bmatrix}1&0&0\end{bmatrix}^T\\
e_2&\triangleq\begin{bmatrix}0&1&0\end{bmatrix}^T\\
e_3&\triangleq\begin{bmatrix}0&0&1\end{bmatrix}^T
\end{aligned}
$$

Using this 3D simplex, let us suppose that each side of the triangles formed by this simplex take length $v=\sqrt{2}$, which is consistent with the l_2 norm of the triangles spanned by the unit simplex ($\sqrt{1^2+1^2}=\sqrt{2}$).

Next, we solve for f by finding the value of a such that the dot product between the unit vector perpendicular to the simplex and a vector of all a is equal to 1:

$$
\begin{aligned}
\begin{bmatrix}a&a&a\end{bmatrix}^T\begin{bmatrix}\frac{1}{\sqrt{3}}&\frac{1}{\sqrt{3}}&\frac{1}{\sqrt{3}}\end{bmatrix}^T&=1\\
\frac{3a}{\sqrt{3}}&=1\\
a=\frac{\sqrt{3}}{3}&=\frac{1}{\sqrt{3}}
\end{aligned}
$$

This value of $a=\frac{1}{\sqrt{3}}=f$. Then we can relate the lengths of the sides v (which correspond to the magnitudes of the vectors between the principal point and the vanishing points ($||\mathbf{r}-\mathbf{r}_i||_2$)) and f:

$$
v=\sqrt{2}, f=\frac{1}{\sqrt{3}}\implies f=\frac{v}{\sqrt{6}}
$$

With this, we've now computed both the principal point and the "focal length" f for camera calibration.

4.2.4 Application: Understand Orientation of Camera w.r.t. World

For this problem, the goal is to transform from the world frame of reference to the camera frame of reference, i.e. find a frame of reference such that the unit vectors $\hat{\mathbf{x}}^c,\hat{\mathbf{y}}^c,\hat{\mathbf{z}}^c$ have the following orthogonality properties:

$$
\begin{aligned}
\hat{\mathbf{x}}^c\cdot\hat{\mathbf{y}}^c&\approx0\\
\hat{\mathbf{y}}^c\cdot\hat{\mathbf{z}}^c&\approx0\\
\hat{\mathbf{x}}^c\cdot\hat{\mathbf{z}}^c&\approx0
\end{aligned}
$$

(Note that the c superscript refers to the camera coordinate system.) If the location of the Center of Projection (COP) is given above the image plane as the vector $\mathbf{p}^c$ and the vanishing points are given as vectors $\mathbf{r}_1^c$, $\mathbf{r}_2^c$, and $\mathbf{r}_3^c$ (note that these vanishing points must be in the same frame of reference in order for this computation to carry out), then we can derive expressions for the unit vectors through the following relations:

$$
\begin{aligned}
(\mathbf{p}^c - \mathbf{r}_1) /\!/ \hat{\mathbf{x}}^c &\implies \hat{\mathbf{x}}^c = \frac{\mathbf{p}^c - \mathbf{r}_1^c}{||\mathbf{p}^c - \mathbf{r}_1^c||_2} \\
(\mathbf{p}^c - \mathbf{r}_2) /\!/ \hat{\mathbf{y}}^c &\implies \hat{\mathbf{y}}^c = \frac{\mathbf{p}^c - \mathbf{r}_2^c}{||\mathbf{p}^c - \mathbf{r}_2^c||_2} \\
(\mathbf{p}^c - \mathbf{r}_3) /\!/ \hat{\mathbf{z}}^c &\implies \hat{\mathbf{z}}^c = \frac{\mathbf{p}^c - \mathbf{r}_3^c}{||\mathbf{p}^c - \mathbf{r}_3^c||_2}
\end{aligned}
$$

Then, after deriving the relative transformation between the world frame (typically denoted w in robotics) and the camera frame (typically denoted c in robotics), we can express the principal point/Center of Projection in the camera coordinate frame:

$$\mathbf{r} = \alpha\hat{\mathbf{x}}^c + \beta\hat{\mathbf{y}}^c + \gamma\hat{\mathbf{z}}^c$$

Where (α, β, γ) are the coordinates in the object coordinate system (since $\hat{\mathbf{x}}^c$, $\hat{\mathbf{y}}^c$, and $\hat{\mathbf{z}}^c$ comprise the orthogonal basis of this coordinate system). Then we can express the relative coordinates of objects in this coordinate system:

$$\mathbf{r}' = \begin{bmatrix}\alpha & \beta & \gamma\end{bmatrix}^T$$

To transform between frames, we can do so via the following equation:

$$\mathbf{r} = \begin{bmatrix} -(\hat{\mathbf{x}}^c)^T \\ -(\hat{\mathbf{y}}^c)^T \\ -(\hat{\mathbf{z}}^c)^T \end{bmatrix} \mathbf{r}'$$

Note that the matrix defined by: $R \triangleq \begin{bmatrix} -(\hat{\mathbf{x}}^c)^T \\ -(\hat{\mathbf{y}}^c)^T \\ -(\hat{\mathbf{z}}^c)^T \end{bmatrix}$ is orthonormal, since these vector entries are orthogonal to one another. This matrix R is a rotation matrix, which means it is skew-symmetric, invertible, and its transpose is equal to its inverse: $R^{-1} = R^T$. Therefore, if we wanted to solve the reverse problem that we did above (finding **object coordinates** from **camera coordinates**), we can do so by simply taking the transpose of the matrix:

$$
\begin{aligned}
\mathbf{r}' &= \begin{bmatrix} -(\hat{\mathbf{x}}^c)^T \\ -(\hat{\mathbf{y}}^c)^T \\ -(\hat{\mathbf{z}}^c)^T \end{bmatrix}^{-1} \mathbf{r} \\
&= \begin{bmatrix} -(\hat{\mathbf{x}}^c)^T \\ -(\hat{\mathbf{y}}^c)^T \\ -(\hat{\mathbf{z}}^c)^T \end{bmatrix}^{T} \mathbf{r}
\end{aligned}
$$

4.3 Brightness

We'll now switch to the topic of brightness, and how we can use it for surface estimation. Recall that for a Lambertian surface (which we'll assume we use here for now), the power received by photoreceptors (such as a human eye or an array of photodiodes) depends both on the power emitted by the source, but also the angle between the light source and the object.

This is relevant for **foreshortening** (the visual effect or optical illusion that causes an object or distance to appear shorter than it actually is because it is angled toward the viewer [6]): the perceived area of an object is the true area times the cosine of the angle between the light source and the object/viewer. **Definition**: Lambertian Object: An object that appears equally bright from all viewing directions and reflects all incident light, absorbing none [5].

Let's look at a simple case: a **Lambertian surface**. If we have the brightness observed and we have this modeled as:

$$E = \hat{\mathbf{n}} \cdot \mathbf{s},$$

Where $\mathbf{s}$ is the vector between the light source and the object, then can we use this information to recover the surface orientation given by $\hat{\mathbf{n}}$. This unit vector surface orientation has degrees of freedom, since it is a vector in the plane.

It is hard to estimate this when just getting a single measurement for brightness. But what if we test different lighting conditions?:

$$\begin{aligned}E_1 &= \hat{\mathbf{n}} \cdot \mathbf{s}_1 \\ E_2 &= \hat{\mathbf{n}} \cdot \mathbf{s}_2 \\ \hat{\mathbf{n}} \cdot \hat{\mathbf{n}} &= ||\hat{\mathbf{n}}||_2 = 1\end{aligned}$$

This is equivalent, intuitively, to finding the intersection between two cones for which we have different angles, which forms a planar curve. We then intersect this planar curve with the unit sphere corresponding to the constraint $||\hat{\mathbf{n}}||_2 = 1$. By Bezout's Theorem, this will produce two solutions.

One problem with this approach, however, is that these equations are not **linear**. We can use the presence of **reflectance** to help us solve this problem. Let us define the following:

Definition: Albedo: This is the ratio of power out of an object divided by power into an object:

$$\text{“albedo”} \triangleq \rho \triangleq \frac{\text{power in}}{\text{power out}} \in [0, 1]$$

Though the definition varies across different domains, in this class, we define albedo to be for a specific orientation, i.e. a specific s_i.

Fun fact: Although an albedo greater than 1 technically violates the 2nd Law of Thermodynamics, **superluminous** surfaces such as those sprayed with flourescent paint can exhibit $\rho > 1$.

Using this albedo term, we can now solve our problem of having nonlinearity in our equations. Note that below we use three different measurements this time, rather than just two:

$$\begin{aligned}E_1 &= \rho\hat{\mathbf{n}} \cdot \mathbf{s}_1 \\ E_2 &= \rho\hat{\mathbf{n}} \cdot \mathbf{s}_2 \\ E_3 &= \rho\hat{\mathbf{n}} \cdot \mathbf{s}_3\end{aligned}$$

This creates a system of 3 unknowns and 3 Degrees of Freedom. We also add the following constraints:

$$\begin{aligned}\mathbf{n} &= \rho\hat{\mathbf{n}} \\ \hat{\mathbf{n}} &= \frac{\mathbf{n}}{||\mathbf{n}||_2}\end{aligned}$$

Combining these equations and constraints, we can rewrite the above in matrix/vector form:

$$\begin{bmatrix}-\hat{\mathbf{s}}_1^T \\ -\hat{\mathbf{s}}_2^T \\ -\hat{\mathbf{s}}_3^T\end{bmatrix}\mathbf{n} = \begin{bmatrix}E_1 \\ E_2 \\ E_3\end{bmatrix} \implies \mathbf{n} = \mathbf{S}^{-1}\begin{bmatrix}E_1 \\ E_2 \\ E_3\end{bmatrix} \quad \text{(Where } S \triangleq \begin{bmatrix}-\hat{\mathbf{s}}_1^T \\ -\hat{\mathbf{s}}_2^T \\ -\hat{\mathbf{s}}_3^T\end{bmatrix}\text{.)}$$

A quick note on the solution above: like other linear algebra-based approaches we've investigated so far, the matrix **S** above isn't necessarily invertible. This matrix is not invertible when light sources are in a **coplanar** orientation relative to the object. If this is the case, then the matrix **S** becomes singular/linearly dependent, and therefore non-invertible.

4.3.1 What if We Can't use Multiple Orientations?

In the case where the orientation of the camera, object, and light source cannot be changed (i.e. everything is fixed), or if it is expensive to generate a surface estimation from a new orientation, then another option is to use different **color sensors**. Most cameras have RGB (“Red-Green-Blue”) sensors, and thus we can use the same approach as above, where each color corresponds to a different orientation.

This RGB approach has a few issues:

- RGB has “crosstalk” between color channels (it can be hard to excite/activate a single channel for color without exciting the others as well).

- The object may not be uniformly-colored (which, practically, is quite often the case).

However, despite the drawbacks, this approach enables us to recover the **surface orientation** of an object from a single RGB monocular camera.

A final note: we originally assumed this object was Lambertian, and because of this used the relation that an object's perceived area is its true area scaled by the cosine of the angle between the viewer/light source and the object, Does this apply for real surfaces? No, because many are not ideal Lambertian surfaces. However, in practice, we can just use a **lookup table** that can be calibrated using real images.

4.4 References

1. RANSAC Algorithm: http://www.cse.yorku.ca/ kosta/CompVis_Notes/ransac.pdf
2. Bilinear Interpolation: https://en.wikipedia.org/wiki/Bilinear_interpolation
3. Bicubic Interpolation: https://en.wikipedia.org/wiki/Bicubic_interpolation
4. Multilateration: https://www.icao.int/Meetings/AMC/MA/2007/surv_semi/Day02_SENSIS_Turner.pdf
5. Lambertian Surface: Robot Vision, Page 212
6. Foreshortening: https://en.wikipedia.org/wiki/Perspective

5 Lecture 6: Photometric Stereo, Noise Gain, Error Amplification, Eigenvalues and Eigenvectors Review

This lecture covers an overview of some relevant concepts from linear algebra. If you would like a refresher on some of these concepts, please check out Professor Horn's posted linear algebra review **here**.

5.1 Applications of Linear Algebra to Motion Estimation/Noise Gain

Recall our problem of noise gain, in that if we observe a noisy estimate of y and we're asked to derive x, the noise from our observed value can greatly affect our estimate of x. We've analyzed this problem in 1D, and now we're ready to generalize this to higher dimensions using eigenvalues and eigenvectors.

For the sake of brevity, the notes from the linear algebra review will not be given here, but they are in the handwritten notes for this lecture, as well as in the linear algebra handout posted above. These notes cover:

- Definitions of eigenvalues/eigenvectors
- Characteristic Polynomials
- Rewriting vectors using an eigenbasis

5.1.1 Application Revisited: Photometric Stereo

Let us now return to an application where we determine shape from shading: photometric stereo. Let us first consider the simple case where the surface we seek to reconstruct is Lambertian:

$$E \propto \cos\theta_i = \rho\hat{\mathbf{n}} \cdot \hat{\mathbf{s}}_i$$

Recall the following, to help frame the setup of this problem:

- For shape from shading problems, our goal is to estimate and reconstruct a 3D scene from a 2D image given information about the shading and surface orientation of the scene.
- In 2D, this shape from shading problem has 2 Degrees of Freedom (abbreviated in these course notes as DOF).
- Recall that we can use albedo (ρ) to actually make this problem easier - can create a problem with 3 unknowns and 3 equations. Without the use of albedo, recall that we had a quadratic constraint we had to follow, which, by **Bezout's theorem**, suggests that there are at two solutions to such a system of equations. In this case, we have 3 linear equations and 3 unknowns, so by Bezout's theorem we have only one solution, as desired.

Recall that for such a system, suppose we have three brightness measurements of the form:

1. $E_1 = \rho\hat{\mathbf{n}} \cdot \hat{\mathbf{s}}_1$ - Intuitively, look at pixel with light source located at $\mathbf{s}_1$ and take measurement E_1.
2. $E_2 = \rho\hat{\mathbf{n}} \cdot \hat{\mathbf{s}}_2$ - Intuitively, look at pixel with light source located at $\mathbf{s}_2$ and take measurement E_2.
3. $E_3 = \rho\hat{\mathbf{n}} \cdot \hat{\mathbf{s}}_3$ - Intuitively, look at pixel with light source located at $\mathbf{s}_3$ and take measurement E_3.

Let's review the linear algebraic system of equations by combining the equations above into a matrix-vector product.

$$\begin{bmatrix} -s_1^T \\ -s_2^T \\ -s_3^T \end{bmatrix} \mathbf{n} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}$$

Note the following:

- Since the matrix on the left-hand side corresponds to the three-dimensional position of the light source in the given frame of reference, we have:

$$\begin{bmatrix} -s_1^T \\ -s_2^T \\ -s_3^T \end{bmatrix} \in \mathbb{R}^{3\times 3}, \quad \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \in \mathbb{R}^3$$

- $\mathbf{n} = \rho\hat{\mathbf{n}}$, i.e. we do not need to deal with the second order constraint $\hat{\mathbf{n}}^T\hat{\mathbf{n}} = 1$. This eliminates the second-order constraint from our set of equations and ensures we are able to derive a unique solutions by solving a system of only linear equations.
- Define $\mathbf{S} \triangleq \begin{bmatrix} -s_1^T \\ -s_2^T \\ -s_3^T \end{bmatrix}$

Like many other linear system of equations we encounter of the form $\mathbf{Ax} = \mathbf{b}$, we typically want to solve for $\mathbf{x}$. In this case, we want to solve for $\mathbf{n}$, which provides information about the surface orientation of our object of interest:

$$\mathbf{Sn} = \mathbf{E} \longrightarrow \mathbf{n} = \mathbf{S}^{-1}\mathbf{E}$$

Like the other "inverse" problems we have solved so far in this class, we need to determine when this problem is ill-posed, i.e. when $\mathbf{S}$ is not invertible. Recall from linear algebra that this occurs when the columns of $\mathbf{S}$ are not linearly independent/the rank of $\mathbf{S}$ is not full.

An example of when this problem is ill-posed is when the light source vectors are coplanar with one another, i.e.

$\mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3 = 0$ (Note that this can be verified by simply computing the vector sum of any three coplanar vectors.)

Therefore, for optimal performance and to guarantee that $\mathbf{S}$ is invertible:

- We make the vectors from the light sources to the objects ($\mathbf{s}_i$) as non-coplanar as possible.
- The best configuration for this problem is to have the vectors from the light sources to the surface be orthogonal to one another. Intuitively, consider that this configuration captures the most variation in the angle between a given surface normal and three light sources.

Example: Determining Depth of Craters on the Moon:

As it turns out, we cannot simply image the moon directly, since the plane between the earth/moon and the sun makes it such that all our vectors $\mathbf{s}_i$ will lie in the same plane/are coplanar. Thus, we cannot observe the moon's tomography from the surface of the earth.

5.2 Lambertian Objects and Brightness

Fun Fact: "Lambert's Law" was derived from a monk observing light passing through an oil spot in a piece of paper.

"Lambert's Law":

$$E_i \propto \cos\theta_i = \hat{\mathbf{n}} \cdot \hat{\mathbf{s}}_i$$

To better understand this law, let us talk more about surface orientation.

5.2.1 Surface Orientation

- For an extended surface, we can take small patches of the surface that are "approximately planar", which results in us constraining the surface to be a **2-manifold** [1] (technically, the language here is "locally planar").
- Using these "approximately planar" patches of the surface results in many unit normal vectors $\hat{\mathbf{n}}_i$ pointing in a variety of different directions for different patches in the surface.
- We can also understand surface orientation from a calculus/Taylor Series point of view. Consider $z(x+\delta x, y+\delta y)$, i.e. an estimate for the surface height at (x,y) perturbed slightly by a small value δ. The Taylor Series expansion of this is given by:

$$z(x+\delta x, y+\delta y) = z(x,y) + \delta x\frac{\partial z}{\partial x} + \delta y\frac{\partial z}{\partial y} + \cdots = z(x,y) + p\delta x + q\delta y$$

$$\text{Where } p \triangleq \frac{\partial z}{\partial x}, q \triangleq \frac{\partial z}{\partial y}$$

- The surface orientation ($p \triangleq \frac{\partial z}{\partial x}, q \triangleq \frac{\partial z}{\partial y}$) captures the gradient of the height of the surface z.
- Note that the surface unit normal $\hat{\mathbf{n}}$ is perpendicular to any two lines in the surface, e.g. tangent lines.

We can use the cross product of the two following tangent lines of this surface to compute the the unit surface normal, which describes the orientation of the surface:

1. $(\delta x, 0, p\delta x)^T = \delta x(1,0,p)^T$
2. $(0, \delta y, q\delta x)^T = \delta y(0,1,q)^T$

Since we seek to find the *unit* surface normal, we can remove the scaling factors out front (δx and δy). Then the cross product of these two vectors is:

$$(1,0,p)^T \times (0,1,p)^T = \det\begin{bmatrix}\hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ 1 & 0 & p \\ 0 & 1 & q\end{bmatrix} = \mathbf{n} = \begin{bmatrix}-p \\ -q \\ 1\end{bmatrix}$$

We can now compute the unit surface normal by normalizing the vector $\mathbf{n}$:

$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{||\mathbf{n}||_2} = \frac{\begin{bmatrix}-p & -q & 1\end{bmatrix}^T}{\sqrt{p^2+q^2+1}}$$

With this surface normal computed, we can also go the opposite way to extract the individual components of the surface orientation (p and q):

$$p = \frac{-\mathbf{n}\cdot\hat{\mathbf{x}}}{\mathbf{n}\cdot\hat{\mathbf{z}}},\quad q = \frac{-\mathbf{n}\cdot\hat{\mathbf{y}}}{\mathbf{n}\cdot\hat{\mathbf{z}}}$$

Note: Do the above equations look similar to something we have encountered before? Do they look at all like the coordinate-wise perspective projection equations?

We will leverage these (p,q)-space plots a lot, using *reflectance maps* (which we will discuss later). Now, instead of plotting in velocity space, we are plotting in surface orientation/spatial derivative space. Individual points in this plane correspond to different surface orientations of the surface.

5.2.2 Surface Orientation Isophotes/Reflectance Maps for Lambertian Surfaces

Recall the equation to derive isophotes of a Lambertian surface (since the brightness depends on the incident angle between the surface and the light source):

$$\hat{\mathbf{n}}\cdot\hat{\mathbf{s}} = E_1 \text{ Where } E_1 \text{ is a constant, scalar level of brightness}$$

If we expand our light source vector $\hat{\mathbf{s}}$, we get the following for this dot product:

$$\hat{\mathbf{n}}\cdot\hat{\mathbf{s}} = \frac{\begin{bmatrix}-p & -q & 1\end{bmatrix}^T}{\sqrt{p^2+q^2+1}}\cdot\frac{\begin{bmatrix}-p_s & -q_s & 1\end{bmatrix}^T}{\sqrt{p_s^2+q_s^2+1}} = E_1$$

Carrying out this dot product and squaring each side, we can derive a parametric form of these isophotes in (p, q) space (note that the light source orientation (p_s, q_s) is fixed (at least for a single measurement), and thus we can treat it as constant:

$$(1 + p_s p + q_s q)^2 = c^2(1 + p^2 + q^2)(1 + p_s^2 + q_s^2)$$

If we plot these isophotes in (p, q) space, we will see they become **conic sections**. Different isophotes will generate different curves. When we have multiple light sources, we can have intersections between these isophotes, which indicate solutions.

5.3 References

[1] Manifolds, https://en.wikipedia.org/wiki/Manifold.

6 Lecture 7: Gradient Space, Reflectance Map, Image Irradiance Equation, Gnomonic Projection

This lecture continues our treatment of the photometric stereo problem, and how we can estimate motion graphically by finding intersections of curves in (p, q) (spatial derivative) space. Following this, we discuss concepts from photometry and introduce the use of lenses.

6.1 Surface Orientation Estimation (Cont.) & Reflectance Maps

Let's review a couple of concepts from the previous lecture:

- Recall our spatial gradient space given by $(p, q) = (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y})$.
- We also discussed **Lambertian surfaces**, or surfaces in which the viewing angle doesn't change brightness, only the angle between the light source and the object affects the brightness. Additionally, the relation between the source/object angle is proportional to: $E \propto \cos\theta_i$, where θ_i is the angle between the light source and the object.
- Recall our derivation of the isophotes of Lambertian surfaces:

  $$\hat{\mathbf{n}} \cdot \hat{\mathbf{s}} = \frac{\begin{bmatrix} -p & -q & 1 \end{bmatrix}^T}{\sqrt{p^2 + q^2 + 1}} \cdot \frac{\begin{bmatrix} -p_s & -q_s & 1 \end{bmatrix}^T}{\sqrt{p_s^2 + q_s^2 + 1}} = c$$

  Where c is a constant brightness level. Carrying out this dot product:

  $$\frac{1 + p_s p + q_s q}{\sqrt{p^2 + q^2 + 1}\sqrt{p_s^2 + q_s^2 + 1}} = c$$

  We can generate plots of these isophotes in surface orientation/(p, q) space. This is helpful in computer graphics, for instance, because it allows us to find not only surface heights, but also the local spatial orientation of the surface. In practice, we can use *lookup tables* for these *reflectance maps* to find different orientations based off of a measured brightness.

6.1.1 Forward and Inverse Problems

As discussed previously, there are two problems of interest that relate surface orientation to brightness measurements:

1. **Forward Problem**: $(p, q) \rightarrow E$ (Brightness from surface orientation)
2. **Inverse Problem**: $E \rightarrow (p, q)$ (Surface orientation from brightness)

In most settings, the second problem is of stronger interest, namely because it is oftentimes much easier to measure brightness than to directly measure surface orientation. To measure the surface orientation, we need many pieces of information about the local geometries of the solid, at perhaps many scales, and in practice this is hard to implement. But it is straightforward to simply find different ways to illuminate the object!

A few notes about solving this inverse problem:

- We can use the reflectance maps to find intersections (either curves or points) between isophotes from different light source orientations. The orientations are therefore defined by the interesections of these isophote curves.
- Intuitively, the reflectance map answers the question "How bright will a given orientation be?"

- We can use an automated lookup table to help us solve this inverse problem, but depending on the discretization level of the reflectance map, this can become tedious. Instead, wwe can use a calibration object, such as a sphere, instead.
- For a given pixel, we can take pictures with different lighting conditions and repeatedly take images to infer the surface orientation from the sets of brightness measurements.
- This problem is known as **photometric stereo**: Photometric stereo is a technique in computer vision for estimating the surface normals of objects by observing that object under different lighting conditions [1].

6.1.2 Reflectance Map Example: Determining the Surface Normals of a Sphere

A few terms to set before we solve for p and q:

- Center of the sphere being imaged: $(x_0, y_0, z_0)^T$
- Location of the light source: $(x, y, z)^T$

Then:

- $z - z_0 = \sqrt{r_0^2 - (x - x_0)^2 + (y - y_0)^2}$
- $p = \frac{x - x_0}{z - z_0}$
- $a = \frac{y - y_0}{z - z_0}$

(Do the above equations carry a form similar to that of perspective projection?)

6.1.3 Computational Photometric Stereo

Though the intersection of curves approach we used above can be used for finding solutions to surfaces with well-defined reflectance maps, in general we seek to leverage approaches that are as general and robust as possible. We can accomplish this, for instance, by using a 3D reflectance map in (E_1, E_2, E_3) space, where each E_i is a brightness measurement. Each discretized (unit of graphic information that defines a point in three-dimensional space [2]) contains a specific surface orientation parameterized by (p, q). This approach allows us to obtain surface orientation estimates by using brightness measurements along with a **lookup table** (what is described above).

While this approach has many advantages, it also has some drawbacks that we need to be mindful of:

- Discretization levels often need to be made coarse, since the number of (p, q) voxel cells scales as a cubic in 3D, i.e. $f(d) \in O_{\text{space}}(d^3)$, where d is the discretization level.
- Not all voxels may be filled in - i.e. mapping (let us denote this as ϕ) from 3D to 2D means that some kind of surface will be created, and therefore the (E_1, E_2, E_3) space may be sparse.

One way we can solve this latter problem is to reincorporate albedo ρ! Recall that we leveraged albedo to transform a system with a quadratic constraint into a system of all linear equations. Now, we have a mapping from 3D space to 3D space:

$$\phi : (E_1, E_2, E_3) \longrightarrow (p, q, \rho) \text{ i.e. } \phi : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$$

It is worth noting, however, that even with this approach, that there will still exist brightness triples (E_1, E_2, E_3) that do not map to any (p, q, ρ) triples. This could happen, for instance, when we have extreme brightness from measurement noise.

Alternatively, an easy way to ensure that this map ϕ is injective is to use the reverse map (we can denote this by ψ):

$$\psi : (p, q, \rho) \longrightarrow (E_1, E_2, E_3) \text{ i.e. } \psi : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$$

6.2 Photometry & Radiometry

Next, we will rigorously cover some concepts in photometry and radiommetry. Let us begin by rigorously defining these two fields:

- **Photometry**: The science of the measurement of light, in terms of its perceived brightness to the human eye [3].
- **Radiometry**: The science of measurement of radiant energy (including light) in terms of absolute power [3].

Next, we will cover some concepts that form the cornerstone of these fields:

- **Irradiance**: This quantity concerns with power emitted/reflected by the object when light from a light source is incident upon it. Note that for many objects/mediums, the radiation reflection is not **isotropic** - i.e. it is not emitted equally in different directions.

  Irradiance is given by: $E = \frac{\delta P}{\delta A}$ (Power/unit area, W/m^2)

- **Intensity**: Intensity can be thought of as the power per unit angle. In this case, we make use of **solid angles**, which have units of Steradians. Solid angles are a measure of the amount of the field of view from some particular point that a given object covers [4]. We can compute a solid angle Ω using: $\Omega = \frac{A_{\text{surface}}}{R^2}$.

  Intensity is given by: Intensity $= \frac{\delta P}{\delta \Omega}$.

  Radiance: Oftentimes, we are only interested in the power reflected from an object that actually reaches the light sensor. For this, we have **radiance**, which we contrast with irradiance defined above. We can use radiance and irradiance to contrast the "brightness" of a scene and the "brightness" of the scene perceived in the image plane.

  Radiance is given by: $L = \frac{\delta^2 P}{\delta A \delta \Omega}$

These quantities provide motivation for our next section: lenses. Until now, we have considered only pinhole cameras, which are infinitisimally small, but to achieve nonzero irradiance in the image plane, we need to have lenses of finite size.

6.3 Lenses

Lenses are used to achieve the same photometric properties of perspective projection as pinhole cameras, but do so using a finite "pinhole" size and thus by encountering a finite number of photons from light sources in scenes. One caveat with lenses that we will cover in greater detail: lenses only work for a finite focal length f. From Gauss, we have that it is impossible to make a perfect lens, but we can come close.

6.3.1 Thin Lenses - Introduction and Rules

Thin lenses are the first type of lens we consider. These are often made from glass spheres, and obey the following three rules:

- Central rays (rays that pass through the center of the lens) are undeflected - this allows us to preserve perspective projection as we had for pinhole cameras.
- The ray from the focal center emerges parallel to the optical axis.
- Any parallel rays go through the focal center.

Thin lenses also obey the following law, which relates the focal centers (a and b) to the focal length of the lens.

$$\frac{1}{a} + \frac{1}{b} = \frac{1}{f_0}$$

Thin lenses can also be thought of as analog computers, in that they enforce the aforementioned rules for all light rays, much like a computer executes a specific program with outlined rules and conditions. Here are some tradeoffs/issues to be aware of for thin lenses:

- **Radial distortion**: In order to bring the entire angle into an image (e.g. for wide-angle lenses), we have the "squash" the edges of the solid angle, thus leading to distortion that is radially-dependent. Typically, other lens defects are mitigated at the cost of increased radial distortion. Some specific kinds of radial distortion [5]:
  - Barrel distortion
  - Mustache distortion
  - Pincushion distortion
- **Lens Defects**: These occur frequently when manufacturing lenses, and can originate from a multitude of different issues.

6.3.2 Putting it All Together: Image Irradiance from Object Irradiance

Here we show how we can relate object irradiance (amount of radiation emitted from an object) to image plane irradiance of that object (the amount of radiation emitted from an object that is incident on an image plane). To understand how these two quantities relate to each other, it is best to do so by thinking of this as a ratio of "Power in / Power out". We can compute this ratio by matching solid angles:

$$\frac{\delta I \cos\alpha}{(f\sec\alpha)^2} = \frac{\delta O\cos\theta}{(z\sec\alpha)^2} \implies \frac{\delta I\cos\alpha}{f^2} = \frac{\delta O}{z^2} \implies \frac{\delta O}{\delta I} = \frac{\cos\alpha}{\cos\theta}\Big(\frac{z}{f}\Big)^2$$

The equality on the far right-hand side can be interpreted as a unit ratio of "power out" to "power in". Next, we'll compute the subtended solid angle Ω:

$$\Omega = \frac{\frac{\pi}{4}d^2\cos\alpha}{(z\sec\alpha)^2} = \frac{\pi}{4}\Big(\frac{d}{z}\Big)^2\cos^3\alpha$$

Next, we will look at a unit change of power (δP):

$$\delta P = L\delta O\Omega\cos\theta = L\delta O\frac{\pi}{4}\Big(\frac{d}{z})^2\cos^3\alpha\cos\theta$$

We can relate this quantity to the irradiance brightness in the image:

$$\begin{aligned}E = \frac{\delta P}{\delta I} &= L\frac{\delta O}{\delta I}\frac{\pi}{4}\Big(\frac{d}{z}\Big)^2\cos^3\alpha\cos\theta \\ &= \frac{\pi}{4}L\Big(\frac{d}{f}\Big)^2\cos^4\alpha\end{aligned}$$

A few notes about this above expression:

- The quantity on the left, E is the **irradiance brightness in the image**.
- The quantity L on the righthand side is the **radiance brightness of the world**.
- The ratio $\frac{d}{f}$ is the inverse of the so-called "f-stop", which describes how open the aperture is. Since these terms are squared, they typically come in multiples of $\sqrt{2}$.
- The reason why we can be sloppy about the word brightness (i.e. **radiance** vs. **irradiance**) is because the two quantities are proportional to one another: $E \propto L$.
- When does the $\cos^4\alpha$ term matter in the above equation? This term becomes non-negligible when we go **off-axis** from the optical axis - e.g. when we have a wide-angle axis. Part of the magic of DSLR lenses is to compensate for this effect.
- Radiance in the world is determined by **illumination, orientation,** and **reflectance**.

6.3.3 BRDF: Bidirectional Reflectance Distribution Function

The BRDF can be interpreted intuitively as a ratio of "Energy In" divided by "Energy Out" for a specific slice of **incident** and **emitted** angles. It depends on:

- Incident and emitted azimuth angles, given by ϕ_i, ϕ_e, respectively.
- Incident and emitted colatitude/polar angles, given by θ_i, θ_e, respectively.

Mathematically, the BRDF is given by:

$$f(\theta_i, \theta_e, \phi_i, \phi_e) = \frac{\delta L(\theta_e, \phi_e)}{\delta E(\theta_i, \phi_i)} = \frac{\text{“Energy In”}}{\text{“Energy Out”}}$$

Let's think about this ratio for a second. Please take note of the following as a mnemonic to remember which angles go where:

- The emitted angles (between the object and the camera/viewer) are parameters for radiance (L), which makes sense because radiance is measured in the image plane.
- The incident angles (between the light source and the object) are parameters for irradiance (E), which makes sense because irradiance is measured in the scene with the object.

Practically, how do we measure this?

- Put light source and camera in different positions
- Note that we are exploring a 4D space comprised of $[\theta_i, \theta_e, \phi_i, \phi_e]$, so search/discretization routines/sub-routines will be computationally-expensive.
- This procedure is carried out via **goniometers** (angle measurement devices). Automation can be carried out by providing these devices with mechanical trajectories and the ability to measure brightness. But we can also use multiple light sources!
- Is the space of BRDF always in 4D? For most surfaces, what really matters are the difference in emitted and incident azimuth angles $(\phi_e - \phi_i)$. This is applicable for materials in which you can rotate the surface in the plane normal to the surface normal without changing the surface normal.
- But this aforementioned dimensionality reduction procedure cannot always be done - materials for which there is directionally-dependent microstructures, e.g. hummingbird's neck, semi-precious stones, etc.

6.3.4 Helmholtz Reciprocity

Somewhat formally, the **Helmholtz reciprocity** principle describes how a ray of light and its reverse ray encounter matched optical adventures, such as reflections, refractions, and absorptions in a passive medium, or at an interface [6]. It is a property of BRDF functions which mathematically states the following:

$$f(\theta_i, \theta_e, \phi_i, \phi_e) = f(\theta_e, \theta_i, \phi_e, \phi_i) \; \forall \; \theta_i, \theta_e, \phi_i, \phi_e$$

In other words, if the ray were reversed, and the incident and emitted angles were consequently switched, we would obtain the same BRDF value. A few concluding notes/comments on this topic:

- If there is not Helmholtz reciprocity/symmetry between a ray and its inverse in a BRDF, then there must be energy transfer. This is consistent with the 2nd Law of Thermodynamics.
- Helmholtz reciprocity has good computational implications - since you have symmetry in your (possibly) 4D table across incident and emitted angles, you only need to collect and populate half of the entries in this table. Effectively, this is a tensor that is symmetric across some of its axes.

6.4 References

1. Photometric Stereo, https://en.wikipedia.org/wiki/Photometric_stereo
2. Voxels, https://whatis.techtarget.com/definition/voxel
3. Photometry, https://en.wikipedia.org/wiki/Photometry_(optics)
4. Solid Angles, https://en.wikipedia.org/wiki/Solid_angle
5. Distortion, https://en.wikipedia.org/wiki/Distortion_(optics)
6. Helmholtz Reciprocity, https://en.wikipedia.org/wiki/Helmholtz_reciprocity

7 Lecture 8: Shape from Shading, Special Cases, Lunar Surface, Scanning Electron Microscope, Green's Theorem in Photometric Stereo

In this lecture, we will continue our discussion of photometry/radiometry, beginning with the concepts covered last lecture. We will then explore Hapke surfaces and how they compare to Lambertian surfaces, and show how they can be used to solve our shape from shading problem. Finally, we will introduce two new types of lenses and some of their applications: “thick” lenses and telecentric lenses.

7.1 Review of Photometric and Radiometric Concepts

Let us begin by reviewing a few key concepts from photometry and radiometry:

- **Photometry**: Photometry is the science of measuring visible radiation, light, in units that are weighted according to the sensitivity of the human eye. It is a quantitative science based on a statistical model of the human visual perception of light (eye sensitivity curve) under carefully controlled conditions [3].
- **Radiometry**: Radiometry is the science of measuring radiation energy in any portion of the electromagnetic spectrum. In practice, the term is usually limited to the measurement of ultraviolet (UV), visible (VIS), and infrared (IR) radiation using optical instruments [3].
- **Irradiance**: $E \triangleq \frac{\delta P}{\delta A}$ (W/m^2). This corresponds to light falling on a surface. When imaging an object, irradiance is converted to a grey level.
- **Intensity**: $I \triangleq \frac{\delta P}{\delta W}$ (W/ster). This quantity applied to a point source and is often directionally-dependent.
- **Radiance**: $L \triangleq \frac{\delta^2 P}{\delta A \delta \Omega}$ (W/m$^2\times$ ster). This photometric quantity is a measure of how bright a surface appears in an image.
- **BRDF (Bidirectional Reflectance Distribution)**: $f(\theta_i, \theta_e, \phi_i, \phi_e) = \frac{\delta L(\theta_e, \phi_e)}{\delta E(\theta_i, \phi_i)}$. This function captures the fact that oftentimes, we are only interested in light hitting the camera, as opposed to the total amount of light emitted from an object. Last time, we had the following equation to relate image irradiance with object/surface radiance:

  $$E = \frac{\pi}{4} L \Big(\frac{d}{f}\Big)^2 \cos^4 \alpha$$

  Where the irradiance of the image E is on the lefthand side and the radiance of the object/scene L is on the right. The BRDF must also satisfy **Helmholtz reciprocity**, otherwise we would be violating the 2nd Law of Thermodynamics. Mathematically, recall that Helmholtz reciprocity is given by:

  $$f(\theta_i, \theta_e, \phi_i, \phi_e) = f(\theta_e, \theta_i, \phi_e, \phi_i) \; \forall \; \theta_i, \theta_e, \phi_i, \phi_e$$

With this, we are now ready to discuss our first type of surfaces: ideal Lambertian surfaces.

7.2 Ideal Lambertian Surfaces

Ideal Lambertian surfaces are of interest because their sensed brightness (irradiance) only depends on the cosine of the incident angles (θ_i, ϕ_i). A few additional notes about these surfaces:

- Ideal Lambertian surfaces are equally bright from all directions, i.e.

  $$\begin{gathered} f(\theta_i, \theta_e, \phi_i, \phi_e) = f(\theta_e, \theta_i, \phi_e, \phi_i) \; \forall \; \theta_i, \theta_e, \phi_i, \phi_e \\ \text{AND} \\ f(\theta_i, \theta_e, \phi_i, \phi_e) = K \in \mathbb{R} \text{ with respect to } \theta_e, \phi_e, \text{ i.e.} \frac{\partial f(\theta_i, \theta_e, \phi_i, \phi_e)}{\partial \theta_e} = \frac{\partial f(\theta_i, \theta_e, \phi_i, \phi_e)}{\partial \phi_e} = 0 \end{gathered}$$

- If the surface is ideal, the Lambertian surface reflects all incident light. We can use this to compute f. Suppose we take a small slice of the light reflected by a Lambertian surface, i.e. a hemisphere for all positive z. The area of this surface is given by: $\sin \theta_e \delta \theta_e \delta \phi_e$, which is the determinant of the coordinate transformation from euclidean to spherical coordinates.

Then, we have that:

$$\begin{aligned}
\int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}}(fE\cos\theta_i\sin\theta_e d\theta_e)d\phi_e &= E\cos\theta_i \;\; \textbf{(Integrate all reflected light)} \\
E\cos\theta_i\int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}}(f\sin\theta_e d\theta_e)d\phi_e &= E\cos\theta_i \;\; \textbf{(Factor out } E\cos\theta_i\textbf{)} \\
\int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}}(f\sin\theta_e d\theta_e)d\phi_e &= 1 \;\; \textbf{(Cancel out on both sides)} \\
2\pi\int_{0}^{\frac{\pi}{2}} f\sin\theta_e\cos\theta_e d\theta_e &= 1 \;\; \textbf{(No dependence on } \phi\textbf{)} \\
\pi f\int_{0}^{\frac{\pi}{2}} 2\sin\theta_e\cos\theta_e d\theta_e &= 1 \;\; \textbf{(Rearranging terms)} \\
\pi f\int_{0}^{\frac{\pi}{2}} \sin(2\theta_e) d\theta_e &= 1 \;\; \textbf{(Using identity } 2\sin\theta\cos\theta = \sin(2\theta)\textbf{)} \\
\pi f[-\frac{1}{2}\cos(2\theta_e)]_{\theta_e=0}^{\theta_e=\frac{\pi}{2}} &= 1 \\
\pi f = 1 \implies f &= \frac{1}{\pi}
\end{aligned}$$

7.2.1 Foreshortening Effects in Lambertian Surfaces

Now let us discuss a key issue in radiation reflectance and machine vision - **foreshortening**. Foreshortening is the visual effect or optical illusion that causes an object or distance to appear shorter than it actually is because it is angled toward the viewer [1]. Let us show how this is relevant for computing f.

Suppose we have a point source radiating isotropically over the positive hemisphere in a 3-dimensional spherical coordinate system. Then, the solid angle spanned by this hemisphere is:

$$\Omega = \frac{A_{\text{surface}}}{r^2} = \frac{2\pi r^2}{r^2} = 2\pi \text{ Steradians}$$

If the point source is radiating isotropically in all directions, then $f = \frac{1}{2\pi}$. But we saw above that $f = \frac{1}{\pi}$ for an ideal Lambertian surface. Why do we have this discrepancy? Even if brightness is the same/radiation emitted is the same in all directions, this does not mean that the power radiated in all directions is hte same. This is due to **foreshortening**, since the angle between the object's surface normal and the camera/viewer changes.

7.2.2 Example: Distant Lambertian Point Source

Here, we will have that $L = \frac{1}{f}Es$ (where s is the location of the light source). Since we have to take foreshortening into account, the perceived area A of an object with world size A' is given by:

$$A = A'\cos\theta_i$$

Therefore, our expression for the the radiance (how bright the object appears in the image) is given by:

$$L = \frac{1}{\pi}Es = \frac{1}{\pi}E_0\cos\theta_i$$

7.3 Hapke Surfaces

Hapke surfaces are another type of surface we study in this course. Formally, a Hapke surface is an object which has reflectance properties corresponding to **Hapke parameters**, which in turn are a set of parameters for an empirical model that is commonly used to describe the directional reflectance properties of the airless regolith surfaces of bodies in the solar system [2].

The BRDF of a Hapke surface is given by:

$$f(\theta_i, \phi_i, \theta_e, \phi_e) = \frac{1}{\sqrt{\cos\theta_e\cos\theta_i}}$$

We can see that the Hapke BRDF also satisfies Helmholtz Reciprocity:

$$f(\theta_i, \phi_i, \theta_e, \phi_e) = \frac{1}{\sqrt{\cos\theta_e \cos\theta_i}} = \frac{1}{\sqrt{\cos\theta_i \cos\theta_e}} = f(\theta_e, \phi_e, \theta_i, \phi_i)$$

Using the Hapke BRDF, we can use this to compute **illumination** (same as **radiance**, which is given by:

$$\begin{aligned} L &= E_0 \cos\theta_i f(\theta_i, \phi_i, \theta_e, \phi_e) \\ &= E_0 \cos\theta_i \frac{1}{\sqrt{\cos\theta_e \cos\theta_i}} \\ &= E_0 \sqrt{\frac{\cos\theta_i}{\cos\theta_e}} \end{aligned}$$

Where L is the radiance/illumination of the object in the image, $E_0 \cos\theta_i$ is the irradiance of the object, accounting for foreshortening effects, and $\frac{1}{\sqrt{\cos\theta_i \cos\theta_e}}$ is the BRDF of our Hapke surface.

7.3.1 Example Application: Imaging the Lunar Surface

What do the isophotes of the moon look like, supposing the moon can fall under some of the different types of surfaces we have discussed?

- **Lambertian**: We will see circles and ellipses of isophotes, depending on the angle made between the viewer and the the moon.
- **Hapke**: Because of the BRDF behavior, isophotes will run **longitudinally** along the moon in the case in which it is a Hapke surface. Recall isophotes are where the brightness of the object in the image, given by:

  $$\sqrt{\frac{\cos\theta_i}{\cos\theta_e}} = \sqrt{\frac{\frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{||\hat{\mathbf{n}}||_2||\hat{\mathbf{s}}||_2}}{\frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}}{||\hat{\mathbf{n}}||_2||\hat{\mathbf{v}}||_2}}} = c \implies \frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}} = c^2 \text{ for some } c \in \mathbb{R}$$

  Where $\hat{\mathbf{n}}$ is the surface normal vector of the object being imaged, $\hat{\mathbf{s}}$ is the unit vector describing the position of the light source, and $\hat{\mathbf{v}}$ is the unit vector describing the position of the vertical. We can derive a relationship between $\hat{\mathbf{n}}$ and $\hat{\mathbf{s}}$:

  $$\hat{\mathbf{n}}\cdot\hat{\mathbf{s}} = c\hat{\mathbf{n}}\cdot\hat{\mathbf{v}} \implies \hat{\mathbf{n}}\cdot(\hat{\mathbf{s}} - c\hat{\mathbf{v}}) = 0 \implies \hat{\mathbf{n}} \perp (\hat{\mathbf{s}} - c\hat{\mathbf{v}})$$

  Next, we will use spherical coordinates to show how the isophotes of this surface will be longitudinal:

  $$\hat{\mathbf{n}} = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix} \in \mathbb{R}^3$$

  Since normal vectors $\hat{\mathbf{n}} \in \mathbb{R}^3$ only have two degrees of freedom (since $||\hat{\mathbf{n}}||_2 = 1$), we can fully parameterize these two degrees of freedom by the **azimuth** and **colatitude/polar** angles ϕ and θ, respectively. Then we can express our coordinate system given by the orthogonal basis vectors $\hat{\mathbf{n}}, \hat{\mathbf{s}}, \hat{\mathbf{v}}$ as:

  $$\begin{aligned} \hat{\mathbf{n}} &= \begin{bmatrix} \sin\theta\cos\phi & \sin\theta\sin\phi & \cos\theta \end{bmatrix}^T \\ \hat{\mathbf{s}} &= \begin{bmatrix} \cos\theta_s & \sin\theta_s & 0 \end{bmatrix}^T \\ \hat{\mathbf{v}} &= \begin{bmatrix} \cos\theta_v & \sin\theta_v & 0 \end{bmatrix}^T \end{aligned}$$

  We can use our orthogonal derivations from above to now show our longitudinal isophotes:

  $$\sqrt{\frac{\cos\theta_i}{\cos\theta_e}} = \frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}} = \frac{\sin\theta\cos\phi\cos\phi_s + \sin\theta\sin\phi\sin\phi_s}{\sin\theta\cos\phi\cos\phi_v + \sin\theta\sin\phi\sin\phi_v} = \frac{\cos(\phi - \phi_s)}{\cos(\phi - \phi_v)} = c \text{ for some } c \in \mathbb{R}$$

  From the righthand side above, we can deduce that isophotes for Hapke surfaces correspond to points with the same azimuth having the same brightness, i.e. the isophotes of the imaged object (such as the moon) are along lines of constant longitude.

7.3.2 Surface Orientation and Reflectance Maps of Hapke Surfaces

Next, we will take another look at our "shape from shading" problem, but this time using the surface normals of Hapke surfaces and relating this back to reflectance maps and our (p, q) space. From our previous derivations, we already have that the BRDF of a Hapke surface is nothing more than the square root of the dot product of the surface normal with the light source vector divided by the dot product of the surface normal with the vertical vector, i.e.

$$\sqrt{\frac{\cos\theta_i}{\cos\theta_e}} = \sqrt{\frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}}}$$

Next, if we rewrite our unit vectors as 3D vectors in (p, q) space, we have:

$$\begin{aligned}
\hat{\mathbf{n}} &= \frac{\begin{bmatrix}-p & -q & 1\end{bmatrix}^T}{\sqrt{p^2+q^2+1}} \\
\hat{\mathbf{s}} &= \frac{\begin{bmatrix}-p_s & -q_s & 1\end{bmatrix}^T}{\sqrt{p_s^2+q_s^2+1}} \\
\hat{\mathbf{v}} &= \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T
\end{aligned}$$

Then, substituting these definitions, we have:

$$\begin{aligned}
\hat{\mathbf{n}}\cdot\hat{\mathbf{v}} &= \frac{1}{\sqrt{1+p^2+q^2}} \\
\hat{\mathbf{n}}\cdot\hat{\mathbf{s}} &= \frac{1+p_s p+q_s q}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}} \\
\frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}} &= \frac{\frac{1+p_s p+q_s q}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}}}{\frac{1}{\sqrt{1+p^2+q^2}}} \\
&= \frac{1+p_s p+q_s q}{\sqrt{1+p_s^2+q_s^2}} \\
&= \frac{1+p_s P+q_s q}{R_s}
\end{aligned}$$

(Where $R_s \triangleq \sqrt{1+p_s^2+q_s^2}$.) Note that p_s and q_s are constants, since they only reflect the spatial derivatives of the light source. This ratio of dot products is therefore linear in (p, q) space! Then, what do the isophotes look like in gradient space? Recall that isophotes take the form:

$$\sqrt{\frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}}} = c \implies \frac{\hat{\mathbf{n}}\cdot\hat{\mathbf{s}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{v}}} = \frac{1+p_s P+q_s q}{R_s} = c^2 \text{ for some } c \in \mathbb{R}\ .$$

Therefore, isophotes of Hapke surfaces are **linear** in spatial gradient space! This is very useful for building reflectance maps because, as we will see shortly, a simple change of coordinate system in (p, q) space tells us a lot about the surface orientation in one of the two directions.

For one thing, let us use our linear isophotes in gradient space to solve our photometric stereo problem, in this case with two measurements under different lighting conditions. Photometric stereo is substantially easier with Hapke surfaces than with Lambertian, because there is no ambiguity in where the solutions lie. Unlike Lambertian surfaces, because of the linearity in (p, q) space we are guaranteed by Bezout's Theorem to have only one unique solution.

7.3.3 Rotated (p', q') Coordinate Systems for Brightness Measurements

We can also derive/obtain a surface orientation in gradient space in a rotated coordinate system in gradient space, i.e. from $(p, q) \longrightarrow (p', q')$.

We can prove (see the synchronous lecture notes for this part of the course) that the transforming the points in (x, y) via a rotation matrix $\mathbf{R}$ is equivalent to rotating the gradient space (p,q) by the same matrix $\mathbf{R}$. I.e.

$$\mathbf{R} : (x,y) \implies (x',y'), \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{R} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$\mathbf{R} : (p,q) \implies (p',q'), \begin{bmatrix} p' \\ q' \end{bmatrix} = \mathbf{R} \begin{bmatrix} p \\ q \end{bmatrix}$$

Where $\mathbf{R}$ is given according to:

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \in SO(2)$$

(Where $SO(2)$ is the Special Orthogonal Group [4].) Note that $\mathbf{R}^{-1} = \mathbf{R}^T$ since $\mathbf{R}$ is orthogonal and symmetric.

By rotating our coordinate system from $(p,q) \longrightarrow (p',q')$, we are able to uniquely specify p', but not q', since our isophotes lie along a multitude/many q' values. Note that this rotation system is specified by having p' lying along the gradient of our isophotes.

Returning to our Lunar Surface application with Hapke surfaces, we can use this surface orientation estimation framework to take an iterative, incremental approach to get a profile of the lunar surface. This enables us to do **depth profile shaping** of the moon simply from brightness estimates! There are a few caveats to be mindful of for this methodology, however:

- We can only recover absolute depth values provided we are given initial conditions (this makes sense, since we are effectively solving a differential equation in estimating the depth z from the surface gradient $(p,q)^T$). Even without initial conditions, however, we can still recover the general shape of profiling, simply without the absolute depth.
- Additionally, we can get profiles for each cross-section of the moon/object we are imaging, but it is important to recognize that these profiles are effectively independent of one another, i.e. we do not recover any information about the relative surface orientation changes between/at the interfaces of different profiles we image. We can use **heuristic-based** approaches to get a **topographic mapping** estimate. We can then combine these stitched profiles together into a 3D surface.

7.4 "Thick" & Telecentric Lenses

We will now shift gears somewhat to revisit lenses, namely "thick" and telecentric lenses.

7.4.1 "Thick" Lenses

In practice, lenses are not thick, and are typically cascaded together to mitigate the specific aberrations and wavelength dependence of each individual lens. For thick lenses, we will leverage the following definitions:

- **Nodal/principal points**: The center points of the outermost (ending and beginning lenses).
- **Principal planes**: The distance from the object to the first imaging lens and the distance from the last imaging lens to the viewer or imaging plane. Abbreviated by b.
- **Focal length**: This definition is the same as before, but in this case, the focal length is the distance from the first imaging lens to the last imaging lens along the optical axis (i.e. orthogonal to the radial unit vectors of each lens in the "thick" lens).

From this, we have the formula:

$$\frac{1}{a} + \frac{1}{b} = \frac{1}{f}$$

One "trick" we can do with this is achieve a long focal length f and a small field of view (FOV), i.e. as is done in a telephoto lens.

7.4.2 Telecentric Lenses

These types of lenses are frequently used in machine vision, such as for circuit board inspection or barcode reading.

Motivation for these types of lenses: When we have perspective projection, the degree of **magnification** depends on **distance**. How can we remove this unwanted distance-dependent magnification due to perspective projection? We can do so by effectively "moving" the Center of Projection (COP) to "$-\infty$". This requires using a lens with a lot of glass.

A few notes about this:

- To image with magnifying effects, the lens must be at least as wide in diameter as the width of the object.
- a A double telecentric lens can be created by moving the other "modal point" (the imaging plane/viewer) to "$+\infty$". In this case, our magnifying term given by $\cos^4 \alpha$ goes to zero, because:

$$\lim_{\text{COP} \to +\infty} \alpha = 0$$

- We can generalize this idea even more with "lenselets": These each concentrate light into a confined area, and can be used in an array fashion as we would with arrays of photodiodes. A useful term for these types of arrays is **fill factor**: The fill factor of an image sensor array is the ratio of a pixel's light sensitive area to its total area [5]. Lenselet arrays can be useful because it helps us to avoid **aliasing effects**. We need to make sure that if we are sampling an object discretely, that we do not have extremely high-frequency signals without some low pass filtering. This sort of low-pass filtering can be achieved with using large pixels and averaging (a crude form of lowpass filtering). However, this averaging scheme does not work well when light comes in at off-90 degree angles. We want light to come into the sensors at near-90 degrees (DSLRs find a way to get a around this).
- Recall that for an object space telecentric device, we no longer have a distance-based dependence. Effectively, we are taking **perspective projection** and making the **focal length** larger, resulting in **approximately orthographic projection**. Recall that for orthographic projection, projection becomes nearly independent in position, i.e. (x, y) in image space has a linear relationship with (X, Y) in the world. This means that we can measure sizes of objects independently of how far away they are!

  Starting with perspective projection:

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

$$\text{Having } |\Delta Z| << |Z|, \text{ where} \Delta Z \text{ is the variation in Z of an object } \implies x = \frac{f}{Z_0}X, y = \frac{f}{Z_0}Y$$

$$\text{Approximating } \frac{f}{Z_0} = 1 \implies x = X, y = Y$$

7.5 References

1. Foreshortening, https://en.wikipedia.org/wiki/Perspective˙(graphical)
2. Hapke Parameters, https://en.wikipedia.org/wiki/Hapke_parameters
3. Understanding Radiance (Brightness), Irradiance and Radiant Flux
4. Orthogonal Group, https://en.wikipedia.org/wiki/Orthogonal group
5. https://en.wikipedia.org/wiki/Fill factor (image sensor)

8 Lecture 9: Shape from Shading, General Case - From First Order Nonlinear PDE to Five ODEs

In this lecture, we will begin by exploring some applications of magnification, shape recovery, and optics through Transmission and Scanning Electron Microscopes (TEMs and SEMs, respectively). Then, we will discuss how we can derive shape from shading using needle diagrams, which capture surface orientations at each (x, y) pixel in the image. This procedure will motivate the use of Green's Theorem, "computational molecules", and a discrete approach to our standard unconstrained optimization problem. We will conclude by discussing more about recovering shape from shading for Hapke surfaces using initial curves and rotated coordinate systems.

8.1 Example Applications: Transmission and Scanning Electron Microscopes (TEMs and SEMs, respectively)

We will begin with a few motivating questions/observations:

- **How do TEMs achieve amazing magnification?** They are able to do so due to the fact that these machines are not restricted by the wavelength of the light they use for imaging (since they are active sensors, they image using their own "light", in this case electrons.
- **What are SEM images more enjoyable to look at than TEM images?** This is because SEM images reflect shading, i.e. differences in brightness based off of surface orientation. TEM images do not do this.
- **How do SEMs work?** Rely on an electron source/beam, magnetic-based scanning mechanisms, photodiode sensors to measure secondary electron current. Specifically:
  - Many electrons lose energy and create secondary electrons. Secondary electrons are what allow us to make measurements.
  - Secondary electron currents vary with surface orientation.
  - Objects can be scanned in a raster-like format.
  - Electron current is used to modulate a light ray. Magnification is determined by the degree of deflection.
  - Gold plating is typically used to ensure object is conductive in a vacuum.
  - Inclines/angles can be used for perturbing/measuring different brightness values.
  - From a **reflectance map** perspective, measuring brightness gives is the **slope** (a scalar), but it does not give us the gradient (a vector). This is akin to knowing speed, but not the velocity.

8.2 Shape from Shading: Needle Diagram to Shape

This is another class of problems from the overarching "Shape from Shading" problem. Let us first begin by defining what a needle diagram is:

A **needle diagram** is a 2D representation of the surface orientation of an object for every pixel in an image, i.e. for every (x, y) pair, we have a surface orientation (p, q), where $p \triangleq \frac{dz}{dt}, q \triangleq \frac{dz}{dy}$. Recall from photometric stereo that we cannot simply parameterize $Z(x, y)$; we can only parameterize the surface gradients $p(x, y)$ and $q(x, y)$.

In this problem, our goal is that given (p, q) for each pixel (i.e. given the needle diagram), recover z for each pixel. Note that this leads to an overdetermined problem (more constraints/equations than unknowns) [1]. This actually will allow us to reduce noise and achieve better results.

For estimating z, we have:

$$\begin{aligned} x : z(x) &= z(0) + \int_0^x p dx' \\ y : z(y) &= z(0) + \int_0^y q dy' \\ x\&y : z(x, y) &= z(0, 0) + \int p dx' + q dy' \end{aligned}$$

Let us define $\delta x' = p dx' + q dy$. Next, we construct a contour in the (x, y) plane of our (p, q) measurements, where the contour starts and ends at the origin, and passes through a measurement. Our goal is to have the integrals of p and q be zero over these contours, i.e.

$$\oint (p dx' + q dy') = 0$$

This is equivalent to "z being conserved" over the contour.

But note that these measurements are noisy, and since we estimate p and q to obtain estimates for z, this is not necessarily true.

Note that an easy way to break this problem down from one large problem into many smaller problems (e.g. for computational parallelization, greater accuracy, etc.) is to decompose larger contours into smaller ones - if z is conserved for a series of smaller loops, then this implies z is conserved for the large loop as well.

8.2.1 Derivation with Taylor Series Expansion

Let us now suppose we have a point (x, y), centered around a square loop with lengths equal to δ. Then, applying the formula above, we have that:

$$p\Big(x, y - \frac{\delta y}{2}\Big)\delta x + q\Big(x + \frac{\delta x}{2}, y\Big)\delta y - p\Big(x, y + \frac{\delta y}{2}\Big)\delta x - q\Big(x - \frac{\delta x}{2}, y\Big)\delta y = 0$$

If we now take the first-order Taylor Series Expansion of this equation above, we have:

$$\textbf{Expansion}: \;\; \Big(p(x,y) - \frac{\delta y}{2}\frac{\partial p(x,y)}{\partial y}\Big)\delta x + \Big(q(x,y) + \frac{\delta x}{2}\frac{\partial q(x,y)}{\partial x}\Big)\delta y - \Big(p(x,y) + \frac{\delta y}{2}\frac{\partial p(x,y)}{\partial y}\Big)\delta x - \Big(q(x,y) - \frac{\delta x}{2}\frac{\partial q(x,y)}{\partial x}\Big)\delta y = 0$$

$$\textbf{Rewriting}: \;\; p(x,y)\delta x - \frac{\delta y \delta x}{2}\frac{\partial p(x,y)}{\partial y} + q(x,y)\delta y + \frac{\delta x \delta y}{2}\frac{\partial q(x,y)}{\partial x} = p(x,y)\delta x + \frac{\delta y \delta x}{2}\frac{\partial p(x,y)}{\partial y} + q(x,y)\delta y - \frac{\delta x \delta y}{2}\frac{\partial q(x,y)}{\partial x}$$

$$\textbf{Simplifying}: \;\; \delta y \delta x \frac{\partial p(x,y)}{\partial y} = \delta x \delta y \frac{\partial q(x,y)}{\partial x}$$

$$\textbf{Solution}: \;\; \frac{\partial p(x,y)}{\partial y} = \frac{\partial q(x,y)}{\partial x}$$

This is consistent with theory, because since our parameters $p \approx \frac{\partial z}{\partial x}$ and $q \approx \frac{\partial z}{\partial y}$, then the condition approximately becomes (under perfect measurements):

$$\frac{\partial p(x,y)}{\partial y} = \frac{\partial}{\partial y}(\frac{\partial z}{\partial x}) = \frac{\partial^2 z}{\partial y \partial x}$$

$$\frac{\partial q(x,y)}{\partial x} = \frac{\partial}{\partial x}(\frac{\partial z}{\partial y}) = \frac{\partial^2 z}{\partial x \partial y}$$

$$\frac{\partial p(x,y)}{\partial y} = \frac{\partial q(x,y)}{\partial x} \implies \frac{\partial^2 z}{\partial y \partial x} = \frac{\partial^2 z}{\partial x \partial y}, \;\; \text{Which is consistent with Fubini's Theorem as well [2].}$$

Though this will not be the case in practice we are measuring p & q, and thus we will encounter some measurement noise.

8.2.2 Derivation with Green's Theorem

We can now show the same result via Green's Theorem, which relates contour integrals to area integrals:

$$\oint_L (Ldx + Mdy) = \iint_D \Big(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\Big)dxdy$$

Where the term $Ldx + Mdy$ on the lefthand side is along the boundary of the contour of interest, and the term $\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}$ is along the interior of the contour.

Green's Theorem is highly applicable in machine vision because we can reduce two-dimensional computations to one-dimensional computations. For instance, Green's Theorem can be helpful for:

- Computing the area of a contoured object/shape
- Computing the centroid of a blob or object in two-dimensional space, or more generally, geometric moments of a surface. Moments can generally be computed just by going around the boundary of a contour.

Let us now apply Green's Theorem to our problem:

$$\oint_L (pdx + qdy) = \iint_D \Big(\frac{\partial q(x,y)}{\partial x} - \frac{\partial p(x,y)}{\partial y}\Big) dxdy = 0$$

This requires $\frac{\partial q(x,y)}{\partial x} - \frac{\partial p(x,y)}{\partial y} = 0 \implies \frac{\partial q(x,y)}{\partial x} = \frac{\partial p(x,y)}{\partial y} \; \forall \; x, y \in D$.

We could solve for estimates of our unknowns of interest, p and q, using unconstrained optimization, but this will be more difficult than before. Let us try using a different tactic, which we will call "Brute Force Least Squares":

$$\min_{z(x,y)} \iint_D \Big(\frac{\partial z}{\partial x} - p\Big)^2 + \Big(\frac{\partial z}{\partial y} - q\Big)^2 dxdy$$

I.e. we are minimizing the squared distance between the partial derivatives of z with respect to x and y and our respective parameters over the entire image domain D.

However, this minimization approach requires having a finite number of variables, but here we are optimizing over a continuous function (which has an infinite number of variables). Therefore, we have infinite degrees of freedom. We can use **calculus of variations** here to help us with this. Let us try solving this as a discrete problem first.

8.3 Shape with Discrete Optimization

For this, let us first take a grid of unknowns $\{z_{ij}\}_{(i,j)\in D}$. Our goal is to minimize the errors of spatial derivatives of z with respect to p and q, our measurements in this case (given by $\{p_{ij}, q_{ij}\}_{(i,j)\in D}$. Our objective for this can then be written as:

$$\min_{\{z_{ij}\}} \sum_i \sum_j \Big(\frac{z_{i,j+1} - z_{i,j}}{\epsilon} - p_{i,j}\Big)^2 + \Big(\frac{z_{i+1,j} - z_{i,j}}{\epsilon} - q_{i,j}\Big)^2$$

Note that these discrete derivatives of z with respect to x and y present in the equation above use finite forward differences.

Even though we are solving this discretely, we can still think of this as solving our other unconstrained optimization problems, and therefore can do so by taking the first-order conditions of each of our unknowns, i.e. $\forall \; (k, l) \in D$. The FOCs are given by $|D|$ equations (these will actually be linear!):

$$\frac{\partial}{\partial z_{k,l}}(J(\{z_{i,j}\}_{(i,j)\in D}) = 0 \; \forall \; (k, l) \in D$$

Let us take two specific FOCs and use them to write a partial differential equation:

- **(k, l) = (i, j)**:

$$\frac{\partial}{\partial z_{k,l}}(J(\{z_{,j}\}_{(i,j)\in D}) = \frac{2}{\epsilon}\Big(\frac{z_{k,l+1} - z_{k,l}}{\epsilon} - p_{k,l}\Big) + \frac{2}{\epsilon}\Big(\frac{z_{k+1,l} - z_{k,l}}{\epsilon} - q_{k,l}\Big) = 0$$

- **(k, l) = (i+1, j+1)**:

$$\frac{\partial}{\partial z_{k,l}}(J(\{z_{,j}\}_{(i,j)\in D}) = \frac{2}{\epsilon}\Big(\frac{z_{k,l} - z_{k,l-1}}{\epsilon} - p_{k,l-1}\Big) + \frac{2}{\epsilon}\Big(\frac{z_{k,l} - z_{k-1,l}}{\epsilon} - q_{k-1,l}\Big) = 0$$

Gathering all terms (we can neglect the zs):

(1) $\frac{p_{k,l} - p_{k,l-1}}{\epsilon} \approx \frac{\partial p}{\partial x}$

(2) $\frac{q_{k,l} - q_{k-1,l}}{\epsilon} \approx \frac{\partial q}{\partial y}$

(3) $\frac{1}{\epsilon^2}((-z_{k,l+1} - z_{k+1,l} - z_{k,l-1} - z_{k-1,l}) + 4z_{k,l}) = 0 \approx -\Delta z = -\nabla^2 z$ **The Laplacian of** z

Where **(1)** + **(2)** + **(3)** = 0

Using the approximations of these three terms, our equation becomes:

$$\frac{\partial p}{\partial x} + \frac{\partial q}{\partial y} - \Delta z = 0 \implies \frac{\partial p}{\partial x} + \frac{\partial q}{\partial y} = \Delta z$$

This approach motivates the use of "computational molecules".

8.3.1 "Computational Molecules"

These are computational molecules that use finite differences [3] to estimate first and higher-order derivatives. They can be thought of as filters, functions, and operators that can be applied to images or other multidimensional arrays capturing spatial structural. Some of these are (please see the handwritten lecture notes for what these look like graphically):

1. $z_x = \frac{1}{\epsilon}(z(x,y) - z(x-1,y))$ (Backward Difference), $\frac{1}{\epsilon}(z(x+1,y) - z(x,y))$ (Forward Difference)
2. $z_y = \frac{1}{\epsilon}(z(x,y) - z(x,y-1))$ (Backward Difference), $\frac{1}{\epsilon}(z(x,y+1) - z(x,y))$ (Forward Difference)
3. $\Delta z = \nabla^2 z = \frac{1}{\epsilon^2}(4z(x,y) - (z(x-1,y) + z(x+1,y) + z(x,y-1) + z(x,y+1)))$
4. $z_{xx} = \frac{\partial^2 z}{\partial x^2} = \frac{1}{\epsilon^2}(z(x-1,y) - 2(x,y) + z(x+1,y))$
5. $z_{yy} = \frac{\partial^2 z}{\partial y^2} = \frac{1}{\epsilon^2}(z(x,y-1) - 2(x,y) + z(x,y+1))$

These computational molecules extend to much higher powers as well. Let us visit the Laplacian operator $\Delta(\cdot)$. This operator comes up a lot in computer vision:

- Definition: $\Delta z = \nabla^2 z = (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y})^T(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}) = \frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2}$
- The Laplacian is the lowest dimensional **rotationally-invariant** linear operator, i.e. for a rotated coordinate system (x', y') rotated from (x, y) by some rotation matrix $\mathbf{R} \in SO(2)$, we have:

  $$z_{x'x'} + z_{y'y'} = z_{xx} + z_{yy}$$

  I.e. the result of the Laplacian is the same in both coordinate systems.

As we can see, the Laplacian is quite useful in our derived solution above.

8.3.2 Iterative Optimization Approach

Let us return to our discrete optimization problem. Our derivation above is a **least-squares** solution. This turns out to be the discrete version of our original continuous problem. Since these first-order equations are linear, we can solve them as a system of linear equations with Gaussian elimination. But note that these processes take $O(N^3)$ time. We can avoid this complexity by taking advantage of the sparsity in these equations.

Iterative Approach: The sparse structure in our First-Order Equations allows us to use an iterative approach for shape estimation. Our "update equation" updates the current depth/shape estimate $z_{k,l}$ using its neighboring indices in two dimensions:

$$z_{k,l}^{(n+1)} = \frac{1}{4}(z_{k,l+1}^{(n)} + z_{k+1,l}^{(n)} + z_{k,l-1}^{(n)} + z_{k-1,l}^{(n)}) - \epsilon(p_{k,l} - p_{k,l-1}) - \epsilon(q_{k,l} - q_{k-1,l})$$

A few terminology/phenomenological notes about this update equation:

- The superscripts n and $n+1$ denote the number of times a given indexed estimate has been updated (i.e. the number of times this update equation has been invoked). It is essentially the iteration number.
- The subscripts k and l refer to the indices.
- The first term on the righthand side $\frac{1}{4}(\cdot)$ is the local average of $z_{k,l}$ using its neighbors.
- This iterative approach converges to the solution much more quickly than Gaussian elimination.
- This iterative approach is also used in similar ways for solving problems in the **Heat and Diffusion Equations** (also PDEs).
- This procedure can be parallelized so long as the computational molecules do not overlap/touch each other. For instance, we could divide this into blocks of size 3 x 3 in order to achieve this.
- From this approach, we can develop robust surface estimates!

8.3.3 Reconstructing a Surface From a Single Image

Recall this other shape from brightness problem we solved for Hapke surfaces (Lecture 8). For Hapke surfaces, we have that our brightness in the image (radiance) L is given by:

$$L = \sqrt{\frac{\cos\theta_i}{\cos\theta_e}} = \sqrt{\frac{\hat{\mathbf{n}} \cdot \hat{\mathbf{s}}}{\hat{\mathbf{n}} \cdot \hat{\mathbf{v}}}}$$

Recall from our last lecture that this gives us a simple reflectance map of straight lines in gradient (p, q) space. By rotating this gradient space coordinate system from $(p, q) \rightarrow (p', q')$, we can simplify our estimates for shape.

With this rotation, we also claimed that rotating the system in gradient space is equivalent to using the same rotation matrix $\mathbf{R}$ in our image space (x, y). Here we prove this:

$$\textbf{Rotating Points}: \; x' = x\cos\theta - y\sin\theta, \;\; y' = x\sin\theta + y\cos\theta$$

$$\textbf{Reverse Rotating Points}: \; x = x'\cos\theta + y'\sin\theta, \;\; y = -x'\sin\theta + y'\cos\theta$$

$$\textbf{Taking Derivatives}: \; \frac{\partial z}{\partial x'} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial x'} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial x'}, \;\; \frac{\partial z}{\partial y'} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial y'} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial y'}$$

$$\textbf{Combining the Above}: p' = p\cos\theta - q\sin\theta, \;\; q' = p\sin\theta + q\cos\theta$$

Then, in our rotated coordinate system where p' is along the brightness gradient, we have that:

$$p' = \frac{p_s p + q_s q}{\sqrt{p_s^2 + q_s^2}} = \frac{r_s E^2 - 1}{\sqrt{p_s^2 + q_s^2}}$$

(Where $p' \triangleq \frac{\partial z}{\partial x'}$ is the slope of the surface of interest in a particular direction. This phenomenon only holds for Hapke surfaces with linear isophotes. We can integrate this expression out for surface estimation:

$$z(x) = z(x_0) + \int_{x}^{x_0} p'(x)dx$$

Integrating out as above allows us to build a surface height profile of our object of interest. Can do this for the y-direction as well:

$$z(x, y) = z(x_0, y) + \int_{x}^{x_0} p'(x, y)dx$$

A few notes from this, which we touched on in lecture 8 as well:

- Adding a constant to z does not change our profile integrals, except by an offset. Therefore, in order to obtain absolute height measurements of z, we need to include initial values.
- In this case, we need an initial condition for every horizontal row/profile. This is the same as requiring an "initial curve", and allows us to effectively reduce our computations from 2D to 1D. Note from our previous lecture that these initial conditions are needed to determine the surface orientation/shape at interfaces between these different profiles.
- Let us examine what this looks like when we parameterize an initial curve with η:

  $$\text{Take } x(\eta), y(\eta), z(\eta), \text{ and rotate with } \xi$$

  Then we can compute δz to recover shape, along with δx and δy:

  1. $\delta x = \frac{q_s}{\sqrt{p_s^2+q_s^2}}\delta\xi$
  2. $\delta y = \frac{p_s}{\sqrt{p_s^2+q_s^2}}\delta\xi$
  3. $\delta z = \frac{[r_s E^2(x,y)-1]}{\sqrt{p_s^2+q_s^2}}\delta\xi$

  Note that we can adjust the speed of motion here by adjusting the parameter δ.

Next time, we will generalize this from Hapke reflectance maps to arbitrary reflectance maps!

8.4 References

1. Overdetermined System, https://en.wikipedia.org/wiki/Overdetermined_system
2. Fubini's Theorem, https://en.wikipedia.org/wiki/Fubini%27s_theorem
3. Finite Differences, https://en.wikipedia.org/wiki/Finite difference

9 Lecture 10: Characteristic Strip Expansion, Shape from Shading, Iterative Solutions

In this lecture, we will continue studying our shape from shading problem, this time generalizing it beyond Hapke surfaces, making this framework amenable for a range of machine vision tasks. We will introduce a set of ordinary differential equations, and illustrate how we can construct a linear dynamical system that allows us to iteratively estimate the depth profile of an object using both position and surface orientation.

9.1 Review: Where We Are and Shape From Shading

Shape from Shading (SfS) is one of the class of problems we study in this course, and is part of a larger class of problems focusing on machine vision from image projection. In the previous lecture, we solved shape from shading for Hapke surfaces using the **Image Irradiance Equation** $E(x, y) = R(p, q)$, where $R(p, q)$ is the reflectance map that builds on the Bidirectional Reflectance Distribution Function (BRDF).

Recall additionally that irradiance (brightness in the image) depends on:

1. Illumination
2. Surface material of object being imaged
3. Geometry of the object being imaged

Recall from the previous lecture that for Hapke surfaces, we were able to solve the SfS problem in a particular direction, which we could then determine object surface orientation along. We can integrate along this direction (the profile direction) to find height z, but recall that we gain no information from the other direction.

For the Hapke example, we have a rotated coordinate system governed by the following ODEs (note that xi will be the variable we parameterize our profiles along):

1. X-direction: $\frac{dx}{d\xi} = p_s$
2. Y-direction: $\frac{dy}{d\xi} = q_s$
3. By chain rule, Z-direction: $\frac{dz}{d\xi} = \frac{\partial z}{\partial x}\frac{dx}{d\xi} + \frac{\partial z}{\partial y}\frac{dy}{d\xi} = pp_s + qq_s = p_s p + q_s q$

Intuition: Infinitesimal steps in the image given by ξ gives $\frac{dx}{d\xi}$, $\frac{dy}{d\xi}$, and we are interested in finding the change in height $\frac{dz}{d\xi}$, which can be used for recovering surface orientation.

Note: When dealing with brightness problems, e.g. SfS, we have implicitly shifted to orthographic projection ($x = \frac{f}{Z_0}X, y = \frac{f}{Z_0}y$). These methods can be applied to perspective projection as well, but the mathematics makes the intuition less clear. We can model orthographic projection by having a **telecentric lens**, which effectively places the object really far away from the image plane.

We can solve the 3 ODEs above using a **forward Euler** method with a given step size. For small steps, this approach will be accurate enough, and accuracy here is not too important anyway since we will have noise in our brightness measurements.

Recall brightness of Hapke surfaces is given by:

$$E = \sqrt{\frac{\hat{\mathbf{n}} \cdot \hat{\mathbf{s}}}{\hat{\mathbf{n}} \cdot \hat{\mathbf{v}}}} = \sqrt{\frac{\cos\theta_i}{\cos\theta_e}} =$$

$$frac1 + p_s p + q_s q\sqrt{1 + p^2 + q^2}r_s \cdot \frac{1}{\sqrt{1 + p_s p + q_s q}}$$

Some cancellation and rearranging yields:

$$p_s p + q_s q = r_s E^2 - 1$$

Therefore, we have a direct relationship between our measured brightness E and our unknowns of interest:

$$\frac{\partial z}{\partial x} p_s + \frac{\partial z}{\partial y} q_s = p p_s + q q_s = r_s E^2 - 1$$

Note here, however, that we do not know surface orientation based on this, since again for Hapke surfaces, we only know slope in one of our two directions in our rotated gradient space (p', q'). A forward Euler approach will generate a set of independent profiles, and we do not have any information about the surface orientation at the interfaces of these independent profiles. This necessitates an **initial curve** containing **initial conditions** for each of these independent profiles. In 3D, this initial curve is parameterized by η, and is given by: $(x(\eta), y(\eta), z(\eta))$. Our goal is to find $z(\eta, \xi)$, where:

- η parameterizes the length of the initial curve
- ξ parameterizes along the 1D profiles we carve out

Therefore, we are able to determine the slope in a particular direction.

9.2 General Case

Let us now generalize our Hapke case above to be any surface. We begin with our Image Irradiance Equation:

$$E(x, y) = R(p, q)$$

Let us now consider taking a small step $\delta x, \delta y$ in the image, and our goal is to determine how z changes from this small step. We can do so using differentials and surface orientation:

$$\delta z = \frac{\partial z}{\partial x} \delta x + \frac{\partial z}{\partial y} \delta y = p \delta x + q \delta y$$

Therefore, if we know p and q, we can compute δz for a given δx and δy. But in addition to updating z using the equation above, we will also need to update p and q (intuitively, since we are still moving, the surface orientation can change):

$$\delta p = \frac{\partial p}{\partial x} \delta x + \frac{\partial p}{\partial y} \delta y$$
$$\delta q = \frac{\partial q}{\partial x} \delta x + \frac{\partial q}{\partial y} \delta y$$

This notion of updating p and q provides motivation for keeping track/updating not only (x, y, z), but also p and q. We can think of solving our problem as constructing a **characteristic strip** of the ODEs above, composed of $(x, y, z, p, q) \in \mathbb{R}^5$. In vector-matrix form, our updates to p and q become:

$$\begin{bmatrix} \delta p \\ \delta q \end{bmatrix} = \begin{bmatrix} r & s \\ s & t \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 z}{\partial x^2} & \frac{\partial^2 z}{\partial y \partial x} \\ \frac{\partial^2 z}{\partial x \partial y} & \frac{\partial^2 z}{\partial y^2} \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} p_x & p_y \\ q_x & q_y \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \mathbf{H} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix}$$

Where $\mathbf{H}$ is the Hessian of second spatial derivatives (x and y) of z. Note that from Fubuni's theorem and from our Taylor expansion from last lecture, $p_y = q_x$.

Intuition:

- First spatial derivatives $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ describe the **surface orientation** of an oject in the image.
- Second spatial derivatives $\frac{\partial^2 z}{\partial x^2}$, $\frac{\partial^2 z}{\partial y^2}$, and $\frac{\partial^2 z}{\partial x \partial y} = \frac{\partial^2 z}{\partial y \partial x}$ describe **curvature**, or the change in surface orientation.
- In 2D, the **curvature** of a surface is specified by the **radius of curvature**, or its inverse, **curvature.**
- In 3D, however, the curvature of a surface is specified by a Hessian matrix $\mathbf{H}$. This curvature matrix allows us to compute p and q.

One issue with using this Hessian approach to update p and q: how do we update the second derivatives r, s, and t? Can we use 3rd order derivatives? It turns out that seeking to update lower-order derivatives with higher-order derivatives will just generate increasingly more unknowns, so we will seek alternative routes. Let's try integrating our brightness measurements into what we already have so far:

$$\begin{aligned}
&\textbf{Image Irradiance Equation}: \;\; E(x, y) = R(p, q) \\
&\textbf{Chain Rule Gives}: \\
&\quad E_x = \frac{\partial E}{\partial x} = \frac{\partial R}{\partial p}\frac{\partial p}{\partial x} + \frac{\partial R}{\partial q}\frac{\partial q}{\partial x} \\
&\quad\quad = \frac{\partial R}{\partial p} r + \frac{\partial R}{\partial q} s \\
&\quad E_y = \frac{\partial E}{\partial y} = \frac{\partial R}{\partial p}\frac{\partial p}{\partial y} + \frac{\partial R}{\partial q}\frac{\partial q}{\partial y} \\
&\quad\quad = \frac{\partial R}{\partial p} s + \frac{\partial R}{\partial q} t
\end{aligned}$$

In matrix-vector form, we have:

$$\begin{bmatrix} E_x \\ E_y \end{bmatrix} = \begin{bmatrix} r & s \\ s & t \end{bmatrix} \begin{bmatrix} \frac{\partial R}{\partial p} \\ \\ \frac{\partial R}{\partial q} \end{bmatrix} = \mathbf{H} \begin{bmatrix} \frac{\partial R}{\partial p} \\ \\ \frac{\partial R}{\partial q} \end{bmatrix}$$

Notice that we have the same **Hessian** matrix that we had derived for our surface orientation update equation before!

Intuition: These equations make sense intuitively - brightness will be constant for constant surface orientation in a model where brightness depends only on surface orientation. Therefore, changes in brightness correspond to changes in surface orientation.

How do we solve for this Hessian $\mathbf{H}$? We can:

- Solve for E_x, E_y numerically from our brightness measurements.
- Solve for R_p, R_q using our reflectance map.

However, solving for $\mathbf{H}$ in this way presents us with 3 unknowns and only 2 equations (undetermined system). We cannot solve for $\mathbf{H}$, but we can still compute $\delta p, \delta q$. We can pattern match δx and δy by using a $(\delta x, \delta y)^T$ vector in the same direction as the gradient of the reflectance map, i.e.

$$\begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} R_p \\ R_q \end{bmatrix} \xi$$

Where the vector on the lefthand side is our step in x and y, our vector on the righthand side is our gradient of the reflectance map in gradient space (p, q), and ξ is the step size. Intuitively, this is the direction where we can "make progress". Substituting this equality into our update equation for p and q, we have:

$$\begin{aligned}
\begin{bmatrix} \delta p \\ \delta q \end{bmatrix} &= \mathbf{H} \begin{bmatrix} R_p \\ R_q \end{bmatrix} \xi \\
&= \begin{bmatrix} E_x \\ E_y \end{bmatrix} \delta\xi
\end{aligned}$$

Therefore, we can formally write out a system of 5 first-order ordinary differential equations (ODEs) that generate our characteristic strip as desired:

1. $\frac{dx}{d\xi} = R_p$
2. $\frac{dy}{d\xi} = R_q$
3. $\frac{dp}{d\xi} = E_x$
4. $\frac{dq}{d\xi} = E_y$

5. $\frac{dz}{d\xi} = pR_p + qR_q$ **(“Output Rule”)**

Though we take partial derivatives on many of the righthand sides, we can think of these quantities as measurements or derived variations of our measurements, and therefore they do not correspond to partial derivatives that we actually need to solve for. Thus, this is why we claim this is a system of ODEs, and not PDEs.

A few notes about this system of ODEs:

- This system of 5 ODEs explores the surface along the characteristic strip generated by these equations.
- Algorithmically, we (a) Look at/compute the brightness gradient, which helps us (b) Compute p and q, which (c) Informs us of R_p and R_q for computing the change in height z.
- ODEs 1 & 2 and ODEs 3 & 4 are two systems of equations that each determine the update for the other system (i.e. there is dynamic behavior between these two systems). The 5th ODE is in turn updated from the results of updating the other 4 ODE quantities of interest.
- The step in the image $(\mathbf{x}, \mathbf{y})$ depends on the gradient of the reflectance map in $(\mathbf{p}, \mathbf{q})$.
- Analogously, the step in the reflectance map $(\mathbf{p}, \mathbf{q})$ depends on the gradient of the image in $(\mathbf{x}, \mathbf{y})$.
- This system of equations necessitates that we cannot simply optimize with block gradient ascent or descent, but rather a process in which we dynamically update our variables of interest using our other updated variables of interest.
- This approach holds generally for any reflectance map $R(p, q)$.
- We can express our image irradiance equation as a first order, *nonlinear* PDE:

$$E(x, y) = R\Big(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}\Big)$$

Next, we will show how this general approach applied to a Hapke surface reduces to our previous SfS results for this problem.

9.2.1 Reducing General Form SfS for Hapke Surfaces

Recall reflectance map for Hapke surfaces:

$$R(p, q) = \sqrt{\frac{1 + p_s p + q_s q}{r_s}}$$

Taking derivatives using the system of 5 ODEs defined above, we have from the multivariate chain rule):

1. $\frac{\mathbf{dx}}{\mathbf{d\xi}} : R_p \triangleq \frac{\partial R}{\partial p} = \frac{1}{\sqrt{r_s}} \frac{1}{2\sqrt{1+p_s p+q_s q}} p_s$
2. $\frac{\mathbf{dy}}{\mathbf{d\xi}} : R_q \triangleq \frac{\partial R}{\partial q} = \frac{1}{\sqrt{r_s}} \frac{1}{2\sqrt{1+p_s p+q_s q}} q_s$
3. $\frac{\mathbf{dz}}{\mathbf{d\xi}} : pR_p + qR_q = \frac{p_s p+q_s q}{2\sqrt{r_s}\sqrt{1+p_s p+q_s q}}$

Since the denominator is common in all three of these derivative equations, we can just conceptualize this factor as a speed/step size factor. Therefore, we can simply omit this factor when we update these variables after taking a step. With this, our updates become:

1. $\delta x \leftarrow p_s$
2. $\delta y \leftarrow q_s$
3. $\delta z \leftarrow (p_s p + q_s q) = r_s E^2 - 1$

Which are consistent with our prior results using our Hapke surfaces. Next, we will apply this generalized approach to Scanning Electron Microscopes (SEMs):

9.2.2 Applying General Form SfS to SEMs

For SEMs, the reflectance map is spherically symmetric around the origin:

$$R(p, q) = f(p^2 + q^2) \text{ for some } f : \mathbb{R} \to \mathbb{R}$$

Using our system of 5 ODEs derived in the general case, we can compute our updates by first calculating derivatives $\frac{dx}{d\xi}$, $\frac{dy}{d\xi}$, and $\frac{dz}{d\xi}$:

1. $\mathbf{\frac{dx}{d\xi}} : R_p \triangleq \frac{\partial R}{\partial p} = 2f'(p^2 + q^2)p$
2. $\mathbf{\frac{dy}{d\xi}} : R_q \triangleq \frac{\partial R}{\partial q} = 2f'(p^2 + q^2)q$
3. $\mathbf{\frac{dz}{d\xi}} : pR_p + qR_q = 2f'(p^2 + q^2)(p^2 + q^2)$

Again, here we can also simplify these updates by noting that the term $2f'(p^2 + q^2)$ is common to all three derivatives, and therefore this factor can also be interpreted as a speed factor that only affects the step size. Our updates then become:

1. $\delta x \leftarrow p$
2. $\delta y \leftarrow q$
3. $\delta z \leftarrow p^2 + q^2$

This tells us we will be taking steps along the brightness gradient. Our solution generates **characteristic strips** that contain information about the surface orientation. To continue our analysis of this SfS problem, in addition to defining characteristic strips, it will also be necessary to define the base characteristic.

9.3 Base Characteristics

Recall that the characteristic strip that our system of ODEs solves for is given by a 5-dimensional surface composed of:

$$\begin{bmatrix} x & y & z & p & q \end{bmatrix}^T \in \mathbb{R}^5$$

Another important component when discussing our solution to this SfS problem is the **base characteristic**, which is the projection of the characteristic strip onto the x, y image plane:

$$\begin{bmatrix} x(\xi) & y(\xi) \end{bmatrix}^T = \text{projection}_{x,y}\{\textbf{characteristic strip}\} = \text{projection}_{x,y}\{\begin{bmatrix} x & y & z & p & q \end{bmatrix}^T\}$$

A few notes/questions about base characteristics?

- **What are base characteristics?** These can be thought of as the profiles in the (x, y) plane that originate (possibly in both directions) from a given point in the initial curve.
- **What are base characteristics used for?** These are used to help ensure we explore as much of the image as possible. We can interpolate/average (depending on how dense/sparse a given region of the image is) different base characteristics to "fill out" the solution over the entire image space.
- **How are base characteristics used in practice?** Like the solution profiles we have encountered before, base characteristics are independent of one another - this allows computation over them to be parallelizable. **Intuition**: These sets of base characteristics can be thought of like a **wavefront** propagating outward from the initial curve. We expect the solutions corresponding to these base characteristics to move at similar "speeds".

9.4 Analyzing "Speed"

When considering how the speeds of these different solution curves vary, let us consider ways in which we can set constant step sizes/speeds. Here are a few, to name:

1. **Constant step size in z**: This is effectively stepping "contour to contour" on a topographic map.
   Achieved by: Dividing by $pR_p + qR_q \implies \frac{dz}{d\xi} = 1$.
2. **Constant step size in image**: A few issues with this approach. First, curves may run at different rates. Second, methodology fails when $\sqrt{R_p^2 + R_q^2} = 0$.
   Achieved by: Dividing by $\sqrt{R_p^2 + R_q^2} \implies \frac{d}{d\xi}(\sqrt{\delta x^2 + \delta y^2}) = 1$.

3. **Constant Step Size in 3D/Object**: Runs into issues when $R_p = R_q = 0$.
   Achieved by: Dividing by $\sqrt{R_p^2 + R_q^2 + (pR_p + qR_q)^2} \implies \sqrt{(\delta x)^2 + (\delta y)^2 + (\delta z)^2} = 1$.

4. **Constant Step Size in Isophotes**: Here, we are effectively taking constant steps in brightness. We will end up dividing by the dot product of the brightness gradient and the gradient of the reflectance map in (p, q) space.
   Achieved by: Dividing by $(\frac{\partial E}{\partial x}\frac{\partial R}{\partial p} + \frac{\partial E}{\partial y}\frac{\partial R}{\partial q})\delta\xi = ((E_x, E_y) \cdot (R_p, R_q))\delta\xi \implies \delta E = 1$.

9.5 Generating an Initial Curve

While only having to measure a single initial curve for profiling is far better than having to measure an entire surface, it is still undesirable and is not a caveat we are satisfied with for this solution. Below, we will discuss how we can generate an initial curve to avoid having to measure one for our SfS problem.

As we discussed, it is undesirable to have to measure an initial curve/characteristic strip, a.k.a. measure:

$$\begin{bmatrix} x(\xi) & y(\xi) & z(\xi) & p(\xi) & q(\xi) \end{bmatrix}^T$$

But we do not actually need to measure all of these. Using the image irradiance equation we can calculate $\frac{\partial z}{\partial \eta}$:

$$\begin{aligned} E(x, y) = R(p, q) \\ \frac{\partial z}{\partial \eta} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial \eta} \\ &= p\frac{\partial x}{\partial \eta} + q\frac{\partial y}{\partial \eta} \end{aligned}$$

Where p and q in this case are our unknowns. Therefore, in practice, we can get by with just initial curve, and do not need the full initial characteristic strip - i.e. if we have $x(\eta), y(\eta)$, and $z(\eta)$, then we can compute the orientation from the reflectance map using our brightness measurements.

9.5.1 Using Edge Points to Autogenerate an Initial Curve

An even more sweeping question: do we even need an initial curve? Are there any special points on the object where we already know the orientation without a measurement? These points are along the **edge**, or **occluding boundary**, of our objects of interest. Here, we know the surface normal of each of these points. Could we use these edge points as "starting points" for our SfS solution?

It turns out, unfortunately, that we cannot. This is due to the fact that as we infinisimally approach the edge of the boundary, we have that $\left|\frac{\partial z}{\partial x}\right| \to \infty, \left|\frac{\partial z}{\partial y}\right| \to \infty$. Even though the slope $\frac{q}{p}$ is known, we cannot use it numerically/iteratively with our step updates above.

9.5.2 Using Stationary Points to Autogenerate an Initial Curve

Let us investigate another option for helping us to avoid having to generate an initial curve. Consider the brightest (or darkest) point on an image, i.e. the **unique, global, isolated extremum**. This point is a stationary point in (p, q) space, i.e.

$$\frac{\partial R}{\partial p} = \frac{\partial R}{\partial q} = 0, \text{ granted that } R(p, q) \text{ is differentiable.}$$

This would allow us to find the surface orientation (p, q) solely through optimizing the reflectance map function $R(p, q)$. One immediate problem with this stationary point approach is that we will not be able to step x and y, since, from our system of ODEs, we have that at our stationary points, our derivatives of x and y are zero:

$$\begin{aligned} \frac{dx}{d\xi} &= R_p = 0 \\ \frac{dy}{d\xi} &= R_q = 0 \end{aligned}$$

Additionally, if we have stationary brightness points in image space, we encounter the "dual" of this problem. By definition stationary points in the image space imply that:

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} = 0$$

This in turn implies that p and q cannot be stepped:

$$\frac{dp}{d\xi} = \frac{\partial E}{\partial x} = 0$$
$$\frac{dq}{d\xi} = \frac{\partial E}{\partial y} = 0$$

Intuition: Intuitively, what is happening with these two systems (each of 2 ODEs ((x, y) or p, q), as we saw in our 5 ODE system above) is that using stationary points in one domain amounts to updates in the other domain going to zero, which in turn prevents the other system's quantities of interest from stepping. Since δz depends on $\delta x, \delta y, \delta p, \delta q$, and since $\delta x, \delta y, \delta p, \delta q = 0$ when we use stationary points as starting points, this means we cannot ascertain any changes in δz along these points.

Note: The extremum corresponding to a stationary point can be **maximum**, as it is for Lambertian surfaces, or a **minimum**, as it is for Scanning Electron Microscopes (SEM).

However, even though we cannot use stationary points themselves as starting points, could we use the area around them? If we stay close enough to the stationary point, we can approximate that these neighboring points have nearly the same surface orientation.

- **Approach 1**: Construct a local tangent plane by extruding a circle in the plane centered at the stationary point with radius ϵ - this means all points in this plane will have the same surface orientation as the stationary point. Note that mathematically, a local 2D plane on a 3D surface is equivalent to a 2-manifold [1]. This is good in the sense that we know the surface orientation of all these points already, but not so great in that we have a degenerate system - since all points have the same surface orientation, under this model they will all have the same brightness as well. This prevents us from obtaining a unique solution.
- **Approach 2**: Rather than constructing a local planar surface, let us take a **curved surface** with **non-constant surface orientation** and therefore, under this model, **non-constant brightness**.

9.5.3 Example

Suppose we have a reflectance map and surface function given by:

$$\textbf{Reflectance Map}: \; R(p, q) = p^2 + q^2$$
$$\textbf{Surface Function}: \; z(x, y) = x^2 + y^2$$

Then, we can compute the derivatives of these as:

$$p = \frac{\partial z}{\partial x} = 2x$$
$$q = \frac{\partial z}{\partial y} = 4y$$
$$E(x, y) = R(p, q) = p^2 + q^2 = 4x^2 + 16y^2$$
$$E_x = \frac{\partial E}{\partial x} = 8x$$
$$E_y = \frac{\partial E}{\partial y} = 32y$$

The brightness gradient $(\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}) = (8x, 32y) = (0, 0)$ at the origin $x = 0, y = 0$.

Can we use the brightness gradient to estimate local shape? It turns out the answer is **no**, again because of stationary points. But if we look at the second derivatives of brightness:

$$E_{xx} = \frac{\partial^2 E}{\partial x^2} = \frac{\partial E}{\partial x}(8x) = 8$$
$$E_{yy} = \frac{\partial^2 E}{\partial y^2} = \frac{\partial E}{\partial y}(32y) = 32$$
$$E_{xy} = \frac{\partial^2 E}{\partial x \partial y} = \frac{\partial}{\partial x}(32y) = \frac{\partial}{\partial y}(8y) = 0$$

These second derivatives, as we will discuss more in the next lecture, will tell us some information about the object's shape. As we have seen in previous lectures, these second derivatives can be computed by applying computational molecules to our brightness measurements.

High-Level Algorithm

(NOTE: This will be covered in greater detail next lecture) Let's walk through the steps to ensure we can autogenerate an initial condition curve without the need to measure it:

1. Compute stationary points of brightness.
2. Use 2nd derivatives of brightness to estimate local shape.
3. Construct small cap (non-planar to avoid degeneracy) around the stationary point.
4. Begin solutions from these points on the edge of the cap surrounding the stationary point.

9.6 References

1. Manifold, https://en.wikipedia.org/wiki/Manifold

10 Lecture 11: Edge Detection, Subpixel Position, CORDIC, Line Detection, (US 6,408,109)

In this lecture, we will introduce some discussion on how patents work, their stipulations, and make our discussion explicit by looking at a patent case study with sub-pixel accuracy edge detection. You can find the patent document on Stellar as well under "Materials" → "US patent 6,408,109".

10.1 Background on Patents

We will begin with some fun facts about industrial machine vision:

- Integrated circuit boards cannot be manufactured without machine vision
- Pharmaceutical chemicals also cannot be manufactured without machine vision

How do entrepreneurs and industrial companies ensure their inventions are protected, while still having the chance to disseminate their findings with society? This si done through **patents**. Patents:

- Can be thought of as a "contract with society" - you get a limited monopoly on your idea, and in turn, you publish the technical details of your approach.
- Can help to reduce litigation and legal fees.
- Can be used by large companies as "ammunition" for "patent wars".

Some "rules" of patents:

- No equations are included in the patent (no longer true)
- No greyscale images - only black and white
- Arcane grammar is used for legal purposes - "comprises", "apparatus", "method", etc.
- References of other patents are often included - sometimes these are added by the patent examiner, rather than the patent authors
- Most patents end with something along the lines of "this is why our invention was necessary" or "this is the technical gap our invention fills"
- Software is not patentable - companies and engineers get around this by putting code on hardware and patenting the "apparatus" housing the code.
- It is also common to include background information (similar to related literature in research).

Now that we have had a high-level introduction to patents, let us turn to focus on one that describes an apparatus for sub-pixel accuracy edge finding.

10.2 Patent Case Study: Detecting Sub-Pixel Location of Edges in a Digital Image

To put this problem into context, consider the following:

- Recall that images typically have large regions of uniform/homogeneous intensity
- Image arrays are very memory-dense. A more sparse way to transfer/convey information about an image containing edges is to use the locations of edges as region boundaries of the image. This is one application of edge finding.
- This patent achieves 1/40$^{\text{th}}$ pixel accuracy.

This methodology and apparatus seeks to find edges of objects in digital images at a high sub-pixel accuracy level. To do so, the authors leverage different detectors, or kernels. These kernels are similar to some of the computational molecules we have already looked at in this course:

- **Robert's Cross**: This approximates derivatives in a coordinate system rotated 45 degrees (x', y'). The derivatives can be approximated using the $K_{x'}$ and $K_{y'}$ kernels:

$$\frac{\partial E}{\partial x'} \rightarrow K_{x'} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$
$$\frac{\partial E}{\partial y'} \rightarrow K_{y'} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

- **Sobel Operator**: This computational molecule requires more computation and it is not as high-resolution. It is also more robust to noise than the computational molecules used above:

$$\frac{\partial E}{\partial x} \rightarrow K_x = \begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
$$\frac{\partial E}{\partial y} \rightarrow K_y = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- **Silver Operators**: This computational molecule is designed for a hexagonal grid. Though these filters have some advantages, unfortunately, they are not compatible with most cameras as very few cameras have a hexagonal pixel structure.

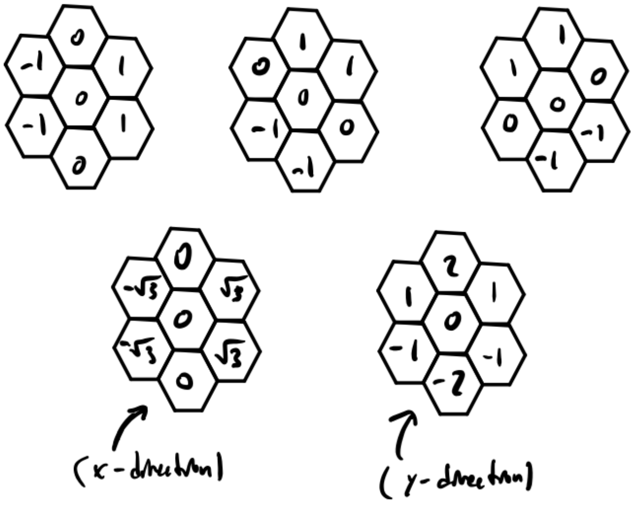


Figure 2: Silver Operators with a hexagonal grid.

For this specific application, we can compute approximate brightness gradients using the filters/operators above, and then we can convert these brightness gradients from Cartesian to polar coordinates to extract brightness gradient magnitude and direction (which are all we really need for this system). In the system, this is done using the CORDIC algorithm [1].

10.2.1 High-Level Overview of Edge Detection System

At a high level, we can divide the system into the following chronological set of processes/components:

1. **Estimate Brightness Gradient**: Given an image, we can estimate the brightness gradient using some of the filters defined above.

2. **Compute Brightness Gradient Magnitude and Direction**: Using the CORDIC algorithm, we can estimate the brightness gradient magnitude and direction. The CORDIC algorithm does this iteratively through a corrective feedback mechanism (see reference), but computationally, only uses SHIFT, ADD, SUBTRACT, and ABS operations.

3. **Choose Neighbors and Detect Peaks**: This is achieved using brightness gradient magnitude and direction and a procedure called non-maximum suppression [2].

   First, using gradient magnitude and direction, we can find edges by looking across the 1D edge (we can search for this edge using the gradient direction G_θ, which invokes Non-Maximum Suppression (NMS). We need to quantize into 8 (Cartesian) or 6 (Polar) regions - this is known as coarse direction quantization.

   Finally, we can find a peak by fitting three points with a parabola (note this has three DOF). This approach will end up giving us accuracy up to 1/10th of a pixel. To go further, we must look at the assumptions of gradient variation with position, as well as:

   - Camera optics
   - Fill factor of the chip sensor
   - How in-focus the image is
   - How smooth the edge transition is

The authors of this patent claim that edge detection performance is improved using an optimal value of "s" (achieved through interpolation and bias correction), which we will see later. For clarity, the full system diagram is here:

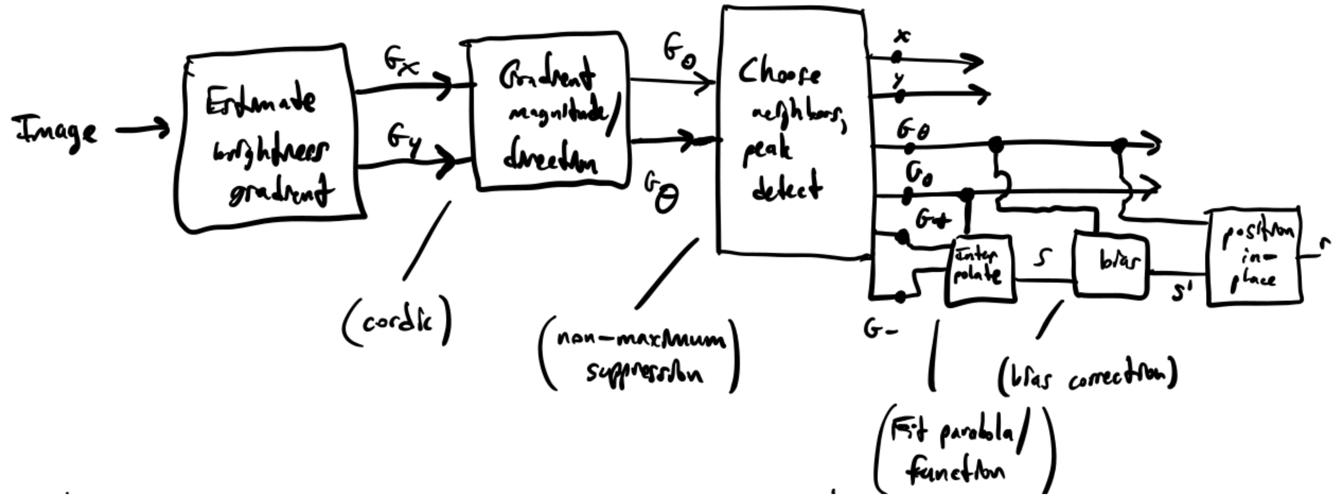


Image
Estimate brightness gradient
Gx
Gy
Gradient magnitude/ direction
G0
Gθ
Choose neighbors, peak detect
x
y
Gθ
G0
G+
G-
Inter polate
s
bias
s'
position in-place
r
(cordic)
(non-maximum suppression)
(Fit parabola/ function)
(bias correction)

Figure 3: Aggregate edge detection system. The steps listed in the boxes correspond to the steps outlined in the procedure above.

Some formulas for this system:

1. $G_0 = \sqrt{G_x^2 + G_y^2}$ **(gradient estimation)**
2. $G_\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$ **(gradient estimation)**
3. $R_0 = \max(|G_x|, |G_y|)$ **(octant quantization)**
4. $S_0 = \min(|G_x|, |G_y|)$ **(octant quantization)**

At a high level, the apparatus discussed in this patent is composed of:

- Gradient estimator
- Peak detector
- Sub-pixel interpolator

Next, let us dive in more to the general edge detection problem.

10.3 Edges & Edge Detection

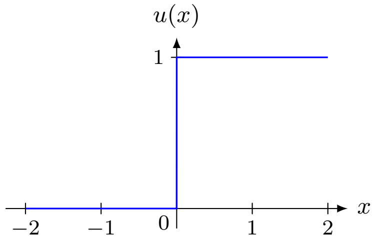
Let us begin by precisely defining what we mean by edges and edge detection:

- **Edge**: A point in an image where the image brightness gradient reaches a local maximum in the image brightness gradient direction. Additionally, an edge is where the second derivative of image brightness (can also be thought of as the gradient of the image brightness gradient) crosses zero. We can look at finding the zeros of these 2nd derivatives as a means to compute edges.
- **Edge Detection**: A process through which we can determine the location of boundaries between image regions that are roughly uniform in brightness.

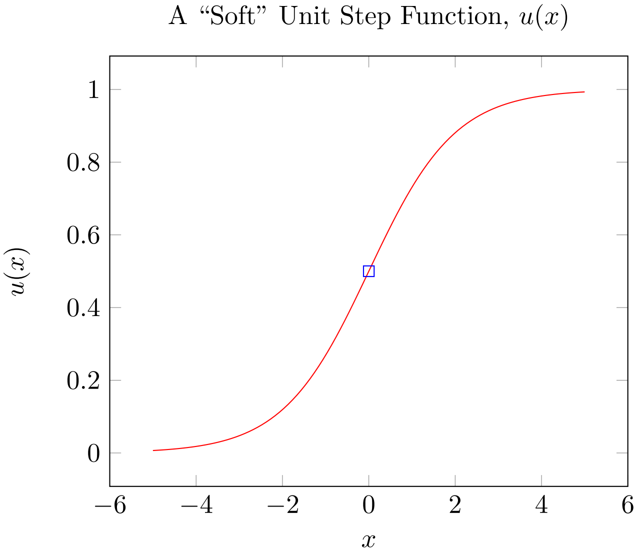
10.3.1 Finding a Suitable Brightness Function for Edge Detection

Let us first approximate an edge brightness function using a step function, given by $u(x)$, such that:

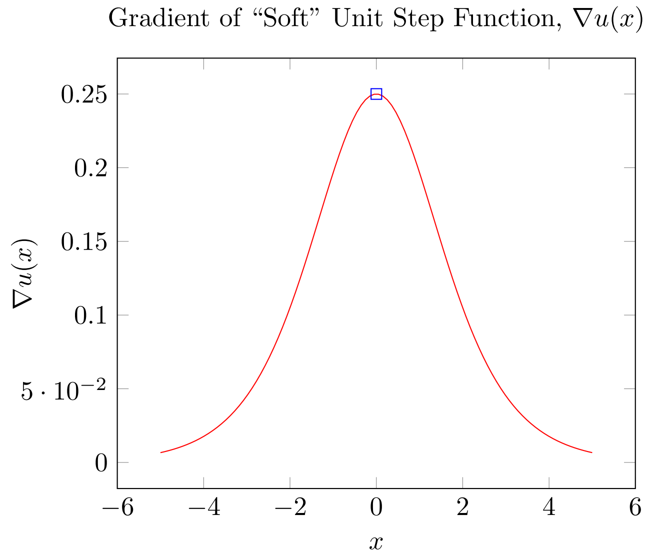
$$u(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



Using this cross-section across the edge to model the edge actually causes problems arising from aliasing: since we seek to find the location of an edge in a discrete, and therefore, sampled image, and since the edge in the case of $u(x)$ is infinitely thin, we will not be able to find it due to sampling. In Fourier terms, if we use a perfect step function, we introduce artificially high (infinite) frequencies that prevent us from sampling without aliasing effects. Let us instead try a "soft" step function, i.e. a "sigmoid" function: $\sigma(x) = \frac{1}{1+e^{-x}}$. Then our $u(x)$ takes the form:

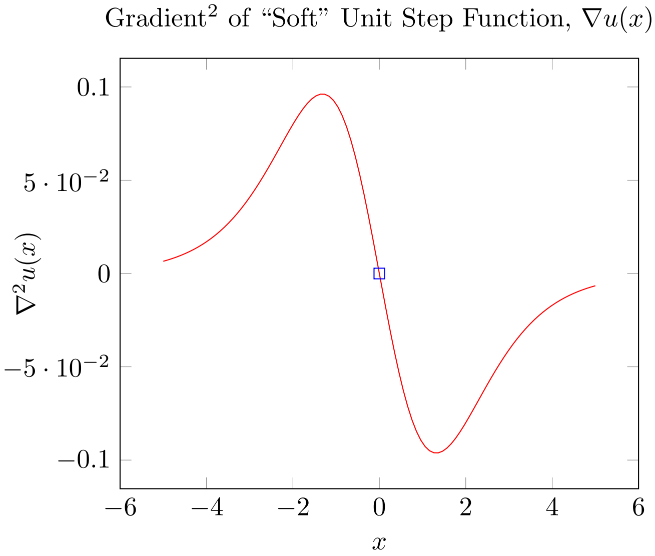


The gradient of this brightness across the edge, given by $\nabla u(x)$ (or $\frac{du}{dx}$ in one dimension), is then given by the following. Notice that the location of the maximum matches the inflection point in the graph above:



Gradient of “Soft” Unit Step Function, $\nabla u(x)$

As we mentioned above, we can find the location of this edge by looking at where the second derivative of brightness crosses zero, a.k.a. where $\nabla(\nabla u(x)) = \nabla^2 u(x) = 0$. Notice that the location of this zero is given by the same location as the inflection point of $u(x)$ and the maximum of $\nabla u(x)$:



Gradient^2 of “Soft” Unit Step Function, $\nabla u(x)$

For those curious, here is the math behind this specific function, assuming a sigmoid for $u(x)$:

1. $u(x) = \frac{1}{1+\exp{(-x)}}$

2. $\nabla u(x) = \frac{du}{dx} = \frac{d}{dx}\left(\frac{1}{1+\exp -x}\right) = \frac{\exp(-x)}{(1+\exp(-x))^2}$

3. $\nabla^2 u(x) = \frac{d^2u}{dx^2} = \frac{d}{dx}(\frac{\exp(-x)}{(1+\exp(-x))^2}) = \frac{-\exp(-x)(1+\exp(-x))^2+2\exp(-x)(1+\exp(-x))\exp(-x)}{(1+\exp(-x))^4}$

Building on top of this framework above, let us now move on to brightness gradient estimation.

10.3.2 Brightness Gradient Estimation

This component of this studied edge detection framework estimates the magnitude and direction of the brightness gradient. Let us look at how this is derived for different filters:

Robert’s Cross Gradient: Since this estimates derivatives at 45 degree angles, the pixels are effectively further apart, and this means there will be a constant of proportionality difference between the magnitude of the gradient estimated here and with a normal (x, y) coordinate system:

$$\sqrt{E_{x'}^2 + E_{y'}^2} \propto \sqrt{E_x^2 + E_y^2}$$

Next, we will look at the Sobel operator. For this analysis, it will be helpful to recall the following result from Taylor Series:

$$f(x+\delta x) = f(x) + \delta x f'(x) + \frac{(\delta x)^2}{2!}f''(x) + \frac{(\delta x)^3}{3!}f'''(x) + \frac{(\delta x)^4}{24}f^{(4)}(x) + ... = \sum_{i=0}^{\infty}\frac{(\delta x)^i f^{(i)}(x)}{i!}, \text{ where } 0! \triangleq 1$$

Let us first consider the simple two-pixel difference operator (in the x-direction/in the one-dimensional case), i.e. $\frac{dE}{dx} \rightarrow K_x = \frac{1}{\delta}(-1\ 1)$. Let us look at the **forward difference** and **backward difference** when this operator is applied:

1. **Forward Difference**: $\frac{\partial E}{\partial x} \approx \frac{f(x+\delta x)-f(x)}{\delta x} = f'(x) + \frac{\delta x}{2}f''(x) + \frac{(\delta x)^2}{6}f'''(x) + ...$
2. **Backward Difference**: $\frac{\partial E}{\partial x} \approx \frac{f(x)-f(x-\delta x)}{\delta x} = -f'(x) - \frac{\delta x}{2}f''(x) + \frac{(\delta x)^2}{6}f'''(x) + ...$

Notice that for both of these, if $f''(x)$ is large, i.e. if $f(x)$ is nonlinear, then we will have second-order error terms that appear in our estimates. In general, we want to aim for removing these lower-order error terms. If we average the forward and backward differences, however, we can see that these second-order error terms disappear:

$$\frac{\frac{f(x+\delta x)-f(x)}{\delta x} + \frac{f(x)-f(x-\delta x)}{\delta x}}{2} = f'(x) + \frac{(\delta x)^2}{6}f'''(x) + ...$$

Now we have increased the error term to 3rd order, rather than 2nd order! As a computational molecule, this higher-order filter Sobel operator looks like $\frac{dE}{dx} \rightarrow K_x = \frac{1}{2\delta}(-1\ 0\ 1)$. But we can do even better! So long as we do not need to have a pixel at our proposed edge, we can use a filter of three elements spanning $(x - \frac{\delta}{2}\ x\ x + \frac{\delta}{2})$. There is no pixel at x but we can still compute the derivative here. This yields an error that is 0.25 the error above due to the fact that our pixels are $\frac{\delta}{2}$ apart, as opposed to δ apart:

$$\text{error} = \frac{(\frac{x\delta}{2})^2}{6}f'''(x)$$

This makes sense intuitively, because the closer together a set of gradient estimates are, the more accurate they will be. We can incorporate y into the picture, making this amenable for two-dimensional methods as desired, by simply taking the center of four pixels, given for each dimension as:

$$\frac{\partial E}{\partial x} \approx K_x = \frac{1}{2\delta_x}\begin{bmatrix}-1 & 1\\-1 & 1\end{bmatrix}$$
$$\frac{\partial E}{\partial y} \approx K_y = \frac{1}{2\delta_y}\begin{bmatrix}-1 & -1\\1 & 1\end{bmatrix}$$

The proposed edge is in the middle of both of these kernels, as shown below:

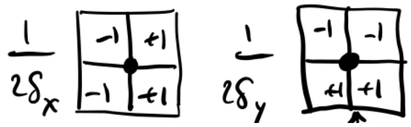


Figure 4: We can estimate the brightness gradient with minimal error by estimating it at the point at the center of these 2D filters.

Estimating these individually in each dimension requires 3 operations each for a total of 6 operations, but if we are able to take the common operations from each and combine them either by addition or subtraction, this only requires 4 operations. Helpful especially for images with lots of pixels.

Next, we will discuss the 3-by-3 Sobel operator. We can think of this Sobel operator (in each dimension) as being the discrete convolution of a 2-by-2 horizontal or vertical highpass/edge filer with a smoothing or averaging filter:

1. **x-direction**: $\frac{1}{2\delta_x}\begin{bmatrix}-1 & 1\\-1 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix}$
2. **y-direction**: $\frac{1}{2\delta_y}\begin{bmatrix}-1 & -1\\1 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}-1 & -2 & -1\\0 & 0 & 0\\1 & 2 & 1\end{bmatrix}$

A few notes about the derivation above:

- The convolution used is a "padded convolution" [3], in which, when implemented, when the elements of the filter/kernel (in this case, the averaging kernel) are not aligned with the image, they are simply multiplied by zero. Zero padding is the most common padding technique, but there are other techniques as well, such as wraparound padding.
- This approach avoids the half-pixel (in which we estimate an edge that is not on a pixel) that was cited above.
- Smoothing/averaging is a double edge sword, because while it can reduce/remove high-frequency noise by filtering, it can also introduce undesirable blurring.

Next, we will look at how the brightness gradient is converted from Cartesian to Polar coordinates:

$$(E_x, E_y) \rightarrow (E_0, E_\theta)$$

$$E_0 = \sqrt{E_x^2 + E_y^2}$$

$$E_\theta = \tan^{-1}\left(\frac{E_y}{E_y}\right)$$

Finally, we conclude this lecture by looking at appropriate values of s for quadratic and triangular functions. This assumes we have three gradient measurements centered on G_0: (1) G_-, (2) G_0, and (3) G_+. Let us look at the results for these two types of functions:

1. **Quadratic**: $s = \frac{G_+ - G_-}{4(G_0 - \frac{1}{2}(G_+ - G_-))}$, this results in $s \in [-\frac{1}{2}, \frac{1}{2}]$.
2. **Triangular**: $s = \frac{G_+ - G_-}{2(G_0 - \min(G_+, G_-))}$

A few notes about these approaches:

- Note that in each case, we only want to keep if the magnitude G_0 is a local maximum, i.e. $G_0 > G_+$ and $G_0 \geq G_-$.
- In the quadratic case, we can parameterize the curve with three data points using three degrees of freedom, i.e. $ax^2 + bx + c = 0$. With this approach, $b \approx$ first derivative and $a \approx$ second derivative.

10.4 References

1. CORDIC Algorithm, https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/
2. Non-Maximum Supression, http://justin-liang.com/tutorials/canny/#suppression
3. Padded Convolution, https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

11 Lecture 12: Blob analysis, Binary Image Processing, Use of Green's Theorem, Derivative and Integral as Convolutions

In this lecture, we will continue our discussion of intellectual property, and how it relevant for all scientists and engineers. We will then elaborate on some of the specific machine vision techniques that were used in this patent, as well as introduce some possible extensions that could be applicable for this patent as well.

11.1 Types of Intellectual Property

Though it is not related to the technical content of machine vision, being familiar with different types and degrees of intellectual property (IP) is crucial to professional success. Below, we discuss some of these different types of intellectual property.

- **Patents**: One major type of these is utility and design patents. In these, the authors are required to disclose the "best mode" of performance. For convenience, here are some notes on patents from our previous lecture:
  - Can be thought of as a "contract with society" - you get a limited monopoly on your idea, and in turn, you publish the technical details of your approach.
  - Can help to reduce litigation and legal fees.
  - Can be used by large companies as "ammunition" for "patent wars".

Some “rules” of patents:

  - No equations are included in the patent (no longer true)
  - No greyscale images - only black and white
  - Arcane grammar is used for legal purposes - “comprises”, “apparatus”, “method”, etc.
  - References of other patents are often included - sometimes these are added by the patent examiner, rather than the patent authors
  - Most patents end with something along the lines of “this is why our invention was necessary” or “this is the technical gap our invention fills”
  - Software is not patentable - companies and engineers get around this by putting code on hardware and patenting the “apparatus” housing the code.
  - It is also common to include background information (similar to related literature in research).

- **Copyright**:
  - Books, song recordings, choreographs
  - Exceptions: presenting (fractional pieces of) information from another author
- **Trademarks**:
  - Must be unique for your field (e.g. Apple vs. Apple).
  - Cannot use common words - this is actually one reason why many companies have slightly misspelled combinations of common words.
  - Can use pictures, character distortions, and color as part of the trademark.
  - No issues if in different fields.
- **Trade Secret**
  - No protections, but not spilled, lasts forever
  - Can enforce legal recourse with Non-Disclosure Agreement (NDA)

11.2 Edge Detection Patent Methodologies

For this next section, we will direct our attention toward covering concepts that were discussed in the edge detection patent (6,408,109). Each of these sections will be discussed in further detail below. Before we get into the specifics of the patent again, it is important to point out the importance of edge detection for higher-level machine vision tasks, such as:

- Attitude (pose estimation) of an object
- Object recognition
- Determining to position

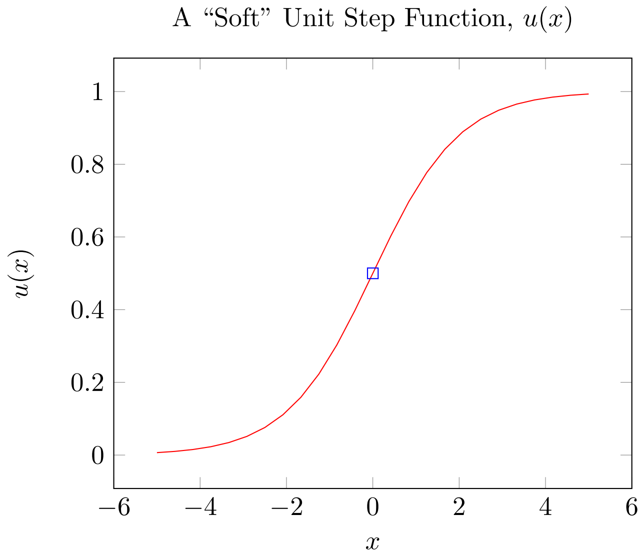
We will touch more on these topics in later lectures.

11.2.1 Finding Edge with Derivatives

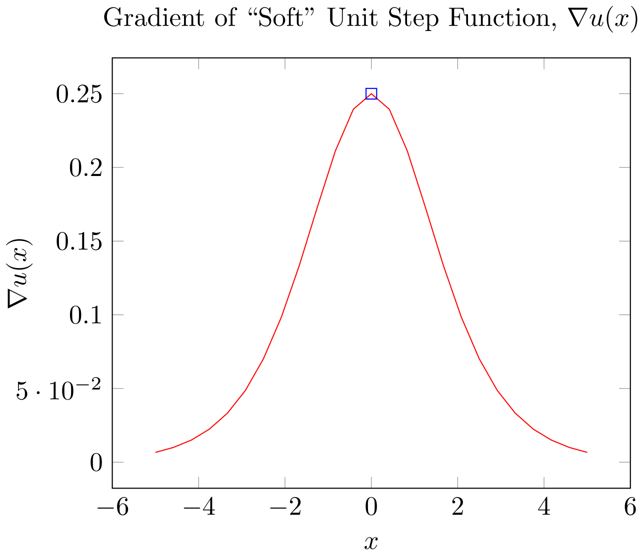
Recall that we find a proposed edge by finding an **inflection point** of the brightness $E(x, y)$. The following methods for finding this point are equivalent:

- Finding an inflection point of brightness $E(x, y)$.
- Finding maximum of brightness gradient magnitude/first derivative $|\nabla E(x, y)|$.
- Finding zero crossing of Laplacian/second derivative $\nabla^2 E(x, y)$.

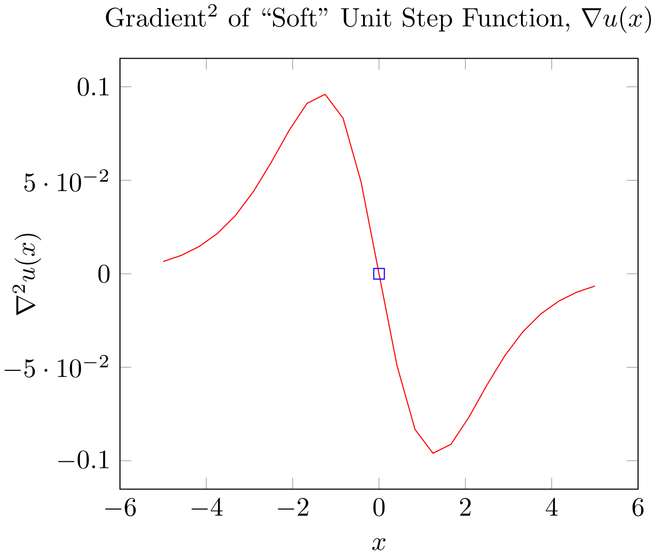
For building intuition, last time we used the following example of $u(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$:


A "Soft" Unit Step Function, $u(x)$

The gradient of this brightness across the edge, given by $\nabla u(x)$ (or $\frac{du}{dx}$ in one dimension), is then given by the following. Notice that the location of the maximum matches the inflection point in the graph above:


Gradient of "Soft" Unit Step Function, $\nabla u(x)$

As we mentioned above, we can find the location of this edge by looking at where the second derivative of brightness crosses zero, a.k.a. where $\nabla(\nabla u(x)) = \nabla^2 u(x) = 0$. Notice that the location of this zero is given by the same location as the inflection point of $u(x)$ and the maximum of $\nabla u(x)$:



For those curious, here is the math behind this specific function, assuming a sigmoid for $u(x)$:

1. $u(x) = \frac{1}{1+\exp{(-x)}}$
2. $\nabla u(x) = \frac{du}{dx} = \frac{d}{dx}\left(\frac{1}{1+\exp{-x}}\right) = \frac{\exp(-x)}{(1+\exp(-x))^2}$
3. $\nabla^2 u(x) = \frac{d^2u}{dx^2} = \frac{d}{dx}(\frac{\exp(-x)}{(1+\exp(-x))^2}) = \frac{-\exp(-x)(1+\exp(-x))^2+2\exp(-x)(1+\exp(-x))\exp(-x)}{(1+\exp(-x))^4}$

11.2.2 More on "Stencils"/Computational Molecules

Recall that we can use finite differences [1] in the forms of "stencils" or computational molecules to estimate derivatives in our images. For this patent, the authors used this framework to estimate the brightness gradient in order to find edges. For instance, partial derivative of brightness w.r.t. x can be estimated by

1. $E_x = \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix}$
2. $E_x = \frac{1}{2\epsilon}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$
3. $E_x = \frac{1}{2\epsilon}\begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix}$

Where for molecule 2, the best point for estimating derivatives lies directly in the center pixel, and for molecules 1 and 3, the best point for estimating derivatives lies halfway between the two pixels.

How do we analyze the efficacy of this approach?

1. **Taylor Series**: From previous lectures we saw that we could use averaging to reduce the error terms from 2nd order derivatives to third order derivatives. This is useful for analytically determining the error.
2. **Test functions**: We will touch more on these later, but these are helpful for testing your derivative estimates using analytical expressions, such as polynomial functions.
3. **Fourier domain**: This type of analysis is helpful for understanding how these "stencils"/molecules affect higher (spatial) frequency image content.

Note that derivative estimators can become quite complicated for high-precision estimates of the derivative, even for low-order derivatives. We can use large estimators over many pixels, but we should be mindful of the following tradeoffs:

- We will achieve better noise smoothing/suppression by including more measured values.
- Larger derivative estimators linearly (1D)/quadratically (2D) increase the amount of computation time needed.
- Features can also affect each other - e.g. a large edge detection estimator means that we can have two nearby edges affecting each other.

We can also look at some derivative estimators for higher-order derivatives. For 2nd-order derivatives, we just apply another derivative operator, which is equivalent to convolution of another derivative estimator "molecule":

$$\frac{\partial^2}{\partial x^2}(\cdot) = \frac{\partial}{\partial x}\Big(\frac{\partial(\cdot)}{\partial x}\Big) \Longleftrightarrow \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix} = \frac{1}{\epsilon^2}\begin{bmatrix}1 & -2 & 1\end{bmatrix}$$

For deriving the sign here and understanding why we have symmetry, remember that convolution "flips" one of the two filters/-operators!

Sanity Check: Let us apply this to some functions we already know the 2nd derivative of.

- $f(x) = x^2$:

  $$\begin{aligned} f(x) &= x^2 \\ f'(x) &= 2x \\ f''(x) &= 2 \end{aligned}$$

  Applying the 2nd derivative estimator above to this function:

  $$\begin{bmatrix}1 & -2 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}f(-1) = 1 & f(0) = 0 & f(1) = 1\end{bmatrix} = \frac{1}{\epsilon^2}((1*1) + (-2*0) + (1*1)) = \frac{1}{\epsilon^2}(1+0+1) = 1*2 = 2$$

  Where we note that $\epsilon = 1$ due to the pixel spacing. This is equivalent to $f''(x) = 2$.

- $f(x) = x$:

  $$\begin{aligned} f(x) &= x \\ f'(x) &= 1 \\ f''(x) &= 0 \end{aligned}$$

  Applying the 2nd derivative estimator above to this function:

  $$\begin{bmatrix}1 & -2 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}f(-1) = -1 & f(0) = 0 & f(1) = 1\end{bmatrix} = \frac{1}{\epsilon^2}((1*-1) + (-2*0) + (1*1)) = \frac{1}{\epsilon^2}(-1+0+1) = 0$$

  Where we note that $\epsilon = 1$ due to the pixel spacing. This is equivalent to $f''(x) = 0$.

- $f(x) = 1$:

  $$\begin{aligned} f(x) &= 1 \\ f'(x) &= 0 \\ f''(x) &= 0 \end{aligned}$$

  Applying the 2nd derivative estimator above to this function:

  $$\begin{bmatrix}1 & -2 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}f(-1) = 1 & f(0) = 1 & f(1) = 1\end{bmatrix} = \frac{1}{\epsilon^2}((1*1) + (-2*1) + (1*1)) = \frac{1}{\epsilon^2}(1+-2+1) = 0$$

  Where we note that $\epsilon = 1$ due to the pixel spacing. This is equivalent to $f''(x) = 0$.

In Practice: As demonstrated in the example "test functions" above, in general a good way to test an N$^{\text{th}}$ order derivative estimator is use polynomial test functions of arbitrary coefficients from order 0 up to order N. For instance, to calculate 4th order derivative estimator, test:

1. $f(x) = a$
2. $f(x) = ax + b$
3. $f(x) = ax^2 + bx + c$
4. $f(x) = ax^3 + bx^2 + cx + d$
5. $f(x) = ax^4 + bx^3 + cx^2 + dx + e$

Note: For derivative estimator operators, the weights of the "stencils"/computational molecules should add up to zero. Now that we have looked at some of these operators and modes of analysis in one dimension, let us now look at 2 dimensions.

11.2.3 Mixed Partial Derivatives in 2D

First, it is important to look at the linear, shift-invariant property of these operators, which we can express for each quality:

- **Shift-Invariant**:

$$\frac{d}{dx}(f(x+\delta)) = f'(x+\delta), \text{ for some } \delta \in \mathbb{R}$$
$$\text{Derivative of shifted function } = \text{ Derivative equivalently shifted by same amount}$$

- **Linear** :

$$\frac{d}{dx}(af_1(x)+bf_2(x)) = af_1'(x)+bf_2'(x) \text{ for some } a, b \in \mathbb{R}$$
$$\text{Derivative of scaled sum of two functions } = \text{ Scaled sum of derivatives of both functions}$$

We will exploit this **linear, shift-invariant** property frequently in machine vision. Because of this joint property, we can treat derivative operators as convolutions in 2D:

$$\frac{\partial^2}{\partial x \partial y}(\cdot) = \frac{\partial}{\partial x}\Big(\frac{\partial}{\partial y}(\cdot)\Big) \Longleftrightarrow \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}1 \\ -1\end{bmatrix} = \frac{1}{\epsilon^2}\begin{bmatrix}-1 & +1 \\ +1 & -1\end{bmatrix}$$

A few notes here:

- The second operator corresponding to E_y has been flipped in accordance with the convolution operator.
- If we project this derivative onto a "diagonal view", we find that it is simply the second derivative of x', where x' is x rotated 45 degrees counterclockwise in the 2D plane: $x' = x\cos 45 + y\cos 45 = \frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y$. In other words, in this 45-degree rotated coordinate system, $E_{x'x'} = E_{xy}$.
- **Intuition for convolution**: If convolution is a new concept for you, check out reference [2] here. Visually, convolution is equivalent to "flipping and sliding" one operator across all possible (complete and partial) overlapping configurations of the filters with one another.

11.2.4 Laplacian Estimators in 2D

The Laplacian $\nabla^2 \triangleq \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is another important estimator in machine vision, and, as we discussed last lecture, is the **lowest-order rotationally-symmetric** derivative operator. Therefore, our finite difference/computational molecule estimates should reflect this property if they are to be accurate. Two candidate estimators of this operator are:

1. **"Direct Edge"**: $\frac{1}{\epsilon^2}\begin{bmatrix}0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0\end{bmatrix}$
2. **"Indirect Edge"**: $\frac{1}{2\epsilon^2}\begin{bmatrix}1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1\end{bmatrix}$

Note that the second operator has a factor of $\frac{1}{2\epsilon^2}$ in front of it because the distance between edges is $\sqrt{2}$ rather than 1, therefore, we effectively have $\frac{1}{\epsilon'^2}$, where $\epsilon' = \sqrt{2}\epsilon$.

How do we know which of these approximations is better? We can go back to our analysis tools:

- Taylor Series
- Test functions
- Fourier analysis

Intuitively, we know that neither of these estimators will be optimal, because neither of these estimators are rotationally-symmetric. Let us combine these intelligently to achieve rotational symmetry. Adding four times the first one with one times the second:

$$4\Big(\frac{1}{\epsilon^2}\begin{bmatrix}0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0\end{bmatrix}\Big) + 1\Big(\frac{1}{2\epsilon^2}\begin{bmatrix}1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1\end{bmatrix}\Big) = \frac{1}{6\epsilon^2}\begin{bmatrix}1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1\end{bmatrix}$$

Using Taylor Series, we can show that this estimator derived from this linear combination of estimators above results in an error term that is **one derivative higher** than suing either of the individual estimators above, at the cost of more computation. Note that the sum of all the entries here is zero, as we expect for derivative estimators.

For a hexagonal grid, this is scaled by $\frac{1}{2\epsilon^2}$ and has entries of all 1s on the outer ring, and an entry of -6 in the center. An example application of a hexagonal grid - imaging black holes! Leads to $\frac{4}{\pi}$ greater efficiency.

As food for thought, what lowest-order rotationally-symmetric *nonlinear* operators?

$$\sqrt{E_{x'}^2 + E_{y'}^2} = \sqrt{E_x^2 + E_y^2} \text{ Where this is the l2 normal of the estimated brightness gradient}$$

11.2.5 Non-Maximum Suppression

Another technique leveraged in this patent was **Non-Maximum Suppression (NMS)**. Idea: Apply edge detector estimator operator everywhere - we will get a small response in most places, so what if we just threshold? This is an instance of **early decision-making**, because once we take out these points, they are no longer considered edge candidates in downstream steps.

It turns out the authors discourage thresholding, and in their work they remove all but the maximum estimated gradient (note that this is quantized at the octant level). Note that the quantized gradient direction is perpendicular to the edge. In this case, for a candidate gradient point G_0 and the adjacent pixels G_- and G_+, we must have:

$$G_0 > G_-, G_0 \geq G_+$$

This forces $-\frac{1}{2} \leq s \leq \frac{1}{2}$. Note that we have the asymmetric inequality signs to break ties arbitrarily. Next we plot the quantized profile that has been interpolated parabolically - i.e. sub-pixel interpolation.

11.2.6 Plane Position

Note that we have not yet done any thresholding. How can we improve this, given that we quantized the edge gradient direction? Could we try not quantizing the edge direction? If we have the true gradient direction, we can find the intersection of this line with the edge (at 90 degrees to the edge gradient) to find a better solution.

To find this point above (please take a look at the handwritten lecture notes for this lecture), we project from the quantized gradient direction to the actual gradient direction. This is the "plane position" component.

11.2.7 Bias Compensation

Another component of the patent focuses on the interpolation technique used for sub-pixel gradient plotting for peak finding. To find an optimal interpolation technique, we can plot s vs. s', where $s' = s|2s|^b$, where $b \in \mathbb{N}$ is a parameter that determines the relationship between s and s'.

In addition to cubic interpolation, we can also consider piecewise linear interpolation with "triangle" functions. For some different values of b:

- $b = 0 \rightarrow s' = s$
- $b = 1 \rightarrow s' = 2\text{sign}(s)s^2$
- $b = 2 \rightarrow s' = 4\text{sign}(s)s^3$

Where different interpolation methods give us different values of b.

11.2.8 Edge Transition and Defocusing Compensation

Another point of motivation: most edge detection results depend on the actual edge transition. Why are edges fuzzy (note that some degree of fuzziness is needed to prevent aliasing)? One major cause of fuzziness is "defocusing", in which the image plane and "in focus" planes are slightly off from one another. This causes a "pillbox" of radius R to be imaged (see handwritten lecture notes), rather than the ideal case of an impulse function $\delta(x, y)$. This radius is determined by:

$$R = \delta \frac{d^2}{f} \text{ (\textbf{Point Spread Function (PSF)})}$$

This pillbox image is given mathematically by:

$$\frac{1}{\pi R^2}(1 - u(r - R))$$

Where $u(\cdot)$ is the unit step function. Where f is the focal length of the lens, d is the diameter of the lens (assumed to be conic), and δ is the distance along the optical axis between the actual image plane and the "in focus" plane.

11.2.9 Multiscale

Note: We will discuss this in greater detail next lecture.

Multiscale is quite important in edge detection, because we can have edges at different scales. To draw contrasting examples, we could have an image such that:

- We have very sharp edges that transition over ≈ only 1 pixel
- We have blurry edges that transition over many pixels

11.2.10 Effect on Image Edge

Here is one possible extension not included in the edge detection patent.

We can slide a circle across a binary image - the overlapping regions inside the circle between the 1-0 edge controls how bright things appear. We can use this technique to see how accurately the algorithm plots the edge position - this allows for error calculation since we have ground truth results that we can compute using the area of the circle. Our area of interest is given by the area enclosed by the chord whose radial points intersect with the binary edge:

$$A = R^2\theta - \frac{2\sqrt{R^2 - X^2}X}{2}$$
$$\theta = \arctan\Big(\frac{\sqrt{R^2 - x^2}}{x}\Big)$$

Another way to analyze this is to compute the analytical derivatives of this brightness function:

1. $\frac{\partial E}{\partial x} = 2\sqrt{R^2 - x^2}$
2. $\frac{\partial^2 E}{\partial x^2} = \frac{-2x}{\sqrt{R^2 - x^2}}$

What can we do with this? We can use this as input into our algorithm to compute teh error and **compensate** for the degree of defocusing of the lens. In practice, there are other factors that lead to fuzzy edge profiles aside from defocusing, but this defocusing compensation helps.

11.2.11 Addressing Quantization of Gradient Directions

Here is another possible extension not included in the edge detection patent.

Recall that because spaces occurs in two sizes (pixel spacing and $\sqrt{2}$ pixel spacing), we need to sample in two ways, which can lead to slightly different error contributions. We do not want quantized gradient directions. To do this, we can just interpolate values G_-, G_0, G_+ along the true edge gradient!

Linear 1D Interpolation:

$$\tilde{f}(x) = \frac{f(a)(b - x) + f(b)(x - a)}{b - a}$$

We can also leverage more sophisticated interpolation methods, such as **cubic spline**.

Why did the authors not leverage this interpolation strategy?

- this requires the spacing of any level, i.e. not just pixel and $\sqrt{2}$ pixel spacing, but everything in between.
- Since you interpolate, you are not using measured values. Introduces some uncertainty that may be too much to achieve 1/40th pixel accuracy.

What can we do to address this? → **Project gradient onto unit circle!** This requires 2D interpolation, which can be done with methods such as bilinear or bicubic interpolation.

11.2.12 CORDIC

As we discussed in the previous lecture, CORDIC is an algorithm used to estimate vector direction by iteratively rotating a vector into a correct angle. For this patent, we are interested in using CORDIC to perform a change of coordinates from cartesian to polar:

$$(E_x, E_y) \rightarrow (E_0, E_\theta)$$

Idea: Rotate a coordinate system to make estimates using test angles iteratively. Note that we can simply compute these with square roots and arc tangents, but these can be prohibitively computationally-expensive:

$$\begin{aligned} E_0 &= \sqrt{E_x^2 + E_y^2} \\ E_\theta &= \arctan\left(\frac{E_y}{E_x}\right) \end{aligned}$$

Rather than computing these directly, it is faster to iteratively solve for the desired rotation θ by taking a sequence of iterative rotations $\{\theta_i\}_{i=1,2,\cdots}^n$. The iterative updates we have for this are, in matrix-vector form:

$$\begin{bmatrix} E_x^{(i+1)} \\ E_y^{(i+1)} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} E_x^{(i)} \\ E_y^{(i)} \end{bmatrix}$$
$$\text{Gradients at next step} = \text{ Rotation R by } \theta_i \times \text{ Gradients at current step}$$

How do we select $\{\theta_i\}_{i=1,2,\ldots}^n$? We can select progressively smaller angles. We can accept the candidate angle and invoke the iterative update above if each time the candidate angle reduces $|E_y|$ and increases $|E_x|$.

The aggregate rotation θ is simply the sum of all these accepted angles: $\theta = \sum_i \theta_i$

One potential practical issue with this approach is that it involves a significant number of multiplications. How can we avoid this? We can pick the angles carefully - i.e. if our angles are given successively by $\frac{\pi}{2}$, $\frac{\pi}{4}$, $\frac{\pi}{8}$, ..., then:

$$\frac{\sin\theta_u}{\cos\theta_i} = \frac{1}{2^i} \rightarrow \text{ rotation matrix becomes : } \frac{1}{\cos\theta_i}\begin{bmatrix} 1 & 2^{-i} \\ -2^{-i} & 1 \end{bmatrix}$$

Note that this reduces computation to 2 additions per iteration. Angle we turn through becomes successively smaller:

$$\cos\theta_i = \sqrt{1 + \frac{1}{2^{2i}}} \rightarrow R = \prod_i \cos\theta_i = \prod_i \sqrt{1 + \frac{1}{2^{2i}}} \approx 1.16 \text{ (precomputed)}$$

11.3 References

1. Finite Differences, https://en.wikipedia.org/wiki/Finite difference

2. Convolution, https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

12 Lecture 13: Object detection, Recognition and Pose Determination, PatQuick (US 7,016,539)

In this lecture, we will begin by looking at some general problems for object detection and pose estimation of objects in an image, and also look at some optimization algorithms we can use for finding optimal matches between a "master image" / template image, which is the object we look for, and this object in another image (perhaps in a scene). We then look at a patent describing some of the approaches taken to solve some of these aforementioned tasks.

12.1 Motivation & Preliminaries for Object Detection/Pose Estimation

Object detection and pose estimation builds on top of previous machine vision tasks we have covered. Some specific tasks include:

- Object detection - i.e. detect and locate object in an image

- Recognize object

- Determine pose of detected/recognized object
- Inspect object

Motivation for these approaches: In machine vision problems, we often manipulate objects in the world, and we want to know what and where these objects are in the world. In the case of these specific problems, we assume prior knowledge of the precise edge points of these objects (which, as we discussed in the two previous lectures, we know how to compute!)

12.1.1 "Blob Analysis"/"Binary Image Processing"

We can use thresholding and other algorithms such as finding a convex hull to compute elements of an object in a binary image (black/white) such as:

- Area (moment order 0)
- Perimeter
- Centroid (moment order 1)
- "Shape" (generalization of different-order moments)
- Euler number - In this case this is the number of blobs minus number of holes

A few notes about computing these different quantities of interest:

- We have seen from previous lectures that we can compute some of these elements, such as perimeter, using **Green's Theorem**. We can also accomplish this with **counting** - can be done simply by counting pixels based off of whether their pixel is a 0 or 1.
- However, the issue with these approaches is that they require **thresholding** - i.e. removing points from any further consideration early on the process and possibly without all information available; essentially removing potentially viable points too early.
- **Shape**: As introduced above, shape is often computed by computing moments of any order. Recall the definition of moments of a 2D shape:
  1. **O-order**: $\iint_D E(x,y)dxdy \rightarrow$ **Area**
  2. **1-order**: $\iint_D E(x,y)xydxdy \rightarrow$ **Centroid**
  3. **2-order**: $\iint_D E(x,y)x^2y^2dxdy \rightarrow$ **Dispersion**

     $\vdots$
  4. **k-order**: $\iint_D E(x,y)x^ky^kdxdy$
- Note that these methods are oftentimes applied to **processed**, not **raw** images.

12.1.2 Binary Template

We will discuss this more later, but this is crucial for the patent on object detection and pose estimation that we will be discussing today. A binary template is:

- A "master image" to define the object of interest that we are trying to detect/estimate the pose of
- You will have a template of an object that you can recognize the object and get the pose/attitude.

12.1.3 Normalized Correlation

This methodology also plays a key role in the patent below.

Idea: Try all possible positions/configurations of the pose space to create a match between the template and runtime image of the object. If we are interested in the squared distance between the displaced template and the image in the other object (for computational and analytic simplicity, let us only consider rotation for now), then we have the following optimization problem:

$$\min_{\delta_x,\delta_y} \iint_D (E_1(x-\delta_x, y-\delta_y) - E_2(x,y))^2 dxdy$$

Where we have denoted the two images separately as E_1 and E_2.

In addition to framing this optimization mathematically as minimizing the squared distance between the two images, we can also conceptualize this as maximizing the correlation between the displaced image and the other image:

$$\max_{\delta_x, \delta_y} \iint_D E_1(x - \delta_x, y - \delta_y) E_2(x, y) dxdy$$

We can prove mathematically that the two are equivalent. Writing out the first objective as $J(\delta_x, \delta_y)$ and expanding it:

$$\begin{aligned}
J(\delta_x, \delta_y) &= \iint_D (E(x - \delta_x, y - \delta_y) - E_2(x, y))^2 dxdy \\
&= \iint_D (E_1^2(x - \delta_x, y - \delta_y) - 2 \iint_D E_1(x - \delta_x, y - \delta_y) E_2(x, y) dxdy + \iint_D E_2^2(x, y) dxdy \\
&\implies \arg\min_{\delta_x, \delta_y} J(\delta_x, \delta_y) = \arg\max_{\delta_x, \delta_y} \iint_D E_1(x - \delta_x, y - \delta_y) E_2(x, y) dxdy
\end{aligned}$$

Since the first and third terms are constant, and since we are minimizing the negative of a scaled correlation objective, this is equivalent to maximizing the correlation of the second objective.

We can also relate this to some of the other gradient-based optimization methods we have seen using Taylor Series. Suppose δ_x, δ_y are small. Then the Taylor Series Expansion of first objective gives:

$$\iint_D (E_1(x - \delta_x, y - \delta_y) - E_2(x, y))^2 dxdy = \iint_D (E_1(x, y) - \delta_x \frac{\partial E_1}{\partial x} - \frac{\partial E_1}{\partial y} + \cdots - E_2(x, y))^2 dxdy$$

If we now consider that we are looking between consecutive frames with time period δ_t, then the optimization problem becomes (after simplifying out $E_1(x, y) - E_2(x, y) = -\delta_t \frac{\partial E}{\partial t}$):

$$\min_{\delta_x, \delta_y} \iint_D (-\delta_x E_x - \delta_y E_y - \delta_t E_t)^2 dxdy$$

Dividing through by δ_t and taking $\lim_{\delta_t \rightarrow 0}$ gives:

$$\min_{\delta_x, \delta_y} \iint_D (uE_x + vE_y + E_t)^2 dxdy$$

A few notes about the methods here and the ones above as well:

- Note that the term under the square directly above looks similar to our BCCE constraint from optical flow!
- Gradient-based methods are cheaper to compute but only function well for small deviations δ_x, δ_y.
- Correlation methods are advantageous over least-squares methods when we have scaling between the images (e.g. due to optical setting differences): $E_1(x, y) = kE_2(x, y)$ for some $k \in \mathbb{R}$.

Another question that comes up from this: How can we match at different contrast levels? We can do so with **normalized correlation**. Below, we discuss each of the elements we account for and the associated mathematical transformations:

1. **Offset**: We account for this by subtracting the mean from each brightness function:

$$\begin{aligned}
E_1^{'}(x, y) = E_1(x, y) - \bar{E}_1, \ \ \bar{E}_1 = \frac{\iint_D E_1(x, y) dxdy}{\iint_D dxdy} \\
E_2^{'}(x, y) = E_2(x, y) - \bar{E}_2, \ \ \bar{E}_2 = \frac{\iint_D E_2(x, y) dxdy}{\iint_D dxdy}
\end{aligned}$$

   This removes offset from images that could be caused by changes to optical setup.

2. **Contrast**: We account for this by computing normalized correlation, which in this case is the Pearson correlation coefficient:

$$\frac{\iint_D E_1^{'}(x - \delta_x, y - \delta_y) E_2^{'}(x, y) dxdy}{\left(\sqrt{\iint_D E_1^{'}(x - \delta_x, y - \delta_y) dxdy}\right)\left(\sqrt{\iint_D E_2^{'}(x, y) dxdy}\right)} \in [-1, 1]$$

   Where a correlation coefficient of 1 denotes a perfect match, and a correlation coefficient of -1 denotes a perfectly imperfect match.

Are there any issues with this approach? If parts/whole images of objects are obscured, this will greatly affect correlation computations at these points, even with proper normalization and offsetting.

With these preliminaries set up, we are now ready to move into a case study: a patent for object detection and pose estimation using probe points and template images.

12.2 Patent 7,016,539: Method for Fast, Robust, Multidimensional Pattern Recognition

This patent aims to extend beyond our current framework since the described methodology can account for more than just translation, e.g. can account for:

- Rotation
- Scaling
- Shearing

12.2.1 Patent Overview

This patent describes a patent for determining the presence/absence of a pre-determined pattern in an image, and for determining the location of each found instance within a multidimensional space.

A diagram of the system can be found below:

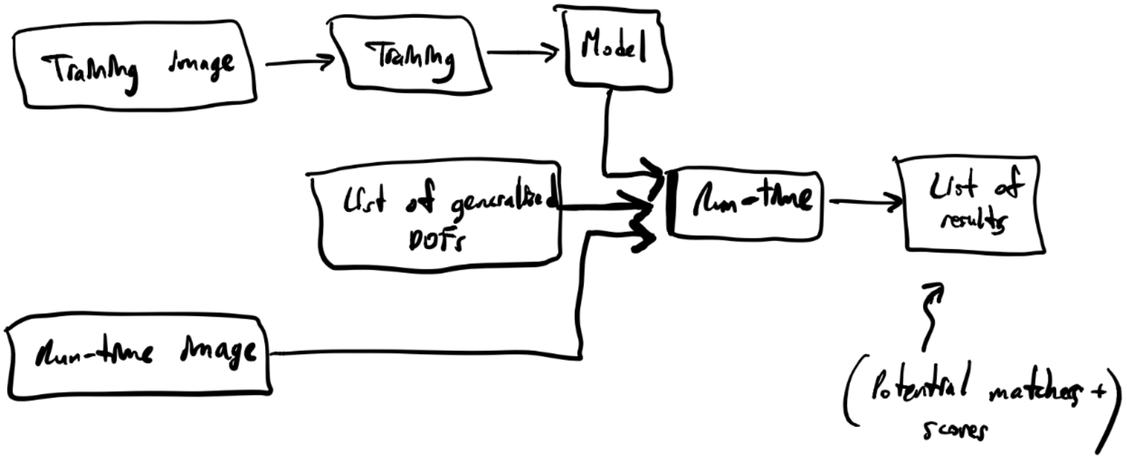


Figure 5: System diagram.

A few notes about the diagram/aggregate system:

- A **match score** is computed for each configuration, and later compared with a threshold downstream. This process leads to the construction of a high-dimensional matches surface.
- We can also see in the detailed block diagram from this patent document that we greatly leverage gradient estimation techniques from the previous patent on fast and accurate edge detection.
- For generalizability, we can run this at multiple scales/levels of resolution.

12.2.2 High-level Steps of Algorithm

1. Choose appropriate level of **granularity** (defined as "selectable size below which spatial variations in image brightness are increasingly attenuated, and below which therefore image features increasingly cannot be resolved") and store in model.
2. Process training/template image to obtain boundary points.
3. Connect neighboring boundary points that have consistent directions.
4. Organize connected boundary points into chains.

5. Remove short or weak chains.
6. Divide chains into segments of low curvature separated by conrner of high curvature.
7. Create evenly-spaced along segments and store them in model.
8. Determine pattern contrast and store in model.

A few notes about this procedure:

- To increase the efficiency of this approach, for storing the model, we store **probes** (along with granularity and contrast), not the image for which we have computed these quantities over.
- When comparing gradients between runtime and training images, project probe points onto the other image - we do not have to look at all points in image; rather, we **compare gradients** (direction and magnitude - note that magnitude is often less viable/robust to use than gradient direction) between the training and runtime images only at the probing points.
- We can also weight our probing points, either automatically or manually. Essentially, this states that some probes are more important that others when scoring functions are called on object configurations.
- This patent can also be leveraged for machine part inspection, which necessitates high degrees of consistent accuracy
- An analog of probes in other machine and computer vision tasks is the notion of **keypoints**, which are used in descriptor-based feature matching algorithms such as SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), FAST (Features from Accelerated Segment Test), and ORB (Oriented FAST and rotated BRIEF). Many of these aforementioned approaches rely on computing gradients at specific, “interesting points” as is done here, and construct features for feature matching using a **Histogram of Oriented Gradients (HoG)** [1].
- For running this framework at multiple scales/resolutions, we want to use different probes at different scales.
- For multiscale, there is a need for running fast low-pass filtering. Can do so with rapid convolutions, which we will be discussing in a later patent in this course.
- Probes “contribute evidence” individually, and are not restricted to being on the pixel grid.
- The accuracy of this approach, similar to the other framework we looked at, is limited by the degree of quantization in the search space.

12.2.3 Framework as Programming Objects

Let us also take a look at the object-oriented nature of this approach to better understand the framework we work with. One of these objects is the model:

Model: This has fields:

- **Probes**: This is the list of probe points. Note that we can also use **compiled probes**. In turn, each element in this list has fields:
  - **Position**
  - **Direction**
  - **Weight**
- **Granularity**: This is a scalar and is chosen during the training step.
- **Contrast**: This field is set to the computed contrast of the training/template pattern.

We can also look at some of the fields Generalized Degree of Freedom (**Generalized DOF**) object as well:

- **Rotation**
- **Shear** - The degree to which right angles are mapped into non-right angles.
- **Log size**
- **Log x size**
- **Log y size**

12.2.4 Other Considerations for this Framework

We should also consider how to run our **translational search**. This search should be algorithmically conducted:

- Efficiently
- At different levels of resolution
- Hexagonally, rather than on a square grid - there is a $\frac{4}{\pi}$ advantage of work done vs. resolution. Here, hexagonal peak detection is used, and to break ties, we arbitrarily set 3 of the 6 inequalities as $\geq$, and the other 3 as $>$.

What is pose?
Pose is short for position and orientation, and is usually determined with respect to a reference coordinate system. In the patent's definition, it is the **"mapping from pattern to image coordinates that represents a specific transformation and superposition of a pattern onto an image."**

Next, let us look into addressing "noise", which can cause random matches to occur. Area under $S(\theta)$ curve captures the probability of random matches, and we can compensate by calculating error and subtracting it out of the results. However, even with this compensation, we are still faced with additional noise in the result.

Instead, we can try to assign scoring weights by taking the dot product between gradient vectors: $\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_2 = \cos\theta$. But one disadvantage of this approach is that we end up quantizing pose space.

Finally, let us look at how we score the matches between template and runtime image configurations: **scoring functions**. Our options are:

- Normalized correlation (above)
- Simple peak finding
- Removal of random matches (this was our "N" factor introduced above)

12.3 References

1. Histogram of Oriented Gradients, https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

13 Lecture 14: Inspection in PatQuick, Hough Transform, Homography, Position Determination, Multi-Scale

In this lecture, we will continue our discussion of "PatQuick", the patent we discussed last time for object detection and pose estimation. We will focus on elements of this patent such as scoring functions and generalized degrees of freedom (DOF), and will use this as a segway into general linear transformations and homography. We will conclude our discussion with subsampling and Hough Transforms, which, at a high level, we can think of as a mapping from image space to parameter space.

13.1 Review of "PatQuick"

To frame our analysis and introduction of other technical machine vision concepts, let us briefly revisit the high-level ideas of "PatQuick". There were three main "objects" in this model:

- **Training/template image**. This produces a model consisting of probe points.
- **Model**, containing probe points.
- **Probe points**, which encode evidence for where to make gradient comparisons, i.e. to determine how good matches between the template image and the runtime image under the current pose configuration.

Once we have the model from the training step, we can summarize the process for generating matches as:

1. Loop over/sample from configurations of the pose space (which is determined and parameterized by our degrees of freedom), and modify the runtime image according to the current pose configuration.
2. Using the probe points of the model, compare the gradient direction (or magnitude, depending on the scoring function) to the gradient direction (magnitude) of the runtime image under the current configuration, and score using one of the scoring functions below.

3. Running this for all/all sampled pose configurations from the pose space produces a multidimensional scoring surface. We can find matches by looking for peak values in this surface.

A few more notes on this framework, before diving into the math:

- Training is beneficial here, because it allows for some degree of automated learning.
- Evidence collected from the probe points is cumulative and computed using many local operations.
- Accuracy is limited by the quantization level of the pose spanned. The non-redundant components of this pose space are:
  - 2D Translation, 2 DOF
  - Rotation, 1 DOF
  - Scaling, 1 DOF,
  - Skew, 1 DOF,
  - Aspect Ratio, 1 DOF

  Together, the space of all these components compose a **general linear transformation**, or an **affine transformation**:

  $$\begin{aligned} x' &= a_{11}x + a_{12}y + a_{13} \\ y' &= a_{21}x + a_{22}y + a_{23} \end{aligned}$$

  While having all of these options leads to a high degree of generality, it also leads to a huge number of pose configurations, even for coarse quantization. This is due to the fact that the number of configurations grows exponentially with the number of **DOF**.

13.1.1 Scoring Functions

Next, let us look at the scoring functions leveraged in this framework. Recall that there will also be random gradient matches in the background texture - we can compute this "probability" as "noise" given by N:

$$N = \frac{1}{2\pi}\int_0^{2\pi} R_{\text{dir}}(\theta)d\theta = \frac{3}{32}\ \textbf{(Using signed values)},\ \frac{6}{32}\ \textbf{(Using absolute values)}$$

Where the first value ($\frac{3}{32}$) corresponds to the probability of receiving a match if we randomly select a probe point's location, the second value corresponds to taking reverse polarity into account, and the function R_{dir} corresponds to the scoring function for the gradient direction, and takes as input the difference between two directions as two-dimensional vectors. Below are the scoring functions considered. Please take note of the following abbreviations:

- p_i denotes the two-dimensional location of the i$^{\text{th}}$ probe point after being projected onto the runtime image.
- d_i denotes the direction of the probe point in the template image.
- w_i denotes the weight of the i$^{\text{th}}$ probe point, which is typically set manually or via some procedure.
- $D(\cdot)$ is a function that takes a two-dimensional point as input, and outputs the direction of the gradient at this point.
- R_{dir} is a scoring function that takes as input the norm of the difference between two vectors representing gradient directions, and returns a scalar.
- $M(\cdot)$ is a function that computes the magnitude of the gradient in the runtime image at the point given by its two-dimensional argument.
- R_{mag} is a scoring function that takes as input the magnitude of a gradient and returns a scalar. In this case, R_{mag} saturates at a certain point, e.g. if $\lim_{x\to\infty} R(x) = K$ for some $K \in \mathbb{R}$, and for some $j \in \mathbb{R}$, $R(j) = K, R(j+\epsilon) = K \ \forall\ \epsilon \in [0,\infty)$.
- N corresponds to the "noise" term computed above from random matches.

With these terms specified, we are now ready to introduce the scoring functions:

1. **Piecewise-Linear Weighting with Direction and Normalization**:

   $$S_{1_a}(a) = \frac{\sum_i \max(w_i, 0) R_{\text{dir}}(||D(a+p_i) - d_i||_2)}{\sum_i \max(w_i, 0)}$$

   Quick note: the function $\max(0, w_i)$ is known as the Rectified Linear Unit (ReLU), and is written as $\text{ReLU}(w_i)$. This function comes up frequently in machine learning.

- Works with "compiled probes". With these "compiled probes", we only vary translation - we have already mapped pose according to the other DOFs above.
- Used in a "coarse step".

2. **Binary Weighting with Direction and Normalization**

$$S_{1_a}(a) = \frac{\sum_i (w_i > 0) R_{\text{dir}}(||D(a+p_i) - d_i||_2)}{\sum_i (w_i > 0)}$$

   Where the predicate $(w_i > 0)$ returns 1 if this is true, and 0 otherwise.

3. **"Preferred Embodiment**:

$$S(a) = \frac{\sum_i (w_i > 0)(R_{\text{dir}}(||D(a+p_i) - d_i||_2) - N)}{(1-N)\sum_i (w_i > 0)}$$

4. **Raw Weights with Gradient Magnitude Scaling and No Normalization**

$$S_2(a) = \sum_i w_i M(a+p_i) R_{\text{dir}}(||D(a+p_i) - d_i||_2)$$

   Note that this scoring function is not normalized, and is used in the fine scanning step of the algorithm.

5. **Raw Weights with Gradient Magnitude Scaling and Normalization**

$$S_3(a) = \frac{\sum_i w_i M(a+p_i) R_{\text{dir}}(||D(a+p_i) - d_i||_2)}{\sum_i w_i}$$

13.1.2 Additional System Considerations

Let us focus on a few additional system features to improve our understanding of the system as well as other principles of machine vision:

- Why do some of these approaches use normalization, but not others? **Normalization is computationally-expensive**, and approaches that avoid a normalization step typically do this to speed up computation.
- For all these scoring functions, the granularity parameter is determined by decreasing the resolution until the system no longer performs well.
- We need to ensure we get the gradient direction right. So far, with just translation, this has not been something we need to worry about. But with our generalized linear transform space of poses, we may have to account for this. Specifically:
  - Translation
  - Rotation
  - Scaling

  do not affect the gradient directions. However:

  - Shear
  - Aspect ratio

  will both affect the gradient directions. We can account for this, however, using the following process:

  1. Compute the gradient in the runtime image prior to invoking any transformations on it.
  2. Compute the isophote in the pre-transformed runtime image by rotating 90 degrees from the gradient direction using the rotation matrix given by:

$$R_{\text{G}\rightarrow\text{I}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

  3. Transform the isophote according to the generalized linear transformation above with the degrees of freedom we consider for our pose space.
  4. After computing this transformed isophote, we can find the transformed gradient by finding the direction orthogonal to the transformed isophote by rotating back 90 degrees using the rotation matrix given by:

$$R_{\text{I}\rightarrow\text{G}} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

13.1.3 Another Application of "PatQuick": Machine Inspection

Let us consider some elements of this framework that make it amenable and applicable for industrial machine part inspection:

- How do we distinguish between multiple objects (a task more generally known as multiclass object detection and classification)? We can achieve this by using **multiple models/template images, i.e. one model/template for each type of object we want to detect and find the relative pose of**.
- With this framework, we can also compute fractional matches - i.e. how well does one template match another object in the runtime image.
- We can also take an edge-based similarity perspective - we can look at the runtime image's edge and compare to edge matches achieved with the model.

13.2 Intro to Homography and Relative Poses

We will now shift gears and turn toward a topic that will also be relevant in the coming lectures on 3D. Let's revisit our perspective projection equations when we have a camera-centric coordinate system:

$$\frac{x}{f} = \frac{X_c}{Y_c}, \frac{y}{f} = \frac{Y_c}{Y_c}$$

Thus far, we have only considered camera-centric coordinate systems - that is, when the coordinates are from the point of view of the camera. But what if we seek to image points that are in a coordinate system defined by some world coordinate system that differs from the camera? Then, we can express these camera coordinates as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

$$\text{cam coord.} = \text{world2cam\_rot} \times \text{world coord.} + \text{world2cam\_trans}$$

$$\text{Where}\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \text{ is a rotation matrix in 3D.}$$

Therefore, in the general case, to find the perspective projection from world coordinates onto our image, we can combine the two previous equations, carrying out the matrix multiplication along the way:

$$\frac{x}{f} = \frac{X_c}{Z_c} = \frac{r_{11}X_W + r_{12}Y_W + r_{13}Z_W + X_0}{r_{31}X_W + r_{32}Y_W + r_{33}Z_W + Z_0}$$

$$\frac{y}{f} = \frac{Y_c}{Z_c} = \frac{r_{21}X_W + r_{22}Y_W + r_{23}Z_W + Y_0}{r_{31}X_W + r_{32}Y_W + r_{33}Z_W + Z_0}$$

Is there a way we can combine **rotation** and **translation** into a single operation? Let us consider a simple case in which the the points in our world coordinate system are coplanar in the three-dimensional plane $Z_W = 0$. Since the third column of the rotation corresponds to all zeros, we can rewrite our equation from the world coordinate frame to the camera frame as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & X_0 \\ r_{21} & r_{22} & Y_0 \\ r_{31} & r_{32} & Z_0 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} X_W \\ Y_W \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ 0 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}$$

The vector $\begin{bmatrix} X_W & Y_W & 1 \end{bmatrix}^T$ expresses our (now 2D) world coordinate system in **homogeneous coordinates**, which have a 1 entry appended to the final element.

In this case, we can fold translation and rotation into a single matrix! We call this matrix **T**, and it is called a **Homography Matrix** that encodes both rotation and translation. We will revisit this concept when we begin our discussion of 3D transformations. Note that while our rotation matrix **R** is orthogonal, this homography matrix T is not necessarily.

13.2.1 How many degrees of freedom

For determining the relative pose between camera and world frames, let us consider the number of degrees of freedom:

- 3 for translation, since we can shift in x, y, and z
- 3 for rotation, since our rotations can preserve the xz axis, xy axis, and yz axis

If we have 9 entries in the rotation matrix and 3 in the translation vector (12 unknowns total), and only 6 degrees of freedom, then how do we solve for these entries? **There is redundancy - the rotation matrix has 6 constraints from orthonormality** (3 from constraining the rows to have unit size, and 3 from having each row being orthogonal to the other).

Mathematically, these constraints appear in our **Homography matrix T** as:

$$r_{11}^2 + r_{21}^2 + r_{31}^2 = 1 \textbf{ (Unit length constraint)}$$
$$r_{12}^2 + r_{22}^2 + r_{23}^2 = 1 \textbf{ (Unit length constraint)}$$
$$r_{11}r_{12} + r_{21}r_{22} + r_{31}r_{32} = 0 \textbf{ (Orthogonal columns)}$$

A few notes here about solving for our coefficients in **T**:

- Do we need to enforce these constraints? Another option is to run least squares on the calibration data.
- We must be cognizant of the following: We only know the Homography matrix **T** up to a constant scale factor, since we are only interested in the ratio of the components of the camera coordinates for perspective projection.

13.3 Hough Transforms

Let us switch gears and talk about another way to achieve edge finding, but more generally the estimation of parameters for any parameterized surface.

Motivation: Edge and line detection for industrial machine vision. This was one of the first machine vision patents (submitted in 1960, approved in 1962). We are looking for lines in images, but our gradient-based methods may not necessarily work, e.g. due to non-contiguous lines that have “bubbles” or other discontinuities. These discontinuities can show up especially for smaller resolution levels.

Idea: The main idea of the **Hough Transform** is to intelligently map from image/surface space to parameter space for that surface. Let us walk through the mechanics of how parameter estimation works for some geometric objects.

Some notes on Hough Transforms:

- Hough transforms are often used as a subroutine in many other machine vision algorithms.
- Hough transforms actually generalize beyond edge and line detection, and extend more generally into any domain in which we map a parameterized surface in image space into parameter space in order to estimate parameters.

13.3.1 Hough Transforms with Lines

A line/edge in image space can be expressed (in two-dimensions for now, just for building intuition, but this framework is amenable for broader generalizations into higher-dimensional lines/planes): $y = mx + c$. Note that because $y = mx + c, m = \frac{y-c}{x}$ and $c = y - mx$. Therefore, this necessitates that:

- A line in image space maps to a singular point in Hough parameter space.
- A singular point in line space corresponds to a line in Hough parameter space.

To estimate the parameters of a line/accomplish edge detection, we utilize the following high-level procedure:

1. Map the points in the image to lines in Hough parameter space and compute intersections of lines.
2. Accumulate points and treat them as “evidence” using accumulator arrays.

3. Take peaks of these intersections and determine what lines they correspond to, since points in Hough parameter space define parameterizations of lines in image space. See the example below:

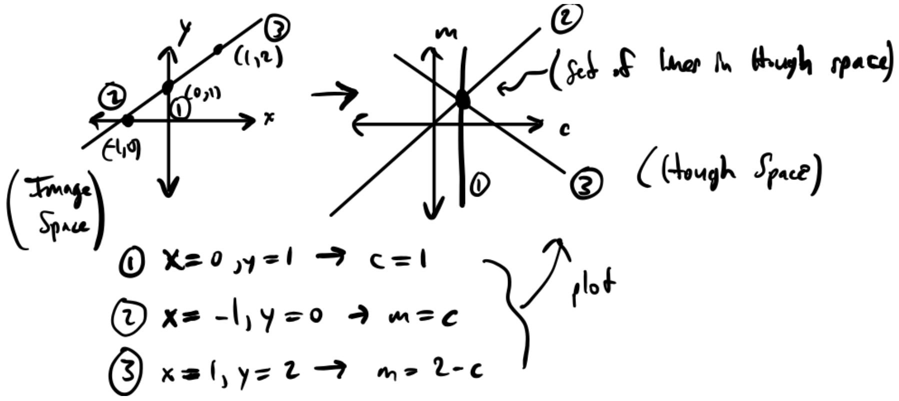


Figure 6: Example of finding parameters in Hough Space via the Hough Transform.

13.3.2 Hough Transforms with Circles

Let us look at how we can find parameters for circles with Hough transforms.

Motivating example: Localization with Long Term Evolution (LTE) Network. Some context to motivate this application further:

- LTE uses Time Division Multiplexing to send signals, a.k.a "everyone gets a slot".
- CDMA network does not use this.
- You can triangulate/localize your location based off of how long it takes to send signals to surrounding cellular towers.

We can see from the diagram below that we map our circles into Hough parameter space to compute the estimate of parameters.

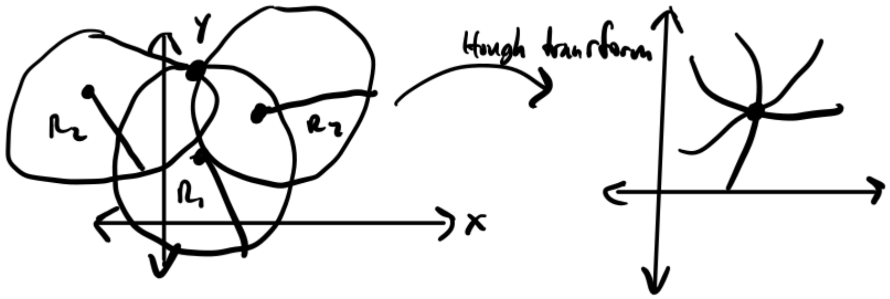


Figure 7: Example of using Hough Transforms to find the parameters of circles for LTE.

As we have seen in other problems we have studied in this class, we need to take more than one measurement. We cannot solve these problems with just one measurement, but a single measurement constrains the solution. Note that this problem assumes the radius is known.

13.3.3 Hough Transforms with Searching for Center Position and Radius

Another problem of interest is finding the center of position of a circle's radius R and its center position (x, y), which comprise the 3 dimensions in Hough parameter space. In Hough Transform space, this forms a cone that expands upward from $R_0 = 0$, where each cross-section of Z is the equation $(x^2 + y^2 = R^2)$ for the given values of x, y, and R.

Every time we find a point on the circle, we update the corresponding set of points on the cone that satisfy this equation.

The above results in many cone intersections with one another - as before, we collect evidence from these intersections, build a score surface, and compute the peak of this surface for our parameter estimates.

13.4 Sampling/Subsampling/Multiscale

Sampling is another important aspect for machine vision tasks, particularly for problems involving multiple scales, such as edge and line detection. **Sampling** is equivalent to working at **different scales**.

Why work at multiple scales?

- More efficient computation when resolution is lower, and is desirable if performance does not worsen.
- Features can be more or less salient at different resolutions, i.e. recall that edges are not as simple as step edges and often exhibit discontinuities or non-contiguous regions.

If we downsample our image by r_n along the rows and r_m along the columns, where $r_n, r_m \in (0, 1)$, then the total amount of work done (including the beginning image size) is given by the infinite geometric series:

$$\sum_{i=0}^{\infty}(r_n r_m)^i = \frac{1}{1 - r_n r_m}$$

$$\text{(Recall that } \sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \text{ for } r \in (0,1))$$

What does the total work look like for some of these values?

- $r_n = r_m = r = \frac{1}{2}$

  $$\text{work} = \frac{1}{1-r^2} = \frac{1}{1-\frac{1}{4}} = \frac{4}{3}$$

  But downsampling by $\frac{1}{2}$ each time is quite aggressive, and can lead to aliasing. Let us also look at a less aggressive sampling ratio.

- $r_n = r_m = r = \frac{1}{\sqrt{2}}$

  $$\text{work} = \frac{1}{1-r^2} = \frac{1}{1-\frac{1}{2}} = 2$$

  How do we sample in this case? This is equivalent to taking every other sample in an image when we downsample. We can do this using a **checkerboard/chess board** pattern. We can even see the selected result as a square grid if we rotate our coordinate system by 45 degrees.

  The SIFT (Scale-Invariant Feature Transform) algorithm uses this less aggressive sampling technique. SIFT is a descriptor-based feature matching algorithm for object detection using a template image.

14 Lecture 15: Alignment, recognition in PatMAx, distance field, filtering and sub-sampling (US 7,065,262)

In this lecture, we will discuss another patent on the topic of object inspection and pose estimation, known as PatMAx. We will then look at computing distance to lines as a means to perform better edge detection, and then will investigate the role of sparse convolution for multiscale systems that perform filtering.

14.1 PatMAx

Another patent we will look at for object inspection is PatMAx.

14.1.1 Overview

Some introductory notes on this:

- This framework builds off of the previous PatQuick patent.
- This framework, unlike PatQuick, does not perform quantization of the pose space, which is one key factor in enabling sub-pixel accuracy.

- PatMAx assumes we already have an approximate initial estimate of the pose.
- PatMAx relies on an iterative process for optimizing energy, and each **attraction step** improves the fit of the configuration.
- Another motivation for the name of this patent is based off of electrostatic components, namely dipoles, from Maxwell. As it turns out, however, this analogy works better with mechanical springs than with electrostatic dipoles.
- PatMAx performs an **iterative attraction process** to obtain an estimate of the pose.
- An iterative approach (e.g. gradient descent, Gauss-Newton, Levenberg-Marquadt) is taken because we likely will not have a closed-form solution in the real world. Rather than solving for a closed-form solution, we will run this iterative optimization procedure until we reach convergence.
- Relating this framework back to PatQuick, PatMAx can be run after PatQuick computes an initial pose estimate, which we can then refine using PatMAx. In fact, we can view our patent workflow as:

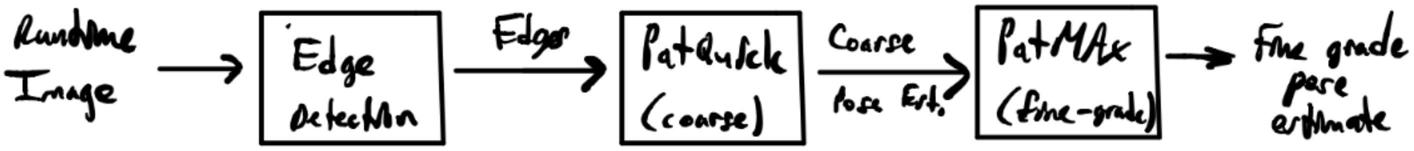


Figure 8: An overview of how the patents we have looked at for object inspection fit together.

A diagram of the system can be found here:

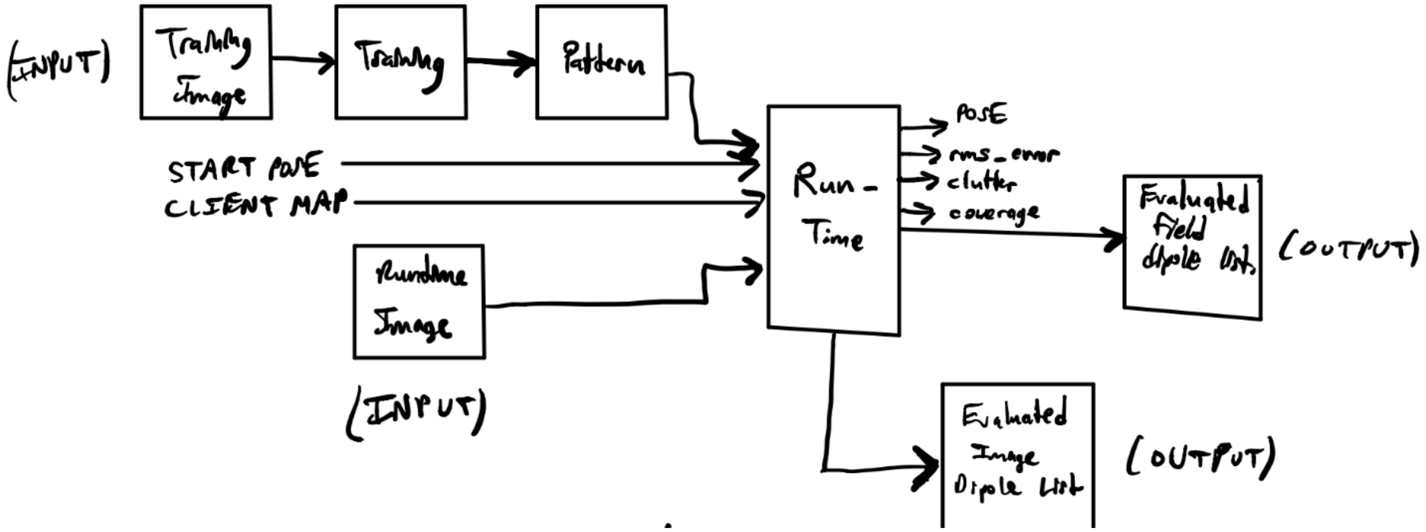


Figure 9: High-level diagram of the PatMAx system.

Now that we have a high-level overview, we are now ready to dive more into the specifics of the system.

14.1.2 Training PatMAx

: The training process can be classified as three distinct steps:

1. We begin with edge detection, which produces a **field dipole list** (essentially the probe points from the PatQuick patent framework).
2. Training also produces a **field**. We compare runtime features with template features and determine the attraction of these features between the images using this field as a vector field.*
3. We map the feature-detected runtime image's features back to the field (this is more computationally-efficient than mapping the field to the runtime image).

*For field generation, we can in turn discuss the steps needed to generate such a field:

1. Initialize
2. Seed
3. Connect
4. Chain
5. Filter
6. Segment
7. Propagate

Many of the steps outlined in this field generation process were also leveraged in the PatQuick method.

Another important aspect of training is **computing field dipoles**. A few notes on this:

- Field dipoles correspond to edge fragments.
- Field dipoles are created as a data structure of flags that provide information about proximity to other components, such as the edge.

Some other notes on this framework:

- Edge detection is largely the same procedure that we have seen in the previous patents (e.g. PatQuick). However, note that because this framework seeks to obtain highly-precise estimates accurate to the sub-pixel level, PatMAx does not use CORDIC or quantized gradient directions.
- **Field dipoles** are computed during training.
- The chaining procedure used in PatMAx is similar to the process we saw before: (i) Link chains, and then (ii) Remove short (weak) chains.
- For **initialization**, the array contains a vector field, but the vectors do not cover the entire array.

We will now explore some specific elements of this framework:

14.1.3 Estimating Other Pixels

- To estimate other pixels, we need to fill in pixels near the edge in an iterative fashion.
- To accomplish this, PatMAx uses a **distance map**, which is common in machine vision applications.
- We can compute the distance to the edge accurately with Manhattan distance, but we use Euclidean distance, which is non-trivial to compute, particularly, as we will see shortly, for corner edges.
- Intuitively, we want the system to be drawn to a "lower energy state", hence the idea of this algorithm being an energy minimization algorithm.
- Identical copies can be resolved via averaging.

14.1.4 Attraction Module

The diagram of the attraction module is given below:

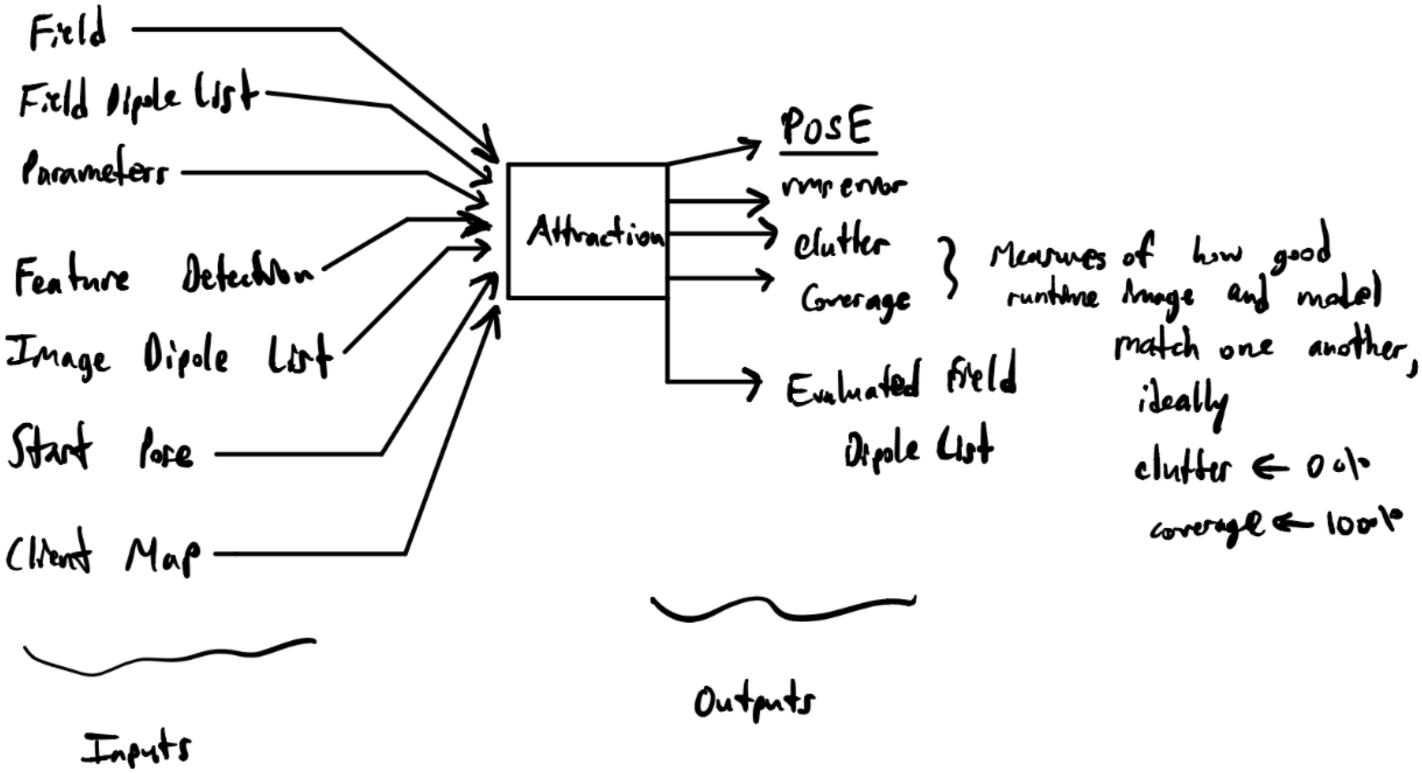


Figure 10: The Attraction module for the PatMAx system. Note that this produces a refined estimate of the pose at the output, which is one of the main goals of the PatMAx system.

Intuition with Mechanical Springs: Scaling adjustments via scaled transformations can be conceptualized as a set of mechanical springs (rather than electrostatic dipoles) that are adjusted until an optimal configuration of the degrees of freedom is found.

Solving this system is equivalent to solving a large **least squares problem**:

- Each runtime dipole has force exerted on it in a certain direction. The goal here is to use diffent movements, which comprise our Degrees of Freedom, to create a configuration of these dipoles that minimizes the overall energy/tension of this system.
- The movements/degrees of freedom that are allowed: (i) Translation in 2D (2 DOF), (ii) Rotation in 2D (1 DOF), (iii) Scaling (1 DOF). Therefore with these movements we have 4 degrees of freedom.
- A closed-form solution of this does not exist, but we can compute a solution to this least squares problem using an **upper triangular matrix accumulator array**. This array is scaled additionally by weights, and can also be used to compute image moments.
- With our movements (translation, rotation, and scaling), we have 4 DOF. With a full set of movements comprising affine linear transformations, we have 6 DOF.
- Local linearization around the operating point is also used when solving for the least squares solution to this problem.
- One computational technique the authors of this patent make use of is the use of **doubly linked lists** for the image dipoles.

14.1.5 PatMAx Claims

As we have seen, another good way to get a sense of the "abstract" of a patent is to look through its claims. The big claim of PatMAx: PatMAx is a geometric pattern-matching method used for iteratively refining the estimate of the true pose (relative to the orientation of the object in the training image) in a runtime image.

14.1.6 Comparing PatMAx to PatQuick

To better understand these frameworks (and how they potentially fit together for cascaded object inspection, let us draw some comparisons between PatQuick and PatMAx):

- PatQuick searched all pose space and does not require an initial guess - PatMAx does require an initial estimate/guess of the pose in order to produce a refined estimate.
- For PatMAx, there is repeated emphasis on avoiding techniques such as thresholding and quantization of gradient directions that have been used in the previous patents we have looked at. This makes sense, since PatMAx aims to output a more refined estimate of the pose than these other frameworks (i.e. reach sub-pixel accuracy).
- Using "dipoles" for PatMAx is misguided - using physical springs as an analog is much more physically consistent.
- For PatMAx, we use evidence collected for determining the quality of alignment, which in turn determines the quality of our refined pose estimate.
- PatMAx and PatQuick each have different methods for assigning weights.
- PatMAx is a nonlinear optimization problem, and therefore does not have a closed-form solution. PatMAx is also iterative - alignment quality and matched edges get closer with more iterations of optimization.

14.1.7 Field Generation for PatMAx

Here, we look at the field generation. For building intuition and simplifying, we will only take into account distance. See the figure below for some examples of distance fields as circles, lines, and (hardest to compute) corner edges.

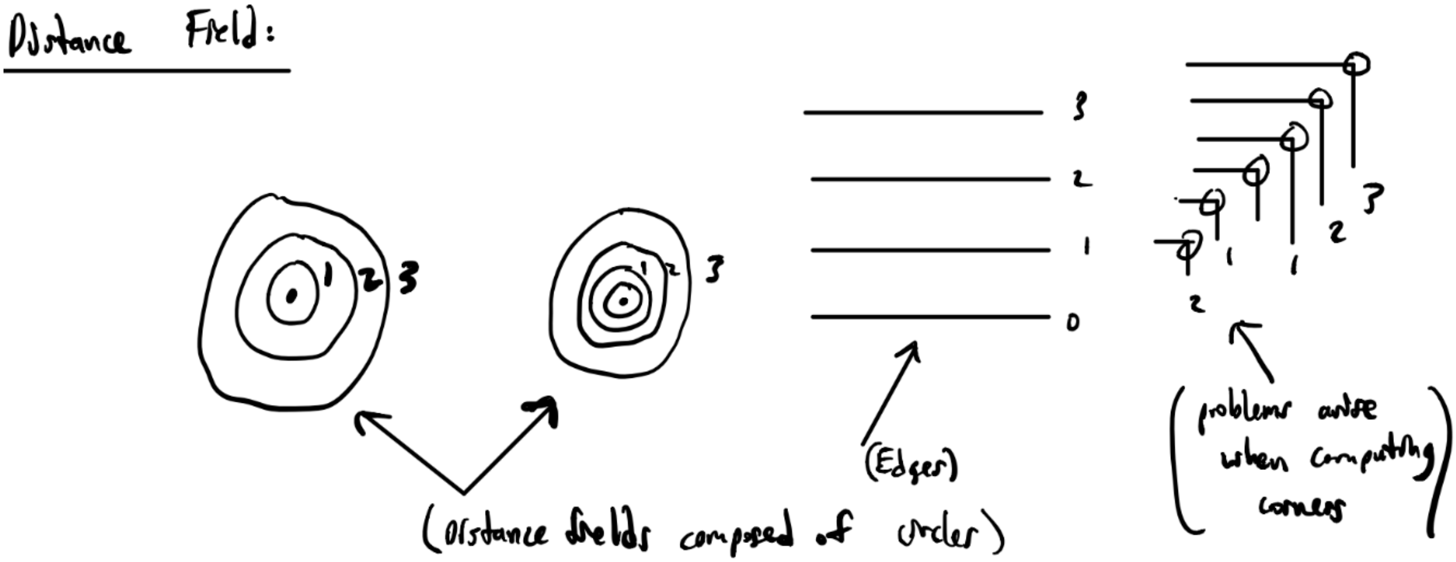

Distance Field:
1
2
3
1
2
3
3
2
1
0
1
1
2
3
2
(Edges)
(distance fields composed of circles)
(problems arise when computing corners)

Figure 11: Examples of distance fields.

A few notes about these:

- If we were working with **Manhattan** (L1-norm) distance rather than **Euclidean** (L2-norm) distance, this would be a much easier problem to solve, especially for edge corners. However, unfortunately this is not an option we have.
- We can weight the forces of **individual dipoles** in the runtime image. Weight is computed for beliefs/evidence for (i) **forces**, (ii) **torques**, and (iii) **scaling**, where $\{w_i\}_{i=1}^N$ is the set of weights:
  1. **Forces (Translation)**: $\mathbf{F} = \frac{\sum_{i=1}^N \mathbf{F}_i w_i}{\sum_{i=1}^N w_i} \in \mathbb{R}^2$
  2. **Torques (Rotation)**: $\tau = \frac{\sum_{i=1}^N w_i(\mathbf{r}_i \times \mathbf{F}_i)}{\sum_{i=1}^N w_i} \in \mathbb{R}$, where r_i is the radial vector
  3. **Scaling**: $s = \frac{\sum_{i=1}^N w_i(\mathbf{r}_i \cdot \mathbf{F}_i)}{\sum_{i=1}^N w_i} \in \mathbb{R}$ where r_i is the radial vector

  Together, these three elements composed of weighted evidence from the dipoles compose our 4 DOF.

14.2 Finding Distance to Lines

One application of performing this task is to improve the performance of edge detection systems by combining shorter edge fragments of objects into longer edge fragments.

For this procedure, we break this down into two steps:

1. Rotate the 2D cartesian coordinate system into a system that is parallel to the line of interest:

$$\begin{aligned} x' &= x\cos\theta + y\sin\theta \\ y' &= -x\sin\theta + y\cos\theta \\ \text{I.e.} \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

2. Translate the origin to the new location to match the line:

$$\begin{aligned} x'' &= x' \\ y'' &= y' - \rho \\ &= -x\sin\theta + y\cos\theta - \rho \end{aligned}$$

Together, these equations imply:

$$\begin{aligned} y'' + x\sin\theta - y\cos\theta + \rho = 0 \\ y'' + x''\sin\theta - y\cos\theta + \rho = 0 \end{aligned}$$

Which in turn has the properties:

- There are no redundant solutions
- The system is parameterized by (ρ, θ)
- There are no singularities

From here, we can use the above framework to find a line by minimizing the following objective over our parameterized degrees of freedom ρ and θ:

$$\begin{aligned} \rho^*, \theta^* &= \arg\min_{\rho,\theta} \sum_{i=1}^{N} (y_i^{''})^2 \\ &= \arg\min_{\rho,\theta} \sum_{i=1}^{N} (x_i\sin\theta - y_i\cos\theta + \rho)^2 \\ &\triangleq \arg\min_{\rho,\theta} J(\rho, \theta) \end{aligned}$$

This problem can be solved through our standard calculus approaches of finding the first-order conditions of our objective $J(\rho, \theta)$ on our degrees of freedom ρ and θ. Since we have two degrees of freedom, we have two First-Order Conditions:

1. $\frac{\partial J(\rho,\theta)}{\partial \rho} = 0$:

$$\begin{aligned} \frac{\partial}{\partial \rho}(J(\rho, \theta)) &= \frac{\partial}{\partial \rho}\Big(\sum_{i=1}^{N}(x_i\sin\theta - y_i\cos\theta + \rho)^2\Big) \\ &= 2\sum_{i=1}^{N}(x_i\sin\theta - y_i\cos\theta + \rho) = 0 \\ &= \sin\theta\Big(\sum_{i=1}^{N} x_i\Big) - \cos\theta\Big(\sum_{i=1}^{N} y_i\Big) + \Big(\sum_{i=1}^{N} \rho\Big) = 0 \\ &= N\bar{x}\sin\theta - N\bar{y}\cos\theta + N\rho = 0 \\ &= \bar{x}\sin\theta - \bar{y}\cos\theta + \rho = 0 \end{aligned}$$

(Where $\bar{x} \triangleq \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\bar{y} \triangleq \frac{1}{N}\sum_{i=1}^{N} y_i$.)

Though this does not give us the final answer, it does provide information on how our solution is constrained, i.e. the line must pass through the centroid given by the mean $(\bar{x}, \bar{y})$. Let us now look at the second FOC to combine insights from that FOC with this FOC in order to obtain our solution.

2. $\frac{\partial J(\rho,\theta)}{\partial \theta} = 0$:

   Before computing this derivative, let us move our coordinates to the centroid, i.e. subtract the mean:

$$x_i^{'} = x_i - \bar{x} \longrightarrow x_i = \bar{x} + x_i^{'}$$
$$y_i^{'} = y_i - \bar{y} \longrightarrow y_i = \bar{y} + y_i^{'}$$

   Plugging this substituted definition into our equations renders them such that the centroid cancels out. Let us now compute the second FOC:

$$\begin{aligned}
\frac{\partial}{\partial \theta}(J(\rho,\theta)) &= \frac{\partial}{\partial \theta}\Big(\sum_{i=1}^{N}(x_i \sin\theta - y_i \cos\theta + \rho)^2\Big) \\
&= 2\sum_{i=1}^{N}(x_i \sin\theta - y_i \cos\theta + \rho)(x' \cos\theta + y' \sin\theta) = 0 \\
&= \sum_{i=1}^{N} x'^2 \sin\theta\cos\theta + x'y' \sin^2\theta - x'y' \cos^2\theta - y'^2 \cos\theta\sin\theta) = 0 \\
&= \sum_{i=1}^{N}(x_i^2 - y_i^2)\sin\theta\cos\theta = \sum_{i=1}^{N} x_i y_i(\cos^2\theta - \sin^2\theta) = 0 \\
&= \frac{1}{2}\sum_{i=1}^{N}(x_i^2 - y_i^2)\sin(2\theta) = \sum_{i=1}^{N} x_i y_i \cos(2\theta) = 0 \\
&= \frac{\sin(2\theta)}{\cos(2\theta)} = \tan(2\theta) = \frac{2\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N}(x_i^2 - y_i^2)}
\end{aligned}$$

   A few notes about this:

   - In the second-to-last step, we used the two following trigonometric identities: (i) $\sin\theta\cos\theta = \sin(2\theta)$, and (ii) $\cos^2\theta - \sin^2\theta = \cos(2\theta)$.
   - Notice that we can separate the angle θ from the sums because it does not depend on the sum index and is a degree of freedom in this optimization problem.

   From here, we can solve for the optimal value of θ (which will also constrain and give us an optimal value of ρ) by taking the arctangent that returns quadrant ("atan2"):

$$\theta^* = \frac{1}{2}\arctan 2\Big(2\sum_{i=1}^{N} x_i y_i, \sum_{i=1}^{N}(x_i^2 - y_i^2)\Big)$$

Therefore, solving the FOCs gives us a closed-form least squares estimate of this line parameterized by (ρ, θ). This solution, unlike the Cartesian $y = mx + b$ fitting of a line, is independent of the chosen coordinate system, allowing for further flexibility and generalizability.

14.3 Fast Convolutions Through Sparsity

Next, we will switch gears and revisit multiscale, which is a common procedure needed in machine vision. Multiscale motivates the need of filtering methods that are computationally-agnostic to the scale or resolution being used. Since filtering is just convolution, this motivates the exploration of another patent, namely on *fast convolutions*. The patent we explore is **"Efficient Flexible Digital Filtering, US 6.457,032**.

The goal of this system is to efficiently compute filters for multiscale. For this, we assume the form of an N$^{\text{th}}$-order piecewise polynomial, i.e. a N$^{\text{th}}$-order spline.

14.3.1 System Overview

The block diagram of this system can be found below:

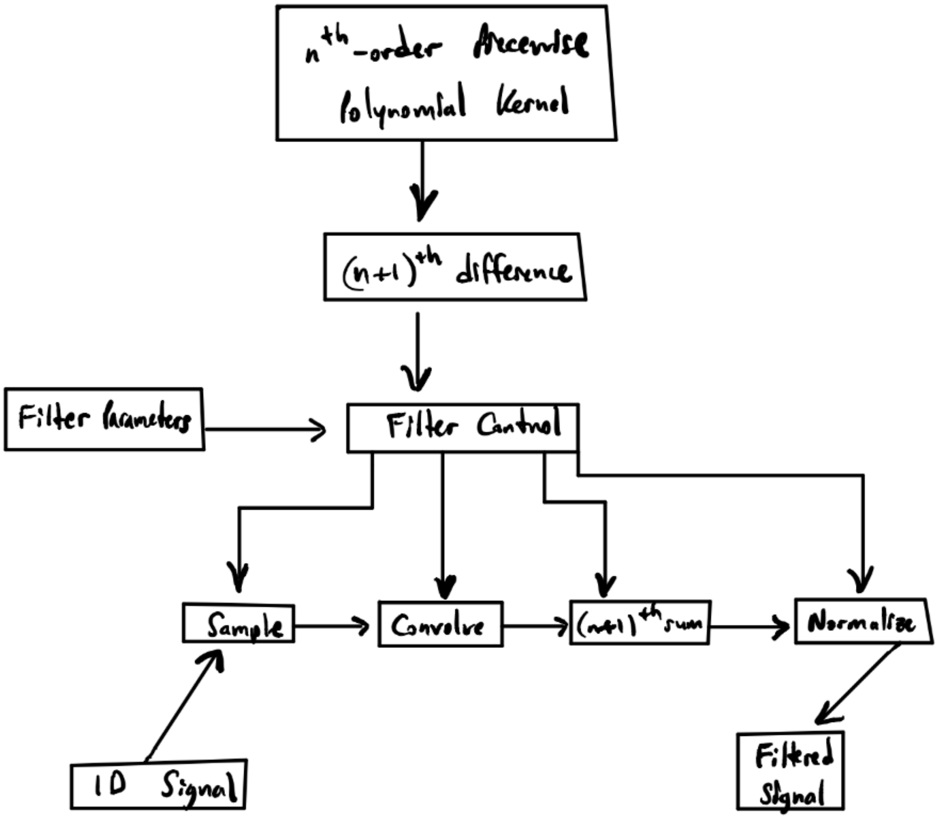


Figure 12: Block diagram of this sparse/fast convolution framework for digital filtering. Note that this can be viewed as a compression problem, in which differencing compresses the signal, and summing decompresses the signal.

A few notes on this system:

- Why is it of interest, if we have N$^{\text{th}}$-order splines as our functions, to take N$^{\text{th}}$-order differences? The reason for this is that the differences create **sparsity**, which is critical for fast and efficient convolution. Sparsity is ensured because:

  $$\frac{d^{N+1}}{dx^{N+1}}f(x) = 0 \; \forall \, x \text{ if } f(x) = \sum_{i=0}^{N} a_i x^i, a_i \in \mathbb{R} \;\; \forall \, i \in \{1, \cdots, N\}$$

  (I.e, if $f(x)$ is a order-N polynomial, then the order-(N+1) difference will be 0 for all x.

  This sparse structure makes convolutions much easier and more efficient to compute by reducing the size/cardinality of the support (we will discuss what a support is in greater detail in the next lecture, as well as how the size of a support affects computational efficiency, but effectively the support is the subset of the domain of a function that is not mapped to zero).

- Why do we apply an order-(N+1) summing operator? We apply this because we need to "invert" the effects of the order-(N+1) difference. Intuitively, this makes sense that the order-(N+1) difference and the order-(N+1) sum commute, because we are simply performing iterative rounds of subtraction and addition (respectively), which we know are commutative algebraic operations. I.e, representing differencing and summing as linear operators where their order is the same, we have:

  $$\begin{aligned}
  &\textbf{First Order}: \; \mathcal{D}\mathcal{S} = I \\
  &\textbf{Second Order}: \; \mathcal{D}\mathcal{D}\mathcal{S}\mathcal{S} = \mathcal{D}\mathcal{S}\mathcal{D}\mathcal{S} = (\mathcal{D}\mathcal{S})(\mathcal{D}\mathcal{S}) = II = I \\
  &\vdots \\
  &\textbf{Order K}: \; (\mathcal{D})^K(\mathcal{S})^K = (\mathcal{D}\mathcal{S})^K = I^K = I
  \end{aligned}$$

14.3.2 Integration and Differentiation as Convolutions

Conceptualizing these differencing/differentiation and summing/integration as linear operators that are commutative and associative, we can then extend this framework to conceptualizing these operators as convolutions:

- **Integration**: This corresponds to the convolution of our piecewise polynomial $f(x)$ with a unit step function $u(x)$.
- **Differentiation**: This corresponds to the convolution of our piecewise polynomial $f(x)$ with two scaled impulses in opposite directions: $\frac{1}{2}\delta(x+\frac{\epsilon}{2})+\frac{1}{2}\delta(x-\frac{\epsilon}{2})$.

This motivates the discussion of some of the properties of convolution this system relies on in order to achieve high performance. For operators $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$, we have that:

1. **Commutativity**: $\mathcal{A} \otimes \mathcal{B} = \mathcal{B} \otimes \mathcal{A}$
2. **Associativity**: $\mathcal{A} \otimes (\mathcal{B} \otimes \mathcal{C}) = (\mathcal{A} \otimes \mathcal{B}) \otimes \mathcal{C}$

These properties stem from the fact that in the Fourier domain, convolution is simply multiplication. Therefore, convolution obeys all the algebraic properties of multiplication.

14.3.3 Sparse Convolution as Compression

As aforementioned, another way to consider this efficient convolution system is as a compression mechanism, in which the **differencing** operation acts as a compression mechanism, and the **summing** operation acts as a decompression mechanism. We can contrast this with standard convolution in the block diagram below. This sparse convolution approach ends up being a much more efficient way to filter a signal.

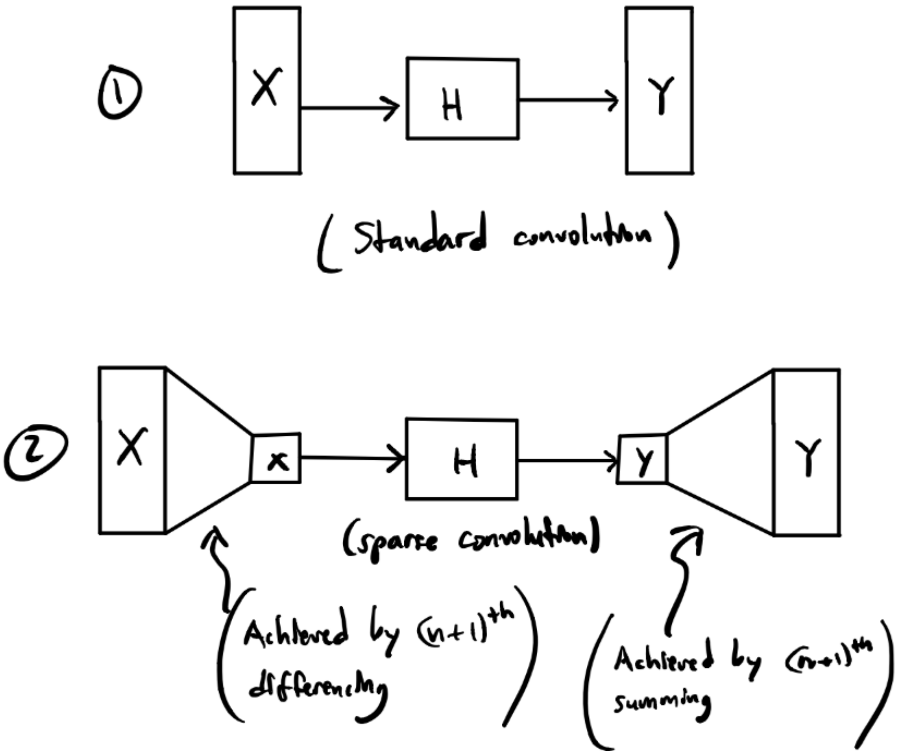

1
X
H
Y
(Standard convolution)
2
X
x
H
y
Y
(sparse convolution)
(Achieved by (n+1)th differencing)
(Achieved by (n+1)th summing)

Figure 13: Comparison of standard filtering and efficient/sparse filtering procedures, where the sparse filtering approach is illustrated as a compression problem. Here, H represents the filter, X and Y represent the uncompressed inputs and outputs, respectively, and x and y represent the compressed inputs and outputs.

14.3.4 Effects on Scaling

This fast/efficient convolution framework yields desirable results for scaling, and, as we will see shortly, for problems in which multiscale approaches are necessary. Because the only places in which we need to do work are now the locations where the piecewise polynomial segments are stitched together, the scaling can be changed to coarser and finer levels without affecting the amount of computation that is required! This improves the efficiency of multiscale approaches.

14.3.5 Filtering (For Multiscale): Anti-Aliasing

We have now reduced the amount of computation needed to compute efficient digital filtering. We now only need final ingredient for multiscale that is motivated by Shannon and Nyquist: anti-aliasing methods (filters).

Recall from Shannon/Nyquist (the Sampling Theorem) that in order to sample (for instance, when we subsample in multiscale problems) without aliasing and high-frequency artifacts, it is critical that we first remove high-frequency components from the signal we are sampling. This high-frequency component removal can be achieved with approximate low pass filtering (which we will cover in greater detail during the next lecture).

We will see in the next lecture that one way we can achieve approximate low pass filtering is by approximating a spatial sinc function (which transforms into an ideal low pass filter in the frequency domain) as a spline.

14.3.6 Extending Filtering to 2D and An Open Research Problem

This framework is built for 1D, but we note that we can extend it to 2D simply by approximating 2D convolution as a cascaded set of 1D convolutions, if we are to continue using this sparse convolution mechanism. This requires some additional run-time memory as we must store an image corresponding to the 1D convolution of the image with a 1D filter, but this allows us to continue using this efficient sparse convolution structure.

One open research problem: how can we extend this sparse convolution structure to 2D?

Finally, we will finish this lecture with a fun fact on **calcite crystals**. Calcite crystals are a type of **birefringent material**, which means that they have two indices of refraction that depend on two polarizations (one in the x-direction and one in the y-direction), and therefore reflect light into two different ways. As we will see in the next lecture, adding birefringent lenses in the analog domain can prevent aliasing affects from occurring that would otherwise be unavoidable. DSLRs have these birefringent lenses affixed to them for this specific anti-aliasing purpose.

15 Lecture 16: Fast Convolution, Low Pass Filter Approximations, Integral Images, (US 6,457,032)

15.1 Sampling and Aliasing

Sampling is a ubiquitous operation for machine vision and general signal processing. Recall that PatMax, for instance, uses sampling in its methodology. PatMax also performs an interesting operation: low-pass filtering before sampling. Why is this performed? To answer this question, let us revisit the **Nyquist Sampling Theorem**.

15.1.1 Nyquist Sampling Theorem

Before we dive into the mathematics behind this theorem, let us first build some intuition surrounding this theory.

- If we can sample a signal at a high enough frequency, we can recover the signal exactly through reconstruction.
- How is this reconstruction performed? We will convolve samples from the signal with sinc functions, and then superimpose these convolved results with one another.
- It is hard to sample from a signal with infinite support.
- What frequency do we need for this? Intuitively, to pick out how fast the signal needs to be moving, we certainly need to sample as quickly as the signal's fastest-varying component itself. But do we need to sample even faster? It turns out the answer is yes. As we will see below:

  $$f_{\text{max}} < \frac{f_{\text{sample}}}{2} \implies f_{\text{sample}} > 2f_{\text{max}}$$

  I.e. we will need to sample at more than twice the frequency of the highest-varying component of the signal.

Let us look at this graphically. What happens if we sample at the frequency of the signal?

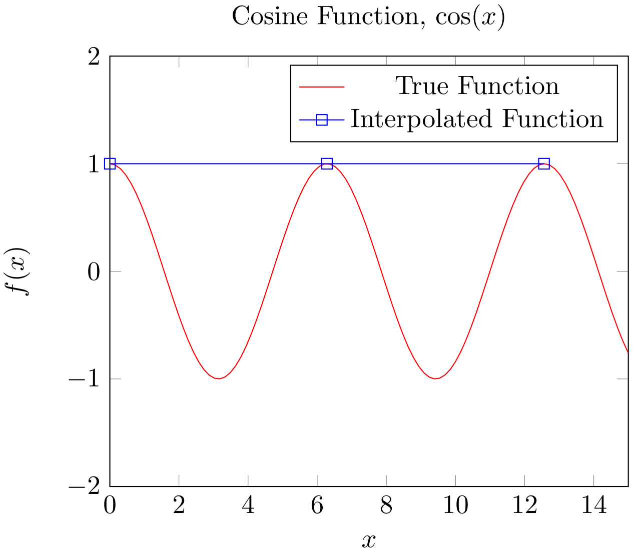


Figure 14: Sampling only once per period provides us with a constant interpolated function, from which we cannot recover the original. Therefore, we must sample at a higher frequency.

Note that this holds at points not on the peaks as well:

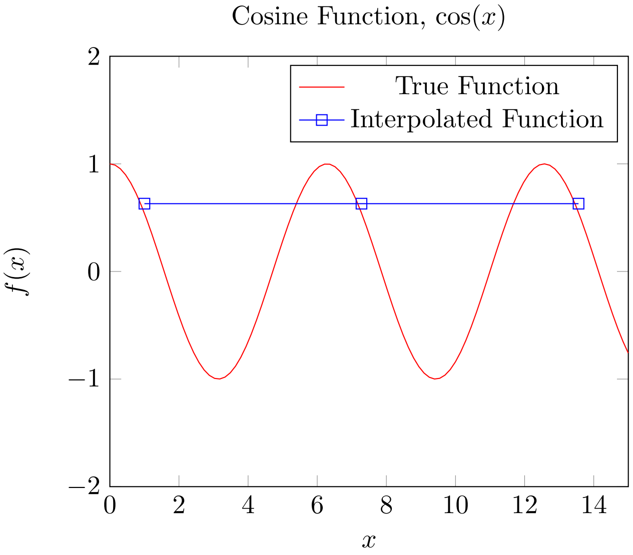


Figure 15: Sampling only once per period provides us with a constant interpolated function, from which we cannot recover the original. Therefore, we must sample at a higher frequency.

What if we sample at twice the frequency? I.e. peaks and troughs:

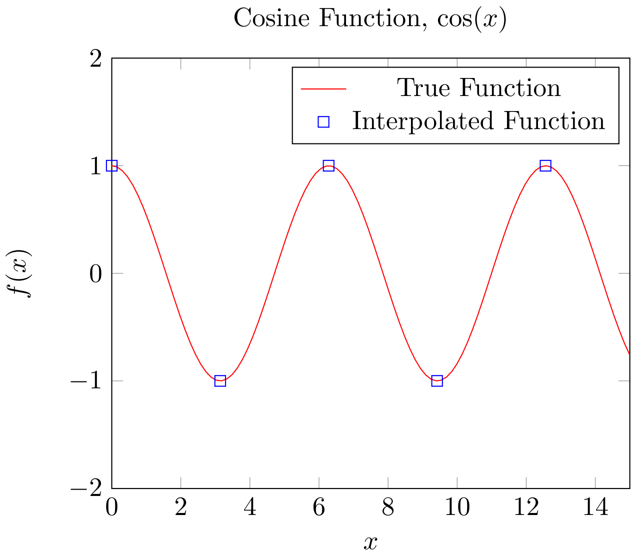


Figure 16: Sampling at twice the rate as the highest-varying component almost gets us there! This is known as the **Nyquist Rate**. It turns out we need to sample at frequencies that are strictly greater than this frequency to guarantee no aliasing - we will see why in the example below.

Is this good enough? As it turns out, the inequality for Nyquist's Sampling Theorem is there for a reason: we need to sample at **greater than** twice the frequency of the original signal in order to uniquely recover it:

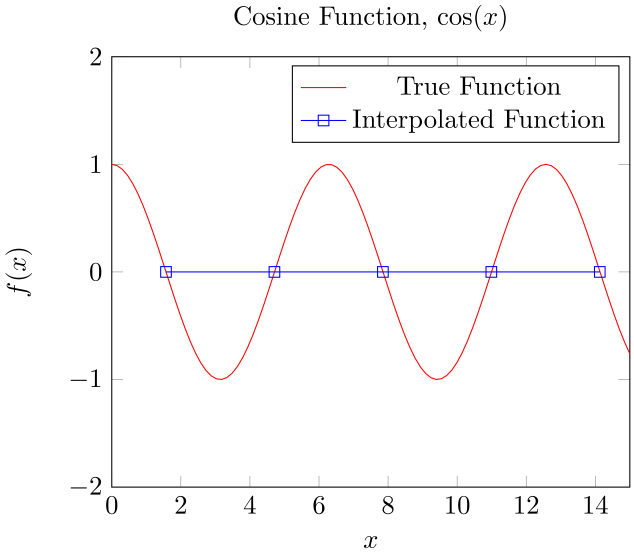


Figure 17: It turns out we need to sample at frequencies that are strictly greater than this frequency to guarantee no aliasing - we will see why in the example below.

Therefore, any rate above 2 times the highest-varying frequency component of the signal will be sufficient to completely avoid aliasing. As a review, let us next discuss aliasing.

15.1.2 Aliasing

Aliasing occurs when higher frequencies become indistinguishable from lower frequencies, and as a result they add interference and artifacts to the signal that are caused by sampling at too low of a frequency.

Suppose we have a signal given by:

$$f(x) = \cos(2\pi f_0 x)$$

And suppose we sample this signal with frequency given by $f_s = \frac{1}{\delta}$. Then our sampled signal is given by:

$$s_k = \cos(2\pi f_0 k \frac{1}{\delta}) = \cos(2\pi \frac{f_0}{f_s} k) \; \forall \; k \in \{1, 2, ...\}$$

Now let us consider what happens when we add multiples of 2π to this:

$$\begin{aligned}
s_{k-2\pi} &= \cos\Big(2\pi\frac{f_0}{f_s}k - 2\pi k\Big) \\
&= \cos\Big(2\pi\Big(\frac{f_0}{f_s} - 1\Big)k\Big) \\
&= \cos\Big(2\pi\Big(\frac{f_0 - f_s}{f_s}\Big)k\Big) \\
&= \cos\Big(2\pi\Big(\frac{f_s - f_0}{f_s}\Big)k\Big), \text{ since } \cos(x) = \cos(-x)\ \forall\ x \in \mathbb{R}
\end{aligned}$$

Another way to put this - you cannot distinguish multiples of base frequencies with the base frequencies themselves if you sample at too low a frequency, i.e. below the **Nyquist Rate**.

15.1.3 How Can We Mitigate Aliasing?

There are several strategies to mitigate aliasing effects:

- (Note) Anti-aliasing measures must be taken before sampling. After sampling occurs, information is "lost", so to speak, and the original signal cannot be recovered.
- High frequency noise suppression with (approximate) low-pass filtering. As it turns out, exact lowpass filtering is impossible due to convergence properties of Fourier Series at cutoff frequencies (a phenomenon known as the **Gibbs Phenomenon** [1].

We will touch more on these anti-aliasing techniques and strategies throughout this lecture.

15.2 Integral Image

Next, we will shift gears somewhat to discuss the notion of an **integral image**, and the critical role this technique plays in improving computational efficiency in image processing and machine vision. We will disucss this concept in both 1D and 2D.

15.2.1 Integral Images in 1D

Block averaging is a common operation in computer vision in which we take the average over a set of values across an entire vector (1D) or matrix (2D), such as an image. This involves summing and then dividing by the total number of elements, which can become prohibitively computationally-expensive, particularly if this operation is being called many times and the averages that we need to compute are very large. Is there a more computationally-efficient way to do this?

It turns out this computationally-simpler solution is through integral images. An integral image is essentially the sum of values from the first value to the i^{th} value, i.e if g_i defines the i^{th} value in 1D, then:

$$G_i \triangleq \sum_{k=1}^{i} g_k \ \forall\ i \in \{1, \cdots, K\}$$

Why is this useful? Well, rather than compute averages (normalized sums) by adding up all the pixels and then dividing, we simply need to perform a single subtraction between the integral image values (followed by a division by the number of elements we are averaging). For instance, if we wanted to calculate the average of values between i and j, then:

$$\bar{g}_{[i,j]} = \frac{1}{j-i}\sum_{k=i}^{j} g_k = \frac{1}{j-i}(G_j - G_i)$$

This greatly reduces the amortized amount of computation, because these sums only need to be computed once, when we calculate the initial values for the integral image.

15.2.2 Integral Images in 2D

Now, we can extend the notion of an integral image to 2D! Note further that integral "images" can extend beyond images! E.g. these can be done with gradients, Jacobians, Hessians, etc. One example in particular is calculating a Histogram of Gradients (HoG), which is quite useful for feature matching algorithms such as Scale-Invariant Feature Transform (SIFT). These approaches also map nicely onto GPUs, enabling for even faster computation times.

Let us now see how block averaging looks in 2D - in the diagram below, we can obtain a block average for a group of pixels in the 2D range (i, j) in x and (k, l) in y using the following formula:

$$\bar{g}_{([i,j],[k,l])} = \frac{1}{(j-i)(l-k)} \sum_{x=i}^{j} \sum_{y=k}^{l} g_{x,y}$$

But can we implement this more efficiently? We can use integral images again:

$$G_{i,j} = \sum_{k=1}^{i} \sum_{l=1}^{j} g_{k,l}$$

Referencing the figure below, this becomes:

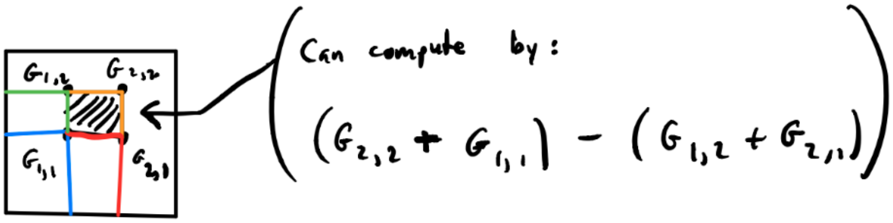


Figure 18: Block averaging using integral images in 2D. As pointed out above, block averaging also extends beyond pixels! This can be computed for other measures such as gradients (e.g. Histogram of Gradients).

Using the integral image values, the block average in the 2D range (i, j) in x and (k, l) in y becomes:

$$\bar{g}_{([i,j],[k,l])} = \frac{1}{(j-i)(l-k)} \Big((G_{j,l} + G_{i,k}) - (G_{i,l} + G_{j,k}) \Big)$$

Some comments about this:

- This analysis can be extended to higher dimensions as well! Though the integral image will take longer to compute, and the equations for computing these block averages become less intuitive, this approach generalizes to arbitrary dimensions.
- As we saw in the one-dimensional case, here we can also observe that after computing the integral image (a one-time operation that can be amortized), the computational cost for averaging each of these 2D blocks becomes **independent of the size of the block being averaged**. This stands in stark contrast to the naive implementation - here, the computational cost scales quadratically with the size of the block being averaged (or linearly in each dimension, if we take rectangular block averages).
- Why is this relevant? Recall that block averaging implements approximate lowpass filtering, which can be used as a frequency suppression mechanism to avoid aliasing when filtering.
- In other domains outside of image processing, the integral image is known as the "Summed-area Table" [2].

Since we intend to use this for approximate lowpass filtering, let us now change topics toward fourier analysis of this averaging mechanism to see how efficacious it is.

15.3 Fourier Analysis of Block Averaging

Let us consider a one-dimensional "filter" that implements an approximate lowpass filtering for mechanism through block averaging. Let us consider a function such that it takes 0 value outside of the range $(-\frac{-\delta}{2}, \frac{\delta}{2})$, and value $\frac{1}{\delta}$ inside this range:

$$h(x) = \begin{cases} \frac{1}{\delta} & x \in (-\frac{-\delta}{2}, \frac{\delta}{2}) \\ 0 & \text{o.w.} \end{cases}$$

Visually:

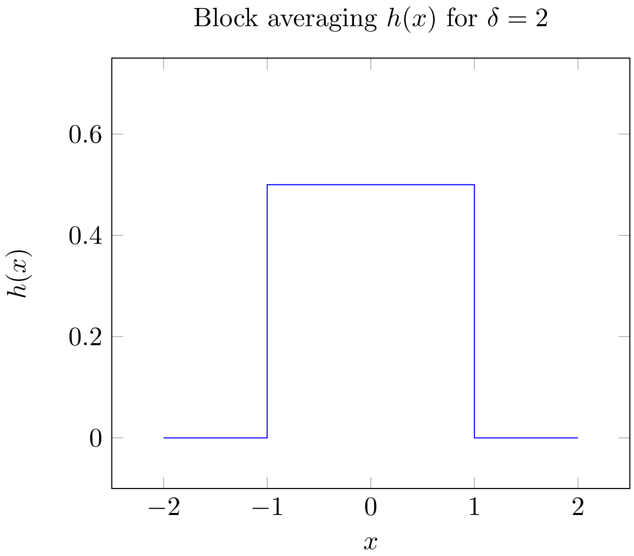


Figure 19: Example $h(x)$ for $\delta = 2$.

Let's see what this Fourier Transform looks like. Recall that the Fourier Transform (up to a constant scale factor, which varies by domain) is given by:

$$F(j\omega) = \int_{-\infty}^{\infty} f(x)e^{-j\omega x}dx$$

Where $j\omega$ corresponds to complex frequency. Substituting our expression into this transform:

$$H(j\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x}dx$$

$$= \int_{-\frac{\delta}{2}}^{\frac{\delta}{2}} \frac{1}{\delta}e^{-j\omega x}dx$$

$$= \frac{1}{\delta}\frac{1}{j\omega}[e^{-j\omega x}]_{x=-\frac{\delta}{2}}^{x=\frac{\delta}{2}}$$

$$= \frac{e^{-\frac{j\omega\delta}{2}} - e^{\frac{j\omega\delta}{2}}}{-j\omega\delta}$$

$$= \frac{\sin\left(\frac{\delta\omega}{2}\right)}{\frac{\delta\omega}{2}} \quad \textbf{(Sinc function)}$$

Where in the last equality statement we use the identity given by:

$$\sin(x) = \frac{e^{jx} - e^{-jx}}{-2j}$$

Graphically, this sinc function appears (for $\delta = 2$):

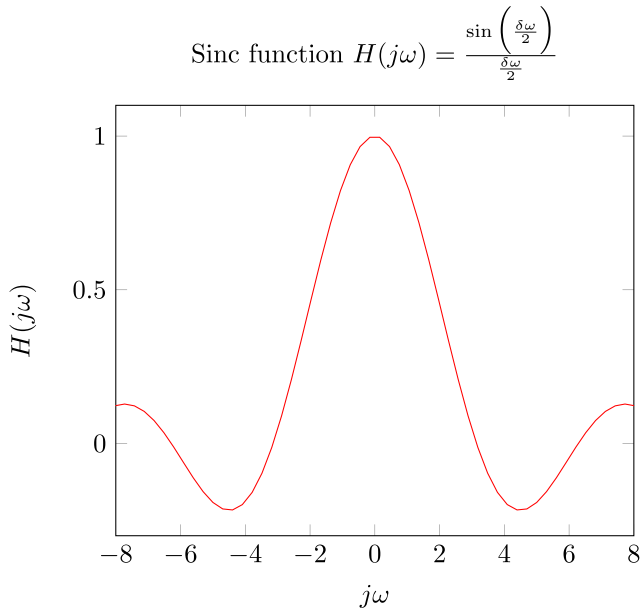


Figure 20: Example $H(j\omega)$ for $\delta = 2$. This is the Fourier Transform of our block averaging "filter".

Although sinc functions in the frequency domain help to attenuate higher frequencies, they do not make the best lowpass filters. This is the case because:

- Higher frequencies are not completely attenuated.
- The first zero is not reached quickly enough. The first zero is given by:

$$\frac{\omega_0 \delta}{2} = \frac{\pi}{2} \implies \omega_0 = \frac{\pi}{\delta}$$

Intuitively, the best lowpass filters perfectly preserve all frequencies up to the cutoff frequencies, and perfectly attenuate everything outside of the passband. Visually:

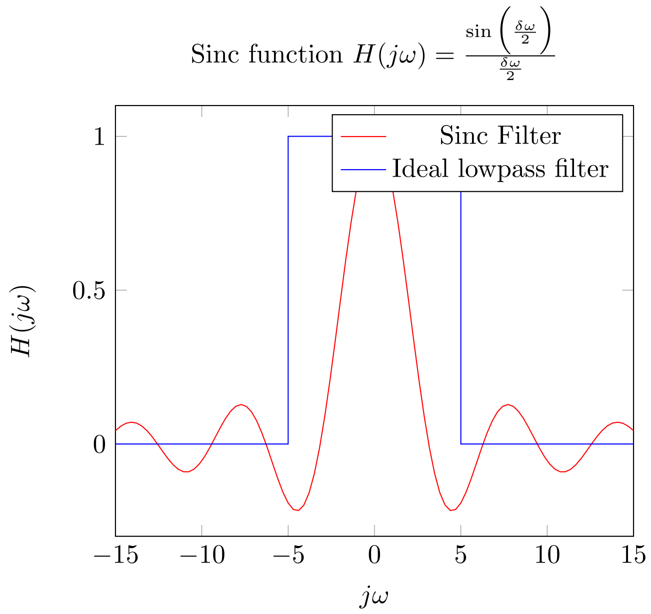


Figure 21: Frequency response comparison between our block averaging filter and an ideal lowpass filter. We also note that the "boxcar" function and the sinc function are Fourier Transform pairs!

Although sinc functions in the frequency domain help to attenuate higher frequencies, they do not make the best lowpass filters. This is the case because:

- Higher frequencies are not completely attenuated.

- The first zero is not reached quickly enough. The first zero is given by:

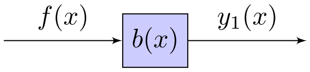
$$\frac{\omega_0 \delta}{2} = \frac{\pi}{2} \implies \omega_0 = \frac{\pi}{\delta}$$

Where else might we see this? It turns out cameras perform block average filtering because pixels have finite width over which to detect incident photons. But is this a sufficient approximate lowpass filtering technique? Unfortunately, oftentimes it is not. We will see below that we can improve with **repeated block averaging**.

15.4 Repeated Block Averaging

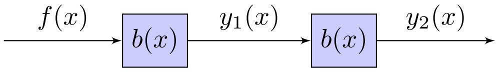
One way we can improve our ability to attenuate higher frequencies - repeated block averaging! If our "boxcar" filter given above is given by $b(x)$ (note that this was $h(x)$ above), then our previous result $y(x)$ can be written as:

$$y_1(x) = f(x) \otimes b(x)$$



What happens if we add another filter? Then, we simply add another element to our convolution:

$$y_2(x) = (f(x) \otimes b(x)) \otimes b(x) = y_1(x) \otimes b(x)$$



Adding this second filter is equivalent to convolving our signal with the convolution of two "boxcar" filters, which is a triangular filter:

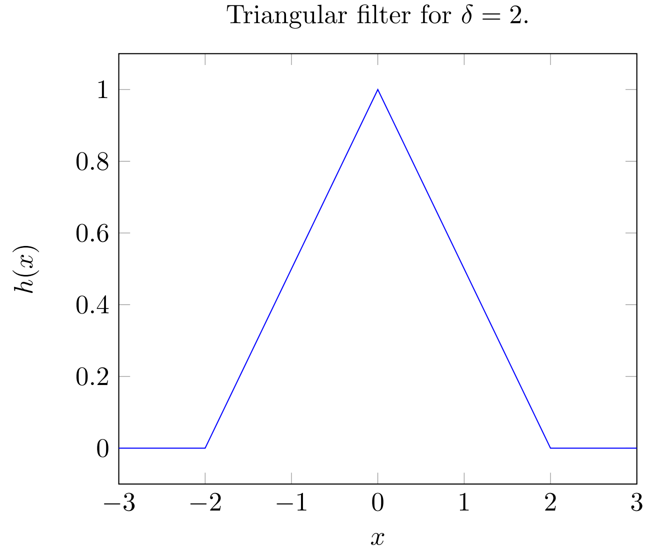


Figure 22: Example of a triangular filter resulting from the convolution of two "boxcar" filters.

Additionally, note that since convolution is associative, for the "two-stage" approximate lowpass filtering approach above, we do not need to convolve our input $f(x)$ with two "boxcar" filters - rather, we can convolve it directly with our trinagular filter $b_2(x) = b(x) \otimes b(x)$:

$$\begin{aligned} y_2(x) &= (f(x) \otimes b(x)) \otimes b(x) \\ &= f(x) \otimes (b(x) \otimes b(x)) \\ &= f(x) \otimes b_2(x) \end{aligned}$$

Let us now take a brief aside to list out how discontinuities affect Fourier Transforms in the frequency domains:

- **Delta Function**: $\delta(x) \overset{\mathcal{F}}{\leftrightarrow} 1$

  Intuition: Convolving a function with a delta function does not affect the transform, since this convolution simply produces the function.

- **Unit Step Function**: $u(x) \overset{\mathcal{F}}{\longleftrightarrow} \frac{1}{j\omega}$

  Intuition: Convolving a function with a step function produces a degree of averaging, reducing the high frequency components and therefore weighting them less heavily in the transform domain.

- **Ramp Function**: $r(x) \overset{\mathcal{F}}{\longleftrightarrow} -\frac{1}{\omega^2}$

  Intuition: Convolving a function with a ramp function produces a degree of averaging, reducing the high frequency components and therefore weighting them less heavily in the transform domain. **Derivative**: $\frac{d}{dx}f(x) \overset{\mathcal{F}}{\longleftrightarrow} j\omega F(j\omega)$

  Intuition: Since taking derivatives will increase the sharpness of our functions, and perhaps even create discontinuities, a derivative in the spatial domain corresponds to multiplying by $j\omega$ in the frequency domain.

As we can see from above, the more "averaging" effects we have, the more the high-frequency components of the signal will be filtered out. Conversely, when we take derivatives and create discontinuities in our spatial domain signal, this increases high frequency components of the signal because it introduces more variation.

To understand how we can use repeated block averaging in the Fourier domain, please recall the following special properties of Fourier Transforms:

1. **Convolution in the spatial domain corresponds to multiplication in the frequency domain**, i.e. for all $f(x), g(x), h(x)$ with corresponding Fourier Transforms $F(j\omega), G(j\omega), H(j\omega)$, we have:

$$h(x) = f(x) \otimes g(x) \overset{\mathcal{F}}{\longleftrightarrow} H(j\omega) = F(j\omega)G(j\omega)$$

2. **Multiplication in the spatial domain corresponds to convolution in the frequency domain**, i.e. for all $f(x), g(x), h(x)$ with corresponding Fourier Transforms $F(j\omega), G(j\omega), H(j\omega)$, we have:

$$h(x) = f(x)g(x) \overset{\mathcal{F}}{\longleftrightarrow} H(j\omega) = F(j\omega) \otimes G(j\omega)$$

For block averaging, we can use the first of these properties to understand what is happening in the frequency domain:

$$y_2(x) = f(x) \otimes (b(x) \otimes b(x)) \overset{\mathcal{F}}{\longleftrightarrow} Y(j\omega) = F(j\omega)(B(j\omega)^2)$$

This operation is equivalent to a sinc^2 function in the spatial domain:

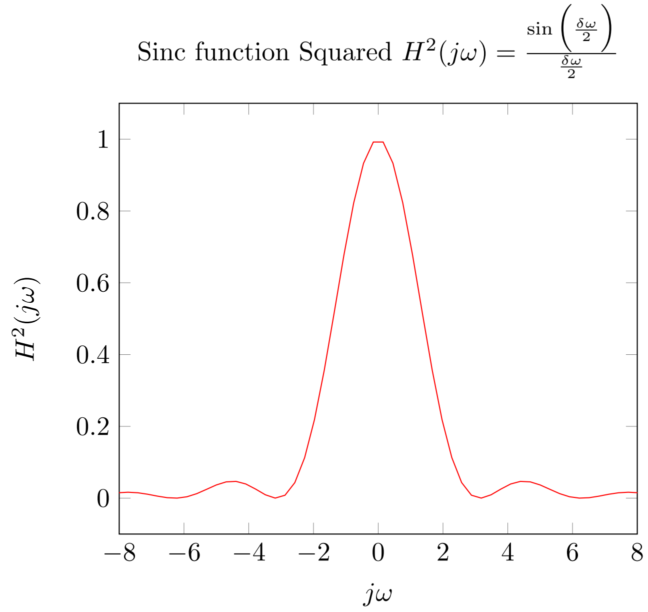


Figure 23: Example $H^2(j\omega)$ for $\delta = 2$. This is the Fourier Transform of our block averaging "filter" convolved with itself in the spatial domain.

.

This is not perfect, but it is an improvement. In fact, the frequencies with this filter drop off with magnitude $\left(\frac{1}{\omega}\right)^2$. What happens if we continue to repeat this process with more block averaging filters? It turns out that for N "boxcar" filters that we use, the magnitude will drop off as $\left(\frac{1}{\omega}\right)^N$. Note too, that we do not want to go "too far" in this direction, because this repeated block averaging process will also begin to attenuate frequencies in the passband of the signal.

15.4.1 Warping Effects and Numerical Fourier Transforms: FFT and DFT

Two main types of numerical transforms we briefly discuss are the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT). The FFT is an extension of the DFT that relies on using a "divide and conquer" approach to reduce the computational runtime from $f(N) \in O(N^2)$ to $f(N) \in O(N \log N)$ [3].

Mathematically, the DFT is given as a transform that transforms a sequence of N complex numbers $\{x_n\}_{n=1}^N$ into another sequence of N complex numbers $\{X_k\}_{k=1}^N$ [4]. The transform for the K^{th} value of this output sequence is given in closed-form as:

$$X_k = \sum_{i=1}^{N} x_n e^{-j\frac{2\pi}{N}kn}$$

And the inverse transform for the n^{th} value of this input sequence is given as:

$$x_n = \sum_{k=1}^{N} X_k e^{j\frac{2\pi}{N}kn}$$

One aspect of these transforms to be especially mindful of is that they introduce a wrapping effect, since transform values are spread out over 2π intervals. This means that the waveforms produced by these transforms, in both the spatial (if we take the inverse transform) and frequency domains may be repeated - this repeating can introduce undesirable discontinuities, such as those seen in the graph below:

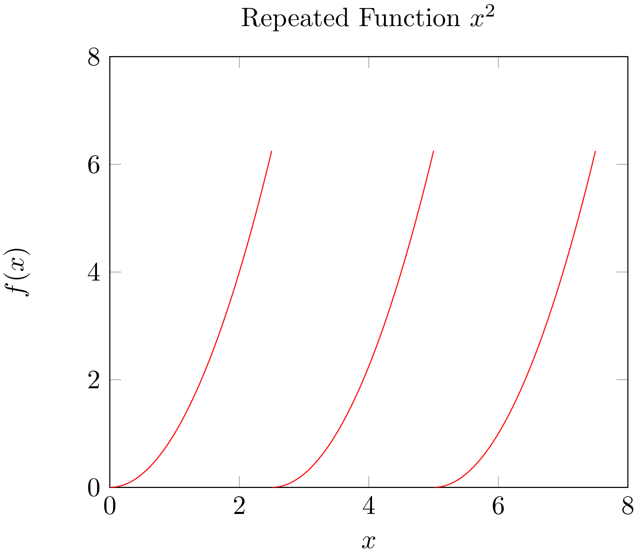


Figure 24: Example of a repeated waveform that we encounter when looking at DFTs and FFTs.

Fun fact: It used to be thought that natural images had a power spectrum (power in the frequency domain) that falls off as $\frac{1}{\omega}$. It turns out that this was actually caused by warping effects introduced by discrete transforms.

This begs the question - how can we mitigate these warping effects? Some methods include:

- **Apodizing**: This corresponds to multiplying your signal by a waveform, e.g. Hamming's Window, which takes the form akin to a Gaussian, or an inverted cosine.
- **Mirroring**: Another method to mitigate these warping effects is through waveform mirroring - this ensures continuity at points where discontinuties occurred:

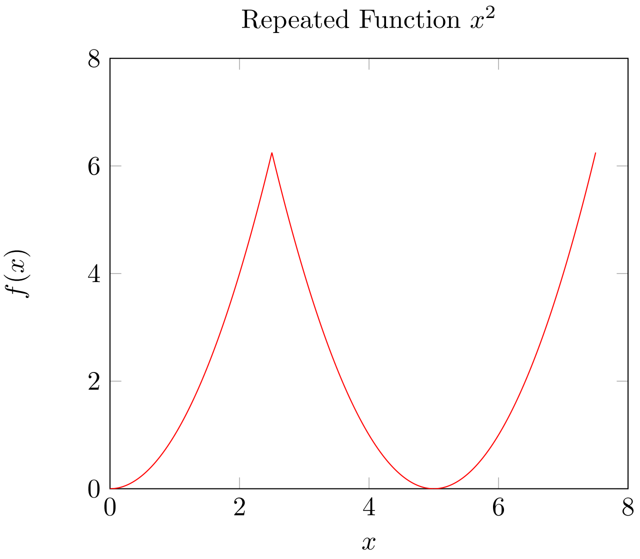


Figure 25: Example of a mirrored waveform that we can use to counter and mitigate the discontinuity effects of warping from transforms such as the DFT and FFT.

With this approach, the power spectrum of these signals falls off at $\frac{1}{\omega^2}$, rather than $\frac{1}{\omega}$.

- **Infinitely Wide Signal**: Finally, a less practical, but conceptual helpful method is simply to take an "infinitely wide signal".

Let us now switch gears to talk more about the unit impulse and convolution.

15.5 Impulses and Convolution

In this section, we will review impulse functions, and discuss how they relate to many properties with convolution. We will begin by reviewing delta functions and their properties.

15.5.1 Properties of Delta Functions

Recall that the Dirac delta function $\delta(x)$, or impulse function, is defined according to the two properties:

1. **Unit Area**: $\int_{-\infty}^{\infty} \delta(x)dx = 1$
2. **Sifting Property**: $f(x_0) = \int_{-\infty}^{\infty} f(x)\delta(x - x_0)dx$

Another way to conceptualize delta functions is through **probabilistic distributions**. We can use Gaussians (one of the only distributions to have a Fourier Transform). Recall that the (zero-mean) Gaussian Fourier Transform pair is given by:

$$\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x}{2\sigma^2}} \overset{\mathcal{F}}{\longleftrightarrow} e^{-\frac{\omega^2\sigma^2}{2}}$$

An impulse can be conceptualized as the limit in which the variance of this Gaussian distribution σ^2 goes to 0, which corresponds to a Fourier Transform of 1 for all frequencies (which is the Fourier Transform of a delta function).

Another way to consider impulses is that they are the limit of "boxcar" functions as their width goes to zero.

Let us next generalize from a single impulse function to combinations of these functions.

15.5.2 Combinations of Impulses

When are combinations of impulses helpful? it turns out that one particular combination can be used for approximating the derivative using our prior work on finite differences:

$$h(x) = \frac{1}{\epsilon}\Big(\delta(x + \frac{\epsilon}{2}) - \delta(x - \frac{\epsilon}{2})\Big) \text{ for some } \epsilon > 0$$

Correlating (*note that this is not convolution - if we were to use convolution, this derivative would be flipped) this combination of impulse "filter" with an arbitrary function $f(x)$, we compute a first-order approximation of the derivative:

$$\begin{aligned} f'(x) &\approx \int_{-\infty}^{\infty} f(x)h(x)dx \\ &= \int_{-\infty}^{\infty} \Big(\frac{1}{\epsilon}\Big(\delta(x+\frac{\epsilon}{2}) - \delta(x-\frac{\epsilon}{2})\Big)\Big)dx \end{aligned}$$

Therefore, combinations of impulses can be used to represent the same behavior as the "computational molecules" we identified before. It turns out that there is a close connection between **linear, shift-invariant** operators and **derivative operators**.

With impulses motivated, let us now formally review convolution.

15.5.3 Convolution Review

Recall from the example above that the convolution operation is simply the result of the correlation operation flipped. Mathematically, the convolution of functions $f(x)$ and $h(x)$ is given by the following commutative operation:

$$\begin{aligned} g(x) &= f(x) \otimes h(x) \\ &= \int_{-\infty}^{\infty} f(\xi)h(x-\xi)d\xi \\ &= h(x) \otimes g(x) \\ &= \int_{-\infty}^{\infty} h(\xi)f(x-\xi)d\xi \end{aligned}$$

15.5.4 Analog Filtering with Birefringent Lenses

Why filter in the analog domain? Returning to our original problem of mitigating aliasing artifacts through high-frequency suppression, unfortunately, if we wait to digitally filter out high frequencies after the image has already been taken by a camera or sensor, then we have already caused aliasing.

One way to achieve this analog filtering is through **Birefringent Lenses**. Here, we essentially take two "shifted" images by convolving the image with a symmetric combination of offset delta functions, given mathematically by:

$$h(x) = \frac{1}{2}\delta(x+\frac{\epsilon}{2}) + \frac{1}{2}\delta(x-\frac{\epsilon}{2}) \text{ for some } \epsilon > 0$$

Let us look at the Fourier Transform of this filter, noting the following Fourier Transform pair:

$$\delta(x-x_0) \overset{\mathcal{F}}{\longleftrightarrow} e^{-j\omega x_0}$$

With this we can then express the Fourier Transform of this filter as:

$$\begin{aligned} F(j\omega) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} h(x)e^{-j\omega x}dx \\ &= \frac{1}{2}\Big(e^{-\frac{j\omega\epsilon}{2}} + e^{\frac{j\omega\epsilon}{2}}\Big) \\ &= \cos\Big(\frac{\omega\epsilon}{2}\Big) \end{aligned}$$

With this framework, the first zero to appear here occurs at $\omega_0 = \frac{\pi}{\epsilon}$. A few notes about these filters, and how they relate to high-frequency noise suppression.

- When these birefringent lenses are cascaded with a **block averaging** filter, this results in a combined filtering scheme in which the zeros of the frequency responses of these filters cancel out most of the high-frequency noise.
- In the 2D case, we will have 2 birefringent filters, one for the x-direction and one for the y-direction. Physically, these are rotated 90 degrees off from one another, just as they are for a 2D cartesian coordinate system.

- High-performance lowpass filtering requires a large **support** (see definition of this below if needed) - the computational costs grow linearly with the size of the support in 1D, and quadratically with the size of the support in 2D. The support of a function is defined as the set where $f(\cdot)$ is nonzero [5]:

$$\text{supp}(f) = \{x : f(x) \neq 0, x \in \mathbb{R}\}$$

- Therefore, one way to reduce the computatonal costs of a filtering system is to reduce the size/cardinality of the support $|\text{supp}(f)|$ - in some sense to encourage sparsity. Fortunately, this does not necessarily mean looking over a narrower range, but instead just considering less points overall.

15.5.5 Derivatives and Integrals as Convolution Operators and FT Pairs

As we have seen before, convolving a function with a unit step function results in integrating the given function. Let us verify this below:

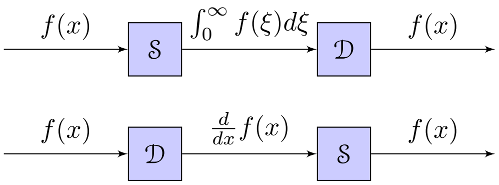
$$\begin{aligned} f(x) \otimes u(x) &= \int_{-\infty}^{\infty} f(\xi)u(x-\xi)d\xi \\ &= \int_{-\infty}^{\infty} f(x-\xi)u(\xi)d\xi \\ &= \int_{0}^{\infty} f(\xi)d\xi \end{aligned}$$

Therefore, we can represent integral and derivative operators as Fourier Transform pairs too, denoted $\mathcal{S}$ for integration and $\mathcal{D}$ for derivative:

- $\mathcal{S} \overset{\mathcal{F}}{\leftrightarrow} \frac{1}{j\omega}$
- $\mathcal{D} \overset{\mathcal{F}}{\leftrightarrow} j\omega$

Niote that we can verify this by showing that convolving these filter operators corresponds to multiplying these transforms in frequency space, which results in no effect when cascaded together:

$$(f(x) \otimes \mathcal{D}) \otimes \mathcal{S} = f(x) \otimes (\mathcal{D} \otimes \mathcal{S}) \overset{\mathcal{F}}{\leftrightarrow} F(j\omega)\Big(j\omega\frac{1}{j\omega}\Big) = F(j\omega) \overset{\mathcal{F}}{\leftrightarrow} f(x)$$



Can we extend this to higher-order derivatives? It turns out we can. One example is the convolution of two derivative operators, which becomes:

$$h(x) = \delta(x+\frac{\epsilon}{2}) - 2\delta(x) + \delta(x-\frac{\epsilon}{2}) = \mathcal{D} \otimes \mathcal{D} \overset{\mathcal{F}}{\leftrightarrow} H(j\omega) = D(j\omega)^2 = (j\omega)^2 = -\omega^2 \text{ (Recall that } j^2 = -1)$$

In general, this holds. Note that the number of integral operators $\mathcal{S}$ must be equal to the number of derivative operators $\mathcal{D}$, e.g. for K order:

$$\Big(\otimes_{i=1}^{K} \mathcal{S}\Big) \otimes \Big(\Big(\otimes_{i=1}^{K} \mathcal{D}\Big) \otimes f(x)\Big)$$

15.5.6 Interpolation and Convolution

A few notes about interpolation methods:

- These methods work well computationally when there is sparsity in the operators we with - for instance, cubic or linear interpolation. Photoshop and other photo-editing software frameworks iuse this interpolation techniques. Performance deteriorates when we switch from **cubic interpolation** to **Nearest Neighbor interpolation**.
- The **bicubic spline** is a popular interpolation technique. It was invented by IBM and was used in early satellite image processing. However, it requires many neighboring pixels, as it is composed of 7 points to interpolate over, so it is less computationally-efficient.

- Recall that one key element of computational efficiency we pursue is to use integral images for block averaging, which is much more efficient than computing naive sums, especially if (1) This block averaging procedure is repeated many times (the amortized cost of computing the integral image is lessened) and (2) This process is used in higher dimensions.
- **Linear interpolation** can be conceptualized as connecting points together using straight lines between points. This corresponds to piecewise-linear segments, or, convolution with a triangle filter, which is simply the convolution of two "boxcar filters":

  $$f(x) = f(1)x + f(0)(1-x)$$

  Unfortunately, one "not-so-great" property of convolving with triangular filters for interpolation is that the noise in the interpolated result varies depending on how far away we are from the sampled noise.
- **Nearest Neighbor** techniques can also be viewed through a convolutional lens - since this method produces piecewise-constant interpolation, this is equivalent to convolving our sampled points with a "boxcar" filter!

15.5.7 Rotationally-Symmetric Lowpass Filter in 2D

Where u and v are the spatial frequencies of x and y, i.e. the 2D Fourier Transform and Inverse Fourier Transform then take the forms of:

$$F(u,v) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j(ux+vy)} dx dy$$
$$f(x,y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{j(ux+vy)} du dv$$

The inverse transform of this can be thought of as a **sinc** function in polar coordinates:

$$f(\rho, \theta) = \frac{B^2}{2\pi} \frac{J_1(\rho B)}{\rho B}$$

A few notes about this inverse transform function:

- This is the point spread function of a microscope.
- $J_1(\cdot)$ is a 1st-order Bessel function.
- This relates to our **defocusing** problem that we encountered before.
- In the case of defocusing, we can use the "symmetry" property of the Fourier Transform to deduce that if we have a circular point spread function resulting from defocusing of the lens, then we will have a Bessel function in the frequency/Fourier domain.
- Though a pointspread function is a "pillbox" in the ideal case, in practice this is not perfect due to artifacts such as lens aberrations.

15.6 References

1. Gibbs Phenomenon, https://en.wikipedia.org/wiki/Gibbs_phenomenon
2. Summed-area Table, https://en.wikipedia.org/wiki/Summed-area_table
3. Fast Fourier Transform, https://en.wikipedia.org/wiki/Fast_Fourier_transform
4. Discrete Fourier Transform, https://en.wikipedia.org/wiki/Discrete_Fourier_transform
5. Support, https://en.wikipedia.org/wiki/Support_(mathematics)

16 Lecture 17: Photogrammetry, Orientation, Axes of Inertia, Symmetry, Absolute, Relative, Interior, and Exterior Orientation

This lecture marks a transition in what we have covered so far in the course from lower-level machine vision to higher-level problems, beginning with photogrammetry.

Photogrammetry: "Measurements from Imagery", for example, map-making.

16.1 Photogrammetry Problems: An Overview

Four important problems in photogrammetry that we will cover are:

- **Absolute Orientation** $3D \longleftrightarrow 3D$
- **Relative Orientation** $2D \longleftrightarrow 2D$
- **Exterior Orientation** $2D \longleftrightarrow 3D$
- **Intrinsic Orientation** $3D \longleftrightarrow 2D$

Below we discuss each of these problems at a high level. We will be discussing these problems in greater depth later in this and following lectures.

16.1.1 Absolute Orientation

We will start with covering **absolute orientation**. This problem asks about the relationship between two or more objects (cameras, points, other sensors) in 3D. Some examples of this problem include:

1. Given two 3D sensors, such as lidar (light detection and ranging) sensors, our goal is to find the **transformation**, or **pose**, between these two sensors.
2. Given one 3D sensor, such as a lidar sensor, and two objects (note that this could be two distinct objects at a single point in time, or a single object at two distinct points in time), our goal is to find the **transformation**, or **pose**, between the two objects.

16.1.2 Relative Orientation

This problem asks how we can find the 2D $\longleftrightarrow$ 2D relationship between two objects, such as cameras, points, or other sensors. This type of problem comes up frequently in machine vision, for instance, **binocular stereo**. Two high-level applications include:

1. Given two cameras/images that these cameras take, our goal is to extract 3D information by finding the relationship between two 2D images.
2. Given two cameras, our goal is to find the (relative) **transformation**, or **pose**, between the two cameras.

16.1.3 Exterior Orientation

This photogrammetry problem aims from going 2D $\longrightarrow$ 3D. One common example in robotics (and other field related to machine vision) is **localization**, in which a robotic agent must find their location/orientation on a map given 2D information from a camera (as well as, possibly, 3D laser scan measurements).

More generally, with **localization**, our goal is to find where we are and how we are oriented in space given a 2D image and a 3D model of the world.

16.1.4 Interior Orientation

This photogrammetry problem aims from going 3D $\longrightarrow$ 2D. The most common application of this problem is **camera calibration**. Camera calibration is crucial for high-precision imaging, as well as solving machine and computer vision problems such as Bundle Adjustment [1]. Finding a principal point is another example of the interior orientation problem.

16.2 Absolute Orientation

We will begin our in-depth discussion and analysis of these four problems with **absolute orientation**. Let us start by introducing binocular stereo.

16.2.1 Binocular Stereopsis

To motivate binocular stereo, we will start with a fun fact: Humans have ≈ 12 depth cues. One of these is **binocular stereopsis**, or **binocular stereo** (binocular ≈ two sensors).

Binocular stereo is given by the figure below:

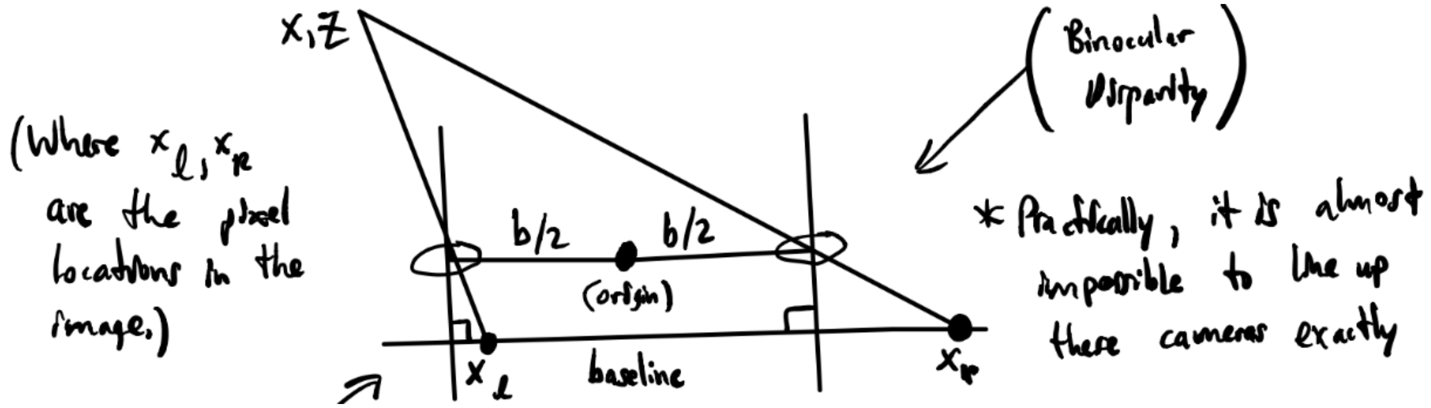

X, Z
(Where x_l, x_r are the pixel locations in the image.)
b/2
b/2
(origin)
baseline
x_l
x_r
(Binocular Disparity)
* Practically, it is almost impossible to line up these cameras exactly

Figure 26: The problem of binocular stereo. Having two 2D sensors enables us to recover 3D structure (specifically, depth) from the scene we image. A few key terms/technicalities to note here: (i) The origin is set to be halfway between the two cameras, (ii) The distance between the cameras is called the **baseline**, and (iii) **binocular disparity** refers to the phenomenon of having each camera generate a different image. Finally, note that in practice, it is almost impossible to line up these cameras exactly.

Goal: Calculate X and Z. This would not be possible with only monocular stereo (monocular ≈ one sensor). Using similar triangles, we have:

$$\frac{X - \frac{b}{2}}{Z} = \frac{x_l}{f}$$

$$\frac{X + \frac{b}{2}}{Z} = \frac{x_r}{f}$$

$$\longrightarrow \frac{x_r - x_l}{f} = \frac{b}{z} \implies z = \frac{bf}{x_r - x_l}$$

Where the **binocular disparity** is given by $(x_r - x_l)$. Solving this system of equations becomes:

$$X = b\frac{x_r + x_l}{x_r - x_l},\ Z = bf\frac{1}{x_r - x_l}$$

Note that, more generally, we can calculate in the same way as well! From these equations, we can see that:

- Increasing the baseline increases X and Z (all things equal)
- Increasing the focal length increases Z.

But it is important to also be mindful of system constraints that determine what range, or the exact value, of what these variables can be. For instance, if we have a self-driving car, we cannot simply have the baseline distance between our two cameras be 100 meters, because this would require mounting the cameras 100 meters apart.

16.2.2 General Case

All of the methods we have looked at so far assume we have already found "interesting points", for instance, through feature detection algorithms (also known as "descriptors") such as SIFT, SURF, ORB, or FAST+BRIEF. In the general case, absolute orientation involves having a point in space that is measured by two separate frames of reference (could be a camera, or another sensor), and the goal is to find the **closest approach** between coordinate systems, a.k.a. the shortest line that connects them. We will arbitrarily assign the 3 x 3 matrix $(\mathbf{x}_l, \mathbf{y}_l, \mathbf{z}_l) \in \mathbb{R}^{3\times3}$ to refer to a "lefthanded" coordinate system, and 3 x 3 matrix $(\mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r) \in \mathbb{R}^{3\times3}$ to refer to a "righthanded" coordinate system, where our goal is to find the transformation between these two coordinate systems. The diagram below illustrates this:

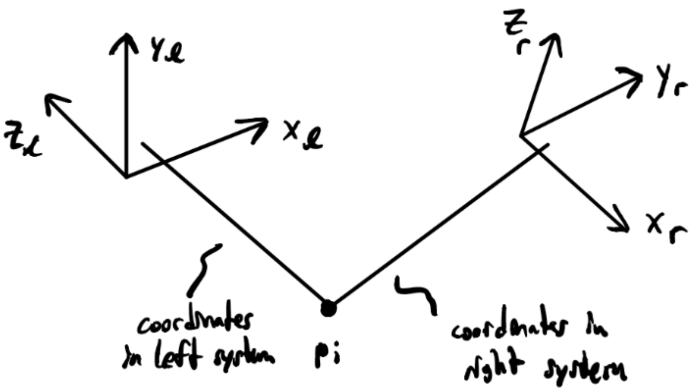


Figure 27: General case of absolute orientation: Given the coordinate systems $(\mathbf{x}_l, \mathbf{y}_l, \mathbf{z}_l) \in \mathbb{R}^{3\times3}$ and $(\mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r) \in \mathbb{R}^{3\times3}$, our goal is to find the transformation, or pose, between them using points measured in each frame of reference $\mathbf{p}_i$.

Here, we also note the following, as they are important for establishing that we are not limited just to finding the transformation between two camera sensors:

- "Two cameras" does not just mean having two distinct cameras (known as "two-view geometry"); this could also refer to having a single camera with images taken at two distinct points in time (known as "Visual Odometry" or "VO").
- Note also that we could have the same scenario described above when introducing this problem, which is that we have either one camera and multiple objects, or multiple objects and one cameras. Hence, there is a sense of duality here with solving these problems:
  1. One camera, two objects?
  2. Two cameras, one object (Two-View Geometry)?
  3. Camera moving (VO)?
  4. Object moving?

To better understand this problem, it is first important to precisely define the **transformation** or **pose** (translation + rotation) between the two cameras.

16.2.3 Transformations and Poses

Recall that in 3D, we have 6 degrees of freedom (DOF), of which **3** are for **translation**, and **3** are for **rotation**. In the general case of a d-dimensional system, we will have d **translational** degrees of freedom, and $\frac{d(d-1)}{2}$ **rotational** degrees of freedom (which, interestingly, is also equal to $\sum_{i=0}^{d-1} i$. Therefore, for a d-dimensional system, the total degrees of freedom from translation and rotation:

$$\mathbf{DOF}_{\text{T, R}}(d) = d + \frac{d(d-1)}{2}$$

What form does our transformation take? Since we are considering these transformations as a transformation determined by translation and rotation, then we can write our transformation as such:

$$\mathbf{r}_r = R(\mathbf{r}_l) + \mathbf{r}_0$$

Where:

- R describes the rotation. Note that this is not necessarily always an orthonormal rotation matrix, and we have more generally parameterized it as a function.
- $\mathbf{r}_0 \in \mathbb{R}^3$ describes the translation.
- The translation vector and the rotation function comprise our unknowns.

When R is described by an orthonormal rotation matrix $\mathbf{R}$, then we require this matrix to have the following properties:

1. $\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = I$, i.e. $\mathbf{R}^T = \mathbf{R}^{-1}$

2. $\mathbf{R}$ is skew-symmetric, so we end up having 3 unknowns instead of 9. Skew-symmetric matrices take the form:

$$\mathbf{R} = \begin{bmatrix} a & b & c \\ -b & d & e \\ -c & -e & f \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

3. $\det |R| = +1$ (This constraint is needed to eliminate reflections.)

4. $\mathbf{R} \in \mathbf{SO}(3)$ (Known as the Special Orthogonal group). One note about this: optimization over the Special Orthogonal group (e.g. for estimating optimal rotation matrices) can be difficult, and one way that this can be done is via Singular Value Decomposition (SVD) from this group/manifold onto some Euclidean space $\mathbb{R}^d$ where $d < 3$.

Supposing we have clusters of points from the left and righthand side, and we want to align clusters of points as closely as possible. With a mechanical spring analog, we have the force (given by Hooke's law), and consequently the energy:

$$F = Ke \tag{48}$$

$$E = \int F = \frac{1}{2}ke^2 \tag{49}$$

Where e (the "error") is the distance between the measured point in the two frames of reference. Therefore, the solution to this problem involves "minimizing the energy" of the system. Interestingly, energy minimization is analogous to least squares regression.

Using the definition of our transformation, our error for the i$^{\text{th}}$ point is given by:

$$\mathbf{e}_i = (R(\mathbf{r}_{l,i}) + \mathbf{r}_0) - \mathbf{r}_{r,i}$$

Then our objective for energy minimization and least squares regression is given by:

$$\min_{R,\mathbf{r}_0} \sum_{i=1}^{N} ||r_i||_2^2$$

This is another instance of solving what is known as the "inverse" problem. The "forward" and "inverse" problems are given by:

- **Forward Problem**: $R, \mathbf{r}_0 \longrightarrow \{\mathbf{r}_{r,i}, \mathbf{r}_{l,i}\}_{i=1}^N$ **(Find correspondences)**
- **Inverse Problem**: $\mathbf{r}_{l,i}\}_{i=1}^N \longrightarrow R, \mathbf{r}_0$ **(Find transformation)**

Now that we have framed our optimization problem, can we decouple the optimization over translation and rotation? It turns out we can by setting an initial reference point. For this, we consider two methods.

16.2.4 Procedure - "Method 1"

:

1. Pick a measured point from one set of measured points as the origin.

2. Take a second point, look at the distance between them, and compute the unit vector. Take unit vector as one axis of the system:

$$\mathbf{x}_l = \mathbf{r}_{l,2} - \mathbf{r}_{l,1} \longrightarrow \hat{\mathbf{x}}_l = \frac{\mathbf{x}_l}{||\mathbf{x}_l||_2}$$

3. Take a third vector, and compute the component of the vector that is equal to the vector from point 1 to point 2. Now, points 1, 2, and 3 from (x, y) plane, and the removed component from point 3 forms the y-component of the coordinate system.

4. We can compute the $\mathbf{y}$ vector:

$$\mathbf{y} = (\mathbf{r}_{l,3} - \mathbf{r}_{l,1}) - (\mathbf{r}_{l,3} - \mathbf{r}_{l,1}) \cdot \hat{\mathbf{x}}_l$$

From here, we then take the unit vector $\mathbf{y}_l = \frac{\mathbf{y}_l}{||\mathbf{y}_l||_2}$. From this, we know $\hat{\mathbf{x}}_l \cdot \hat{\mathbf{y}}_l = 0$.

5. Obtain the z-axis by the cross product:

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

   (Then we also have that $\hat{\mathbf{z}}_l \cdot \hat{\mathbf{y}}_l = 0$ and $\hat{\mathbf{z}}_l \cdot \hat{\mathbf{x}}_l = 0$. This then defines a coordinate system $(\hat{\mathbf{x}}_l, \hat{\mathbf{y}}_l, \hat{\mathbf{z}}_l)$ for the left camera/point of reference. Note that this only requires 3 points!

6. To calculate this for the righthand frame of reference, we can repeat steps 1-5 for the righthand side to obtain the coordinate system $(\hat{\mathbf{x}}_r, \hat{\mathbf{y}}_r, \hat{\mathbf{z}}_r)$.

From here, all we need to do is find the transformation (rotation, since we have artificially set the origin) between the coordinate system $(\hat{\mathbf{x}}_r, \hat{\mathbf{y}}_r, \hat{\mathbf{z}}_r)$ and $(\hat{\mathbf{x}}_l, \hat{\mathbf{y}}_l, \hat{\mathbf{z}}_l)$. Mathematically, we have the following equations:

$$\begin{aligned}
\hat{\mathbf{x}}_r &= R(\hat{\mathbf{x}}_l) \\
\hat{\mathbf{y}}_r &= R(\hat{\mathbf{y}}_l) \\
\hat{\mathbf{z}}_r &= R(\hat{\mathbf{z}}_l)
\end{aligned}$$

We can condense these equations into a matrix equation, and subsequently a matrix inversion problem:

$$\begin{bmatrix}\hat{\mathbf{x}}_r & \hat{\mathbf{y}}_r & \hat{\mathbf{z}}_r\end{bmatrix} = R\begin{bmatrix}\hat{\mathbf{x}}_l & \hat{\mathbf{y}}_l & \hat{\mathbf{z}}_l\end{bmatrix}$$

Then we can solve this as a matrix inversion problem:

$$R = \begin{bmatrix}\hat{\mathbf{x}}_r & \hat{\mathbf{y}}_r & \hat{\mathbf{z}}_r\end{bmatrix}\begin{bmatrix}\hat{\mathbf{x}}_l & \hat{\mathbf{y}}_l & \hat{\mathbf{z}}_l\end{bmatrix}^{-1}$$

A few notes on this system:

- Is this system invertible? Yes, because by construction these vectors are orthogonal to one another.
- 3 vector equalities $\longrightarrow$ 9 scalar equalities
- Are we using all of these constraints?
  - For point 1, we use all 3 constraints.
  - For point 2, we use 2 constraints.
  - For point 3, we use 1 constraint.

  It turns out this system "favors" points over one another (usually sub-optimal).
- This is a somewhat ad hoc method - this could be useful for deriving a fast approximation or initial estimate, but this method is definitely suboptimal.

16.2.5 Procedure - "Method 2"

For this algorithm, recall our prior work with **blob analysis**. We can look at axes of inertia (see the figure below), where the inertia of the "blob" is given by:

$$I = \iint_O r^2 dm$$

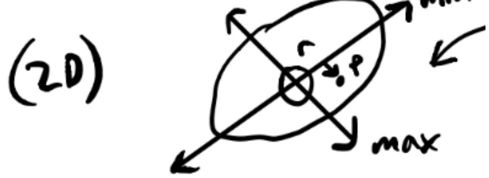


Figure 28: Computing the axes of inertia in a 2D blob.

How do we generalize this from 2D to 3D? How do we figure out these axes of inertia (see the example below)?

$$I = \iiint_O r^2 dm$$

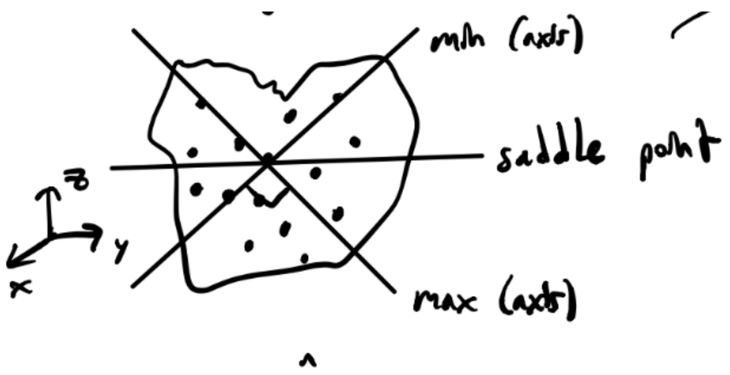


Figure 29: Computing the axes of inertia for a 3D blob - we can generalize the notion of inertia from 2D to 3D.

One trick we can use here is using the **centroid** as the origin.

Using the triangle figure below:

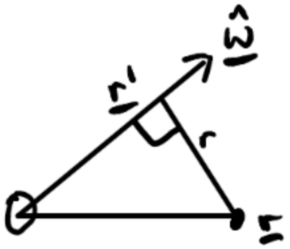


Figure 30: Triangle for computing our $\mathbf{r}'$ vector.

Then we can compute the hypotenuse $\mathbf{r}'$ vector in the $\hat{\boldsymbol{\omega}}$ direction:

$$\mathbf{r}' = (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})\hat{\boldsymbol{\omega}}$$

Then:

$$\mathbf{r} - \mathbf{r}' = \mathbf{r} - (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})\hat{\boldsymbol{\omega}}$$

And we can compute $\mathbf{r}^2$:

$$\begin{aligned}\mathbf{r}^2 &= (\mathbf{r} - \mathbf{r}') \cdot (\mathbf{r} - \mathbf{r}') \\ &= \mathbf{r} \cdot \mathbf{r} - 2(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 + (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2(\hat{\boldsymbol{\omega}} \cdot \hat{\boldsymbol{\omega}}) \\ &= \mathbf{r} \cdot \mathbf{r} - 2(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 + (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 \\ &= \mathbf{r} \cdot \mathbf{r} - (\mathbf{r} \cdot \hat{\omega})^2\end{aligned}$$

Substituting this into our inertia expression:

$$I = \iiint_O (\mathbf{r} \cdot \mathbf{r} - (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2) dm$$

Let us simplify some of these formulas:

1. $(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2$:

$$\begin{aligned}(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 &= (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})(\mathbf{r} \cdot \hat{\boldsymbol{\omega}}) \\ &= (\hat{\boldsymbol{\omega}} \cdot \mathbf{r})(\mathbf{r}\hat{\boldsymbol{\omega}}) \\ &= \hat{\boldsymbol{\omega}}^T(\mathbf{r}\mathbf{r}^T)\hat{\boldsymbol{\omega}}\end{aligned}$$

Where $\mathbf{r}\mathbf{r}^T \in \mathbb{R}^{3 \times 3}$ is the dyadic product.

2. $(\mathbf{r}\cdot\mathbf{r})$:

$$\begin{aligned}(\mathbf{r}\cdot\mathbf{r}) &= (\mathbf{r}\cdot\mathbf{r})(\hat{\boldsymbol{\omega}}\cdot\hat{\boldsymbol{\omega}}) \\ &= (\mathbf{r}\cdot\mathbf{r})\hat{\boldsymbol{\omega}}^T\mathbf{I}_3\hat{\boldsymbol{\omega}}\end{aligned}$$

Where $\mathbf{I}_3$ is the 3×3 identity matrix.

Therefore, the inertia matrix becomes:

$$\begin{aligned}I &= \iiint_O (\mathbf{r}\cdot\mathbf{r} - (\mathbf{r}\cdot\hat{\boldsymbol{\omega}})^2)dm \\ &= \iiint_O (\mathbf{r}\cdot\mathbf{r} - (\mathbf{r}\cdot\hat{\boldsymbol{\omega}})^2)dm \\ &= \iiint_O (\mathbf{r}\cdot\mathbf{r})\hat{\boldsymbol{\omega}}^T\mathbf{I}_3\hat{\boldsymbol{\omega}} - \hat{\boldsymbol{\omega}}^T(\mathbf{r}\mathbf{r}^T)\hat{\boldsymbol{\omega}})dm \\ &= \hat{\boldsymbol{\omega}}\Big(\iiint_O ((\mathbf{r}\cdot\mathbf{r})\mathbf{I}_3 - \mathbf{r}\mathbf{r}^T)dm\Big)\hat{\boldsymbol{\omega}}\end{aligned}$$

From this expression, we want to find the extrema. We can solve for the extrema by solving for the minimium and maximums of this objective:

1. **Minimum**: $\min_{\hat{\boldsymbol{\omega}}} \hat{\boldsymbol{\omega}}^T A\hat{\boldsymbol{\omega}}$
2. **Maximum**: $\max_{\hat{\boldsymbol{\omega}}} \hat{\boldsymbol{\omega}}^T A\hat{\boldsymbol{\omega}}$

Where $A \triangleq \iiint_O ((\mathbf{r}\cdot\mathbf{r})\mathbf{I}_3 - \mathbf{r}\mathbf{r}^T)dm$. This matrix is known as the "inertia matrix".

How can we solve this problem? We can do so by looking for the **eigenvectors** of the inertia matrix:

- For minimization, the **eigenvector** corresponding to the **smallest eigenvalue** of the inertia matrix corresponds to our solution.
- For maximization, the **eigenvector** corresponding to the **largest eigenvalue** of the inertia matrix corresponds to our solution.
- For finding the saddle point, our solution will be the **eigenvector** corresponding to the **middle eigenvalue**.

Since this is a polynomial system of degree 3, we have a closed-form solution! These three eigenvectors will form a coordinate system for the lefthand system.

Taking a step back, let us look at what we have done so far. We have taken the cloud of points from the left frame of reference/coordinate system and have estimated a coordinate system for it by finding an **eigenbasis** from solving these optimization problems over the objective $\hat{\boldsymbol{\omega}}^T A\hat{\boldsymbol{\omega}}$. With this, we can then repeat the same process for the righthand system.

Why does this approach work better?

- We use all the data in both point clouds of 3D data.
- We weight all the points equally, unlike in teh previous case.

But why is this approach not used in practice?

- This approach fails under symmetry - i.e. of the inertia is hte same in all directions (for shapes such as spheres, polyhedra, octahedra, cube, etc.)
- "Double-Edged Sword" - using correspondences can provide you with more information, but can run into issues with symmetry.

16.2.6 Computing Rotations

Next, we will dive into how we can compute and find rotations. To do this, let us first go over the following properties of rotations:

1. **Rotations preserve dot products**: $R(\mathbf{a})\cdot R(\mathbf{b}) = \mathbf{a}\cdot\mathbf{b}$

2. **Rotations preserve length**: $R(\mathbf{a}) \cdot R(\mathbf{a}) = \mathbf{a} \cdot \mathbf{a}$
3. **Rotations preserve angles**: $|R(\mathbf{a}) \times R(\mathbf{b})| = |\mathbf{a} \times \mathbf{b}|$
4. **Rotations preserve triple products**: $[R(\mathbf{a})\ R(\mathbf{b})\ R(\mathbf{c})] = [\mathbf{a}\ \mathbf{b}\ \mathbf{c}]$ (Where the triple product $[\mathbf{a}\ \mathbf{b}\ \mathbf{c}] = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$).

Using these properties, we are now ready to set this up as least squares, using correspondences between points measured between two coordinate systems:

Transform: $\mathbf{r}_r = R(\mathbf{r}_l) + \mathbf{r}_0$

Error: $\mathbf{e}_i = \mathbf{r}_{r,i} - R(\mathbf{r}_{l,i}) - \mathbf{r}_0$

And we can write our **optimization** as one that minimizes this error term:

$$R^*, \mathbf{r}_0^* \sum_{i=1}^{N} ||\mathbf{e}_i||^2$$

Next, we can compute teh centroids of the left and right systems:

$$\bar{\mathbf{r}}_l = \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}_{l,i},\ \bar{\mathbf{r}}_r = \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}_{r,i}$$

We can use these computed centroids from points so we do not have to worry about translation. A new feature of this system is that the new centroid is at the origin. To prove this, let us "de-mean" (subtract the mean) of our coordinates in the left and righthand coordinate systems:

$$\mathbf{r}'_{l,i} = \mathbf{r}_{l,i} - \bar{\mathbf{r}}_l,\ \mathbf{r}'_{r,i} = \mathbf{r}_{r,i} - \bar{\mathbf{r}}_r$$

Because we subtract the mean, the mean of these new points now becomes zero:

$$\hat{\mathbf{r}}'_l = \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}'_{l,i} = 0 = \hat{\mathbf{r}}'_r = \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}'_{r,i}$$

Substituting this back into the objective, we can solve for an optimal rotation R:

$$\begin{aligned} R^* &= \min_{R,\mathbf{r}'_0} \sum_{i=1}^{N} ||\mathbf{r}'_i - R(\mathbf{r}'_{l,i} - \mathbf{r}'_0)||_2^2 \\ &= \min_{R,\mathbf{r}'_0} \sum_{i=1}^{N} ||\mathbf{r}'_i - R(\mathbf{r}'_{l,i})||_2^2 - 2\mathbf{r}'_0 \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}_{l,i})) + N||\mathbf{r}'_0||_2^2 \\ &= \min_{R,\mathbf{r}'_0} \sum_{i=1}^{N} ||\mathbf{r}'_i - R(\mathbf{r}'_{l,i})||_2^2 + N||\mathbf{r}'_0||_2^2 \end{aligned}$$

Since only the last term depends on $\mathbf{r}'_0$, we can set $\mathbf{r}'_0$ to minimize. Moreover, we can solve for the true $\mathbf{r}_0$ by back-solving later:

$$\mathbf{r}_0 = \bar{\mathbf{r}}_r - R(\bar{\mathbf{r}}_l)$$

Intuitively, this makes sense: the translation vector between these two coordinate systems/point clouds is the difference between the centroid of the right point cloud and the centroid of the left point cloud after it has been rotated.

Since we now have that $\mathbf{r}'_0 = \mathbf{0} \in \mathbb{R}^3$, we can write our error term as:

$$\mathbf{e}_i = \mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})$$

Which in turn allows us to write the objective as:

$$\begin{aligned} \min \sum_{i=1}^{N} ||\mathbf{e}_i||_2^2 &= \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) \\ &= \sum_{i=1}^{N} ||\mathbf{r}'_{r,i}||_2^2 - \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) - \sum_{i=1}^{N} ||\mathbf{r}'_{l,i}||_2^2 \end{aligned}$$

Where the first of these terms is fixed, the second of these terms is to be maximized (since there is a negative sign in front of this term, this thereby minimizes the objective), and the third of these terms is fixed. Therefore, our rotation problem can be simplified to:

$$\min \sum_{i=1}^{N} ||\mathbf{e}_i||_2^2 = -2\sum_{i=1}^{N}(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) \tag{50}$$

To solve this objective, we could take the derivative $\frac{d}{dR}(\cdot)$ of the objective, but constraints make this optimization problem difficult to solve (note that we are not optimizing over a Euclidean search space - rather, we are optimizing over a generalized transformation). We will look into a different representation of R for next class to find solutions to this problem!

16.3 References

1. Bundle Adjustment, https://en.wikipedia.org/wiki/Bundle_adjustment

17 Lecture 18: Rotation and How to Represent it, Unit Quaternions, the Space of Rotations

Today, we will focus on rotations. Note that unlike **translations**, **rotations** are not **commutative**, which makes fitting estimates to data, amongst other machine vision tasks, more challenging. In this lecture, we will cover rotation in terms of:

- Properties
- Representations
- Hamilton's Quaternions
- Rotation as Unit Quaternion
- Space of rotations
- Photogrammetry
- Closed-form solution of absolute quaternions
- Divison algebras, quaternion analysis, space-time

We will start by looking at some motivations for why we might care about how we formulate and formalize rotations: **What is rotation used for?**

- Machine vision
- Recognition/orientation
- Graphics/CAD
- Virtual Reality
- Protein Folding
- Vehicle Attitude
- Robotics
- Spatial Reasoning
- Path Planning - Collision Avoidance

17.1 Euclidean Motion and Rotation

Rotation and translation form Euclidean motion. Some properties of Euclidean motion, including this property. Euclidean motion:

- Contains **translation** and **rotation**
- Preserves distances between points
- Preserves angles between lines
- Preserves handedness
- Preserves dot-products
- Preserves triple products
- Does not contain:
  - Reflections
  - Skewing
  - Scaling

17.2 Basic Properties of Rotation

Now that we have framed rotation to be a property of Euclidean motion, we can dive further into some properties of rotation:

- **Euler's Theorem**: There is a line of fixed points that remain the same in any rotation. This is known as the axis of rotation.
- **Parallel axis theorem**: Any rotation is equivalent to rotation through the origin along with a translation. This allows for the decoupling between translation and rotation.
- **Rotation of sphere includes rotation of space**: I.e, any rotation that is induced on a space can be induced on an equivalent-dimensional sphere.
- **Attitude, Orientation - Rotation Relative to Reference**: This helps for determining the orientation of objects relative to another frame of reference.
- **Degrees of Freedom (in 3D)** is 3.

Some additional properties that are helpful for understanding rotation:

- **Rotational velocity** can be described by a vector. This set of rotational vectors together form a **lie algebra**, specifically, $\mathbf{so}(3)$. The set of rotations form the counterpart to this lie algebra, namely the **Special Orthogonal group** $\mathbf{SO}(3)$, which is a **Lie group**. If you are not familiar with Lie groups/algebras, here are a few useful terms to get started with these concepts:
  - A **group** is a set equipped with a binary operation that combines any two elements to form a third element in such a way that four conditions called group axioms are satisfied, namely closure, associativity, identity and invertibility [1].
  - A **manifold** is a d-dimensional topological space that locally resembles Euclidean space at each point i.e. any patch of the manifold can be "locally" approximated as a d-dimensional Euclidean space [2].
  - A **Lie group** is a group that is also a differentiable manifold. One key Lie group that we will study in this course is the **Special Orthogonal group**, known as $\mathbf{SO}(3)$, which is the space of orthonormal rotation matrices.
- We can define the derivative of the radius vector $\dot{\mathbf{r}}$ using **Poisson's Formula**: $\dot{\mathbf{r}} = \hat{\boldsymbol{\omega}} \times \mathbf{r}$
- **Rotational velocities add** - this holds because these vectors belong to **Lie algebras**
- **Finite rotations do not commute** - this holds because rotations are defined by **Lie groups** in a non-Euclidean space.
- The **Degrees of Freedom** for rotation in dimension $\mathbf{n}$ is given by $\frac{n(n-1)}{2}$, which coincidentally equals 3 in 3 dimensions.
- **Intuition**: Oftentimes, it is easier to think about rotation "in planes", rather than "about axes". Rotations preserve points in certain planes.

17.2.1 Isomorphism Vectors and Skew-Symmetric Matrices

A technical note that is relevant when discussing cross products: Although a cross product produces a vector in 3D, in higher dimensions the result of a cross product is a **subspace**, rather than a vector.

Specifically, this **subspace** that forms the result of a higher-dimensional cross product is the space that is **perpendicular/orthogonal** to the two vectors the cross product operator is applied between.

With this set up, we can think of cross products as producing the following isomorphism vectors and skew-symmetric matrices: **One Representation**:

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

An Isomorphic Representation:

$$\mathbf{a} \times \mathbf{b} = \bar{\mathbf{B}}\mathbf{a}$$
$$\mathbf{A} = \begin{bmatrix} 0 & b_z & -b_y \\ -b_z & 0 & b_x \\ b_y & -b_x & 0 \end{bmatrix}$$

Note that while these skew-symmetric matrices have 9 elements as they are 3 × 3 matrices, they only have 3 DOF.

17.3 Representations for Rotation

There are a myriad of representations for rotations, highlighting that different applications/domains/problem-solving techniques demand different representations for these transformations. Some of these representations include (we will proceed in greater detail about these later):

1. Axis and angle
2. Euler Angles
3. Orthonormal Matrices
4. Exponential cross product
5. Stereography plus bilinear complex map
6. Pauli Spin Matrices
7. Euler Parameters
8. Unit Quaternions

Let us delve into each of these in a little more depth.

17.3.1 Axis and Angle

This representation is composed of a vector $\hat{\boldsymbol{\omega}}$ and an angle θ, along with the **Gibb's vector** that combines these given by $\hat{\boldsymbol{\omega}} \tan\left(\frac{\theta}{2}\right)$, which has magnitude $\tan\left(\frac{\theta}{2}\right)$, providing the system with an additional degree of freedom that is not afforded by unit vectors. Therefore, we have our full 3 rotational DOF.

17.3.2 Euler Angles

A few notes about these:

- There are over 24 definitions of Euler angles! This is the case because these definitions are permutations - i.e. the order of the angles matters here. The order of angular composition matters.
- These rotations are defined through each axis, **roll**, **pitch**, and **yaw**.

17.3.3 Orthonormal Matrices

We have studied these previously, but these are the matrices that have the following properties:

1. $\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = I, \mathbf{R}^T = \mathbf{R}^{-1}$ (skew-symmetric)
2. $\det |R| = +1$
3. $\mathbf{R} \in \mathbf{SO}(3)$ (see notes on groups above) - being a member of this Special Orthogonal group is contingent on satisfying the properties above.

17.3.4 Exponential Cross Product

We can also write rotations in terms of a matrix exponential. To derive this matrix exponential, let us consider the matrix first-order, homogeneous differential equation defined by:

$$\frac{d\mathbf{R}}{d\theta} = \mathbf{\Omega}\mathbf{R}$$

Has the solution given by the matrix exponential:

$$\mathbf{R} = e^{\theta\mathbf{\Omega}}$$

Taking the Taylor expansion of this expression, we can write this matrix exponential as:

$$\begin{aligned}\mathbf{R} &= e^{\theta\mathbf{\Omega}} \\ &= \sum_{i=0}^{\infty} \frac{1}{i!}(\theta\mathbf{\Omega})^i \\ &= \sum_{i=0}^{\infty} \frac{\theta^i}{i!}(\mathbf{V}_\Omega\mathbf{\Lambda}_\Omega^i\mathbf{V}_\Omega^T) \;\; \textbf{(Taking an eigendecomposition of } \mathbf{\Omega} = \mathbf{V}_\Omega\mathbf{\Lambda}_\Omega\mathbf{V}_\Omega^T\textbf{)}\end{aligned}$$

(Optional) Let us look a little deeper into the mathematics behind this exponential cross product. We can write a rotation about $\hat{\boldsymbol{\omega}}$ through angle θ as:

$$\begin{aligned}&\mathbf{r} = R(\theta)\mathbf{r}_0 \\ &\frac{d\mathbf{r}}{d\theta} = \frac{d}{d\theta}(R(\theta)\mathbf{r}_0) \\ &\frac{d\mathbf{r}}{d\theta} = \hat{\boldsymbol{\omega}} \times \mathbf{r} = \mathbf{\Omega}\mathbf{r} = \mathbf{\Omega}R(\theta)\mathbf{r}_0 \\ &\frac{d}{d\theta}R(\theta)\mathbf{r}_0 = \mathbf{\Omega}R(\theta)\mathbf{r}_0\end{aligned}$$

Then for all $\mathbf{r}_0$:

$$\frac{d}{d\theta}R(\theta) = \mathbf{\Omega}R(\theta) \implies R(\theta) = e^{\theta\mathbf{\Omega}}$$

17.3.5 Stereography Plus Bilinear Complex Map

Intuitively, this notion of rotation corresponds to mapping a sphere to a complex plane, making transformations in the complex plane, and then mapping this transformed mapping from the complex plane back to the sphere. A few points here:

- This is an instance of **Stereography**: a **conformal** (angle-preserving) spherical mapping.
- The rotation of a sphere induces the rotation of a space.
- This plane is treated as a complex plane that we can apply homogeneous transforms to.

We can conceptually visualize this mapping from a sphere to the complex plane using the figure below:

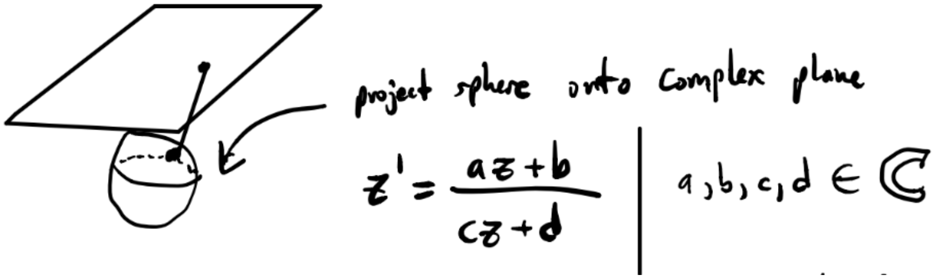

project sphere onto Complex plane
$z' = \frac{az+b}{cz+d}$
$a, b, c, d \in \mathbb{C}$

Figure 31: Mapping from a sphere to a complex plane, which we then apply a homogeneous transformation to and map back to the sphere in order to induce a rotation.

Specifically, the homogeneous transform we consider maps the complex variable z to z':

$$z' = \frac{az+b}{cz+d}, \text{ for some } a, b, c, d \in \mathbb{C}$$

We can actually generalize this framework even further - any rotation in 3-space can be thought of as an operation in a complex plane.

17.3.6 Pauli Spin Matrices

With a physical motivation, these are 2×2 complex-valued, unitary, Hermitian matrices ($\mathbf{A} = \bar{\mathbf{A}}^T$). All these constraints create 3 DOF for these matrices:

$$\mathbf{S}_x = \frac{\hbar}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathbf{S}_y = \frac{\hbar}{2}\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \mathbf{S}_z = \frac{\hbar}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

17.3.7 Euler Parameters

There are also known as **Rodrigues** parameters, leading to **Rodrigues formula**, which can be thought of as a rotation about the unit vector $\hat{\boldsymbol{\omega}}$ through the angle θ:

$$\mathbf{r}' = (\cos\theta)\mathbf{r} + (1 - \cos\theta)(\hat{\boldsymbol{\omega}} \cdot \mathbf{r})\hat{\boldsymbol{\omega}} + \sin\theta(\hat{\boldsymbol{\omega}} \times \mathbf{r})$$

The geometry of this problem can be understood through the following figure:

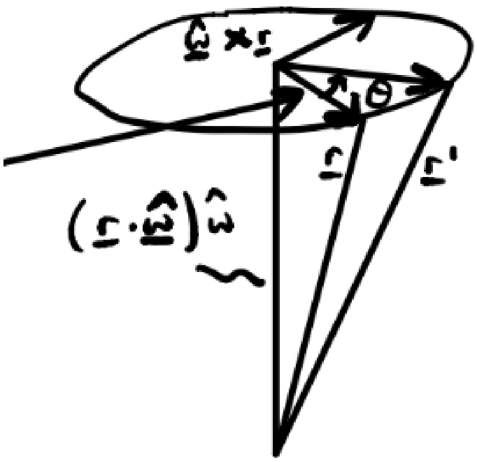


Figure 32: Geometric interpretation of the Rodrigues formula: Rotation about the vector $\hat{\boldsymbol{\omega}}$ through an angle θ.

One disadvantage of this approach is that there is no way to have **compositions** of rotations.

Next, let us take an in-depth analysis of the Rodrigues Formula and the Exponential Cross Product:

$$\frac{d\mathbf{R}}{d\theta} = \mathbf{\Omega R} \implies \mathbf{R} = e^{\mathbf{\Omega}\theta}$$

$$e^{\mathbf{\Omega}\theta} = \mathbf{I} + \theta\mathbf{\Omega} + \frac{1}{2!}(\theta\mathbf{\Omega})^2 + ... = \sum_{i=0}^{\infty}\frac{\theta^i}{i!}\mathbf{\Omega}^i$$

Next, we have that:

$$\begin{aligned}\mathbf{\Omega}^2 &= (\hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T - \mathbf{I})\\ \mathbf{\Omega}^3 &= -\mathbf{\Omega}\end{aligned}$$

We can then write this matrix exponential as:

$$\begin{aligned}e^{\theta\mathbf{\Omega}} &= \mathbf{I} + \mathbf{\Omega}(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} + \cdots) + \mathbf{\Omega}^2(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} + \cdots)\\ &= \mathbf{I} + \mathbf{\Omega}(\sum_{i=0}^{\infty}\frac{\theta^{2i+1}}{(2i+1)!}(-1)^i) + \mathbf{\Omega}^2(\sum_{i=0}^{\infty}\frac{\theta^{2i+2}}{(2i+2)!}(-1)^i)\\ &= \mathbf{I} + \mathbf{\Omega}\sin\theta + \mathbf{\Omega}^2(1-\cos\theta)\\ &= \mathbf{I} + (\sin\theta)\mathbf{\Omega} + (\hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T - \mathbf{I})(1-\cos\theta)\\ &= (\cos\theta)\mathbf{I} + (\sin\theta)\mathbf{\Omega} + (1-\cos\theta)\hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T\end{aligned}$$

From this, we have:

$$\begin{aligned}\mathbf{r} &= e^{\theta\mathbf{\Omega}}\mathbf{r}\\ \mathbf{r}' &= (\cos\theta)\mathbf{r} + (1-\cos\theta)(\hat{\boldsymbol{\omega}}\cdot\mathbf{r})\hat{\boldsymbol{\omega}} + \sin\theta(\hat{\boldsymbol{\omega}}\times\mathbf{r})\end{aligned}$$

Where the last line is the result of the Rodrigues formula.

17.4 Desirable Properties of Rotations

We would like rotations to exhibit the following properties:

- The ability to rotate vectors - or coordinate systems
- The ability to compose rotations
- Intuitive, non-redundant representation - e.g. rotation matrices have 9 entries but only 3 degrees of freedom
- Computational efficiency
- Interpolate orientations
- Averages over range of rotations
- Derivative with respect to rotations - e.g. for optimization and least squares
- Sampling of rotations - uniform and random (e.g. if we do not have access to closed-form solutions)
- Notion of a space of rotations

17.5 Problems with Some Rotation Representations

Let us discuss some problems with the representations we introduced and discussed above:

1. **Orthonormal Matrices**: Redundant, with complex constraints.
2. **Euler Angles**: Inability to compose rotations, "gimbal lock" (when two axes line up, you lose a DOF, which can be disastrous for control systems).
3. **Gibb's Vector**: Singularity when $\theta = \pi$, since $\tan(\frac{\pi}{2})$ is not defined.
4. **Axis and Angle**: Inability to compose rotations
5. There is no notion of the "space of rotations" (at least not in Euclidean space).

How can we overcome these challenges? One way through which we can do is to look at another representation approach, this time through Hamilton.

17.6 Quaternions

In this section, we will discuss another way to represent rotations: quaternions.

17.6.1 Hamilton and Division Algebras

Goal: Develop an algebra where we can use algebraic operations such as addition, subtraction, mulitplication, and division.

1. Hamilton's representation is motivated by **Algebraic couples**, for instance, complex numbers as pairs of reals $a+bi, a-bi$
2. **Here, we want:** Multiplicative inverses
3. **Examples**: Real numbers, complex numbers
4. **Expected next**: Three conponents (vectors) - note that this was before Gibb's vector and 3-vectors

Hamilton's insight here is that these quaternions require a "4th dimension for the sole purpose of calculating triples".

17.6.2 Hamilton's Quaternions

Hamilton noted the following insights in order to formalize his quaternion:

- Quaternions cannot be constructed with three components
- Quaternions need additional square roots of -1

Therefore, the complex components i, j, k defined have the following properties:

1. $i^2 = j^2 = k^2 = ijk = -1$
2. From which follows:
   (a) $ij = k$
   (b) $jk = i$
   (c) $ki = j$
   (d) $ji = -k$
   (e) $kj = -i$
   (f) $ik = -j$

   Note: As you can see from these properties, multiplication of the components of these quaternions is not commutatitve.

17.6.3 Representations of Quaternions

There are several ways through which we can represent these quaternions:

1. **Real and imaginary parts**: $q_0 + iq_x + jq_y + kq_z$
2. **Scalar and 3-vector**: $(q, \mathbf{q})$
3. **4-vector**: $\overset{o}{q}$
4. **Certain Orthogonal 4 x 4 Matrices q** (this is an isomorphism of quaternions that allows us to do multiplications).
5. **Complex Composite of Two Complex Numbers**: Here, we have $(a + bi)$ and $(c + di)$, and we replace all the real constants a, b, c, d with complex numbers. We can build sophisticated algebras from these operations.

17.6.4 Representations for Quaternion Multiplication

With several ways to represent these quaternions, we also have several ways through which we can represent quaternion multiplication:

1. **Real and 3 Imaginary Parts**:

$$\begin{aligned}(p_0 + p_x i + p_y j + p_z k)((q_0 + q_x i + q_y j + q_z k) =&(p_0q_0 - p_xq_x - p_yq_y - p_zq_z)+\\ &(p_0q_x + p_xq_0 + p_yq_z - p_zq_y)i+\\ &(p_0q_y - p_xq_z + p_yq_z + p_zq_x)j+\\ &(p_0q_z + p_xq_y - p_yq_x + p_zq_0)k\end{aligned}$$

2. **Scalar and 3-Vector**:

$$(p, \mathbf{p})(q, \mathbf{q}) = (pq - \mathbf{p} \cdot \mathbf{q}, p\mathbf{q} + q\mathbf{p} + \mathbf{p} \times \mathbf{q})$$

   Note: This multiplication operation is not commutative.

3. **4-Vector**:

$$\begin{bmatrix} p_0 & -p_x & -p_y & -p_z \\ p_x & p_0 & -p_z & p_y \\ p_y & p_z & p_0 & -p_x \\ p_z & -p_y & p_x & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix}$$

   Note: Here we also have an isomorphism between the quaternion and the 4 x 4 orthogonal matrix (this matrix is orthonormal if we have unit quaternions). Here we can show the isomorphism and relate this back to the cross product we saw before by considering the equivalence of the two following righthand-side expressions:

   (a) $\overset{o}{p}\overset{o}{q} = P\overset{o}{q}$, where $P = \begin{bmatrix} p_0 & -p_x & -p_y & -p_z \\ p_x & p_0 & -p_z & p_y \\ p_y & p_z & p_0 & -p_x \\ p_z & -p_y & p_x & p_0 \end{bmatrix}$

   (b) $\overset{o}{p}\overset{o}{q} = \bar{\mathbf{q}}\overset{o}{p}$, where $\bar{\mathbf{q}} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & q_z & -q_y \\ q_y & -q_z & q_0 & q_x \\ q_z & q_y & -q_x & q_0 \end{bmatrix}$

   A few notes about these matrices $\mathbf{P}$ and $\bar{\mathbf{q}}$:

   - These matrices are orthonormal if quaternions are unit quaternions
   - $\mathbf{P}$ is normal if $\overset{o}{p}$ is a unit quaternion, and $\bar{\mathbf{q}}$ is normal if $\overset{o}{q}$ is a unit quaternion.
   - $\mathbf{P}$ is skew-symmetric if $\overset{o}{p}$ has zero scalar part, and $\bar{\mathbf{q}}$ is skew-symmetric if $\overset{o}{q}$ has zero scalar part.
   - $\mathbf{P}$ and $\bar{\mathbf{q}}$ have the same signs for the first row and column, and flipped signs for off-diagonal entries in the bottom right 3 x 3 blocks of their respective matrices.

17.6.5 Properties of 4-Vector Quaternions

These properties will be useful for representing vectors and operators such as rotation later:

1. **Not commutative**: $\overset{o}{p}\overset{o}{q} \neq \overset{o}{q}\overset{o}{p}$

2. **Associative**: $(\overset{o}{p}\overset{o}{q})\overset{o}{r} = \overset{o}{p}(\overset{o}{q}\overset{o}{r})$

3. **Conjugate**: $(p, \mathbf{p})^* = (p, -\mathbf{p}) \implies (\overset{o}{p}\overset{o}{q}) = \overset{o^*}{q}\,\overset{o}{p}$

4. **Dot Product**: $(p, \mathbf{p}) \cdot (q, \mathbf{q}) = pq + \mathbf{p} + \mathbf{q}$

5. **Norm**: $||\overset{o}{q}||_2^2 = \overset{o}{q} \cdot \overset{o}{q}$

6. **Conjugate Multiplication**: $\overset{o}{q}\overset{o}{q}^*$:

$$\begin{aligned}\overset{o}{q}\overset{o}{q}^* &= (q, \mathbf{q})(q, -\mathbf{q}) \\ &= (q^2 + \mathbf{q}\cdot\mathbf{q}, 0) \\ &= (\overset{o}{q}\cdot\overset{o}{q})\overset{o}{e}\end{aligned}$$

Where $\overset{o}{e} \triangleq (1, 0)$, i.e. it is a quaternion with no vector component. Conversely, then, we have: $\overset{o}{q}^*\overset{o}{q} = (\overset{o}{q}\overset{o}{q})\overset{o}{e}$.

7. **Multiplicative Inverse**: $\overset{o}{q}^{-1} = \frac{\overset{o}{q}^*}{(\overset{o}{q}\cdot\overset{o}{q}}$ (Except for $\overset{o}{q} = (0, \mathbf{0})$, which is problematic with other representations anyway.)

We can also look at properties with dot products:

1. $(\overset{o}{p}\overset{o}{q})\cdot(\overset{o}{p}\overset{o}{q}) = (\overset{o}{p}\cdot\overset{o}{p})(\overset{o}{q}\cdot\overset{o}{q})$
2. $(\overset{o}{p}\overset{o}{q})\cdot(\overset{o}{p}\overset{o}{r}) = (\overset{o}{p}\cdot\overset{o}{p})(\overset{o}{q}\cdot\overset{o}{r})$
3. $(\overset{o}{p}\overset{o}{q})\cdot\overset{o}{r} = \overset{o}{p}\cdot(\overset{o}{r}\overset{o}{q}^*)$

We can also represent these quaternions as vectors. Note that these quaternions all have zero scalar component.

1. $\overset{o}{r} = (0, \mathbf{r})$
2. $\overset{o}{r}^* = (0, -\mathbf{r})$
3. $\overset{o}{r}\cdot\overset{o}{s} = \mathbf{r}\cdot\mathbf{s}$
4. $\overset{o}{r}\overset{o}{s} = (-\mathbf{r}\cdot\mathbf{s}, \mathbf{r}\times\mathbf{s})$
5. $(\overset{o}{r}\overset{o}{s})\cdot\overset{o}{t} = \overset{o}{r}\cdot(\overset{o}{s}\overset{o}{t}) = [\mathbf{r}\ \mathbf{s}\ \mathbf{t}]$ (Triple Products)
6. $\overset{o}{r}\overset{o}{r} = -(\mathbf{r}\cdot\mathbf{r})\overset{o}{e}$

Another **note**: For representing rotations, we will use unit quaternions. We can represent scalars and vectors with:

- **Representing scalars**: $(s, \mathbf{0})$
- **Representing vectors**: $(0, \mathbf{v})$

17.7 Quaternion Rotation Operator

To represent a rotation operator using quaternions, we need a quaternion operation that maps from vectors to vectors. More specifically, we need an operation that maps from 4D, the operation in which quaternions reside, to 3D in order to ensure that we are in the correct for rotation in 3-space. Therefore, our rotation operator is given:

$$\begin{aligned}\overset{o}{r}' = R(\overset{o}{r}) &= \overset{o}{q}\overset{o}{r}\overset{o}{q}^* \\ &= (Q\overset{o}{r})\overset{o}{q}^* \\ &= (\bar{Q}^T Q)\overset{o}{r}\end{aligned}$$

Where the matrix $\bar{Q}^T Q$ is given by:

$$\bar{Q}^T Q = \begin{bmatrix} \overset{o}{q}\cdot\overset{o}{q} & 0 & 0 & 0 \\ 0 & q_0^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & q_0^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_z) & q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

A few notes about this matrix:

- Since the scalar component of $\overset{o}{q}$ is zero, the first row and matrix of this column are sparse, as we can see above.
- If $\overset{o}{q}$ is a unit quaternion, the lower right 3×3 matrix of $\bar{Q}^T Q$ will be orthonormal (it is an orthonormal rotation matrix).

Let us look at more properties of this mapping $\overset{o}{r}' = \overset{o}{q}\overset{o}{r}\overset{o}{q}^*$:

1. **Scalar Component**: $r' = r(\overset{o}{q} \cdot \overset{o}{q})$
2. **Vector Component**: $\mathbf{r}' = (q^2 - \mathbf{q} \cdot \mathbf{q})\mathbf{r} + 2(\mathbf{q} \cdot \mathbf{r})\mathbf{q} + 2q(\mathbf{q} \times \mathbf{r})$
3. **Operator Preserves Dot Products**: $\overset{o'}{r} \cdot \overset{o'}{s} = \overset{o}{r} \cdot \overset{o}{s} \implies \mathbf{r}' \cdot \mathbf{s}' = \mathbf{r} \cdot \mathbf{s}$
4. **Operator Preserves Triple Products**: $(\overset{o'}{r} \cdot \overset{o'}{s}) \cdot \overset{o'}{t} = (\overset{o}{r} \cdot \overset{o}{s}) \cdot \overset{o}{t} \implies (\mathbf{r}' \cdot \mathbf{s}')\mathbf{t}' = (\mathbf{r} \cdot \mathbf{s}) \cdot \mathbf{t} \implies [\mathbf{r}'\ \mathbf{s}'\ \mathbf{t}'] = [\mathbf{r}\ \mathbf{s}\ \mathbf{t}]$
5. **Composition (of rotations!)**: Recall before that we could not easily compose rotations with our other rotation representations. Because of associativity, however, we can compose rotations simply through quaternion multiplication:

$$\overset{o}{p}(\overset{o}{q}\overset{o}{r}\overset{o}{q}^*)\overset{o}{p}^* = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{q}^*\overset{o}{p}^*) = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{p}\overset{o}{q})^*$$

I.e. if we denote the product of quaternions $\overset{o}{z} \triangleq \overset{o}{p}\overset{o}{q}$, then we can write this rotation operator as a single rotation:

$$\overset{o}{p}(\overset{o}{q}\overset{o}{r}\overset{o}{q}^*)\overset{o}{p}^* = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{q}^*\overset{o}{p}^*) = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{p}\overset{o}{q})^* = \overset{o}{z}\overset{o}{r}\overset{o}{z}^*$$

This ability to compose rotations is quite advantageous relative to many of the other representations of rotations we have seen before (orthonormal rotation matrices can achieve this as well).

17.7.1 Relation of Quaternion Rotation Operation to Rodrigues Formula

Let us see how this operator relates to the Rodrigues formula and the axis-angle representation of rotations:

$$\begin{aligned}\mathbf{r}' &= (q^2 - \mathbf{q} \cdot \mathbf{q})\mathbf{r} + 2(\mathbf{q} \cdot \mathbf{r})\mathbf{q} + 2q(\mathbf{q} \times \mathbf{r}) \\ &= (\cos\theta)\mathbf{r} + (1 - \cos\theta)(\hat{\boldsymbol{\omega}} \cdot \mathbf{r})\hat{\boldsymbol{\omega}} + (\sin\theta)(\hat{\boldsymbol{\omega}} \times \mathbf{r})\end{aligned}$$

We have that q is parallel to $\hat{\boldsymbol{\omega}}$. Note the following equalities:

- $(q^2 - \mathbf{q} \cdot \mathbf{q}) = \cos\theta$
- $2||\mathbf{q}||_2^2 = (1 - \cos\theta)$
- $2q||\mathbf{q}||_2^2 = \sin\left(\frac{\theta}{2}\right)$
- $q = \cos\left(\frac{\theta}{2}\right)$
- $||\mathbf{q}||_2^2 = \sin\left(\frac{\theta}{2}\right)$

From these statements of equality, we can conclude:

$$\overset{o}{q} = (\cos\big(\frac{\theta}{2}\big), \hat{\boldsymbol{\omega}}\sin\big(\frac{\theta}{2}\big))$$

A few notes on this:

- We see that both the scalar and vector components of the quaternion $\overset{o}{q}$ depend on the axis of rotation $\hat{\boldsymbol{\omega}}$ and the angle of rotation θ.
- The vector component of this quaternion is parallel to $\hat{\boldsymbol{\omega}}$.
- This representation is one way to represent a unit quaternion.
- Knowing the axis and angle of a rotation allows us to compute the quaternion.
- Note that $-\overset{o}{q}$ represents the same mapping as $\overset{o}{q}$ since:

$$(-\overset{o}{q})\overset{o}{r}(-\overset{o}{q}^*) = \overset{o}{q}\overset{o}{r}\overset{o}{q}^*$$

To build intuition with this quaternion rotation operator, one way we can conceptualize this is by considering that our space of rotations is a 3D sphere in 4D, and opposite points on this sphere represent the same rotation.

17.8 Applying Quaternion Rotation Operator to Photogrammetry

Now that we have specified this operator and its properties, we are ready to apply this to photogrammetry, specifically for **absolute orientation**. Let us briefly review our four main problems of photogrammetry:

1. **Absolute Orientation** (3D to 3D): Range Data
2. **Relative Orientation** (2D to 2D): Binocular Stereo
3. **Exterior Orientation** (3D to 2D): Passive Navigation
4. **Interior Orientation** (2D to 3D): Camera Calibration

We will start by applying this to our first problem, **absolute orientation**. Recall our goal with this photogrammetry problem is to find the 3D transformation between our coordinate systems.

17.8.1 Least Squares Approach to Find R

Recall our first attempt to solve for this transformation was done through least squares to solve for an optimal rotation matrix and a translation vector. While we showed in lecture 17 that we can solve for an optimal translation vector that depends on the rotation matrix R, we could not find a closed-form solution for the rotation matrix R. We review why:

$$\begin{aligned}\min +R\sum_{i=1}^{n}||\mathbf{e}_i||_2^2 &= \sum_{i=1}^{n}(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) \\ &= \sum_{i=1}^{n}||\mathbf{r}'_{r,i}||_2^2 - \sum_{i=1}^{n}(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) - \sum_{i=1}^{n}||\mathbf{r}'_{l,i}||_2^2\end{aligned}$$

Where the first of these terms is fixed, the second of these terms is to be maximized (since there is a negative sign in front of this term, this thereby minimizes the objective), and the third of these terms is fixed. Therefore, our rotation problem can be simplified to:

$$R^* = \min_R \sum_{i=1}^{n}||\mathbf{e}_i||_2^2 = -2\sum_{i=1}^{n}(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) \tag{51}$$

$$= \min_R(-2\sum_{i=1}^{n}\mathbf{r}'_{r,i} \cdot R(\mathbf{r}'_{l,i})) \tag{52}$$

$$= \max_R(\sum_{i=1}^{n} rb'_{r,i} \cdot R(\mathbf{r}'_{l,i})) \tag{53}$$

But since we are optimizing over an orthonormal rotation matrix R, we cannot simply take the derivative and set it equal to zero as we usually do for these least squares optimization problems. Though we can solve this as a Lagrangian optimization problem, specifying these constraints is difficult and makes for a much more difficult optimization problem. It turns out this a common problem in spacecraft attitude control. Let us see how we can use quaternions here!

17.8.2 Quaternion-based Optimization

Solving this with our quaternion operator above, and noting the following definitions:

- $\overset{o'}{r}_{l,i} = (0, \mathbf{r}'_{l,i})$
- $\overset{o'}{r}_{r,i} = (0, \mathbf{r}'_{l,i})$

Then we can solve for our optimal rotation by solving for quaternions instead:

$$\begin{aligned}
R^* &= \max_R \sum_{i=1}^{n} rb'_{r,i} \cdot R(\mathbf{r}'_{l,i}) \\
&= \max_{\overset{o}{q}, ||\overset{o}{q}||_2 = 1} \sum_{i=1}^{n} (\overset{o}{q}\overset{o'}{r}_{l,i}\overset{o*}{q}) \cdot \overset{o'}{r}_{r,i} \\
&= \max_{\overset{o}{q}, ||\overset{o}{q}||_2 = 1} \sum_{i=1}^{n} (\overset{o}{q}\overset{o'}{r}_{l,i}) \cdot (\overset{o'}{r}_{r,i}\overset{o}{q}) \\
&= \max_{\overset{o}{q}, ||\overset{o}{q}||_2 = 1} \sum_{i=1}^{n} (\bar{R}_{l,i}\overset{o}{q}) \cdot (R_{r,i}\overset{o}{q}) \\
&= \max_{\overset{o}{q}, ||\overset{o}{q}||_2 = 1} \overset{o}{q}^T \Big(\sum_{i=1}^{n} \bar{R}_{l,i}^T R_{r,i}\Big)\overset{o}{q} \quad \textbf{(Since } \overset{o}{q} \textbf{ does not depend on } i\textbf{)}
\end{aligned}$$

Where the term in the sum is a 4×4 matrix derived from point cloud measurements.

From here, we can solve for an optimal rotation quaternion through Lagrangian Optimization, with our objective given by:

$$\max_{\overset{o}{q}} \overset{o}{q}^T N \overset{o}{q}, \text{ subject to: } \overset{o}{q} \cdot \overset{o}{q} = 1, \; N \triangleq \sum_{i=1}^{n} \bar{R}_{l,i}^T R_{r,i}$$

Then written with the Lagrangian constraints this optimization problem becomes:

$$\max_{\overset{o}{q}} \overset{o}{q}^T N \overset{o}{q} + \lambda(1 - \overset{o}{q} \cdot \overset{o}{q})$$

Differentiating this expression w.r.t. $\overset{o}{q}$ and setting the result equal to zero yields the following first-order condition:

$$2N\overset{o}{q} - 2\lambda\overset{o}{q} = 0$$

It turns out that similarly to our inertia problem from the last lecture, the quaternion solution to this problem is the solution to an eigenvalue/eigenvector problem. Specifically, our solution is the eigenvector corresponding to the largest eigenvalue of a 4×4 real symmetric matrix N constructed from elements of the matrix given by a dyadic product of point cloud measurements from the left and righthand systems of coordinates:

$$M = \sum_{i=1}^{n} \mathbf{r}'_{l,i} \mathbf{r}'^{\,T}_{r,i}$$

This matrix M is an asymmetric 3×3 real matrix. A few other notes about this:

- The **eigenvalues** of this problem are the **Lagrange Multipliers** of this objective, and the eigenvectors are derived from the eiegenvectors of N, our matrix of observations.
- This analytic solution leads to/requires solving a quartic polynomial - fortunately, we have closed-form solutions of polynomials up to a quartic degree! Therefore, a closed-form solution exists. Specifically, the characteristic equation in this case takes the form:

  $$\lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0 = 0$$

  Because the matrix of point cloud measurements N is symmetric, this characteristic equation simplifies and we get the following coefficients:

  1. $c_3 = \text{tr}(N) = 0$
  2. $c_2 = -2\text{tr}(M^T M)$
  3. $c_1 = -8 \det |M|$
  4. $c_0 = \det |N|$

In addition to solving **absolute orientation** problems with quaternions, this approach has applications to other problems as well, such as:

- Relative Orientation (Binocular Stereo)
- Camera Calibration
- Manipulator Kinematics
- Manipulator Fingerprinting
- Spacecraft Dynamics

17.9 Desirable Properties of Quaternions

Next, let us look at what propreties we would like to obtain from using quaternions, and let us see how well these properties are satisfied:

- **The ability to rotate a vector or coordinate system**
  Yes!
- **Ability to compose rotations**
  Yes!
- **Intuitive, non-redundant representations**
  There is some redundancy (but note that this is solved by using unit quaternions!), and this representation is not the most intuitive.
- **Computational efficiency**
  We will discuss this in more detail further below.
- **Ability to Interpolate Orientations**
  Yes!
- **Ability to average over rotations**
  Yes!
- **Ability to take derivative w.r.t. rotation - e.g. for optimization and least squares**
  Yes!
- **Ability to sample rotations**
  Yes!
- **Notion of a space of rotations**
  Yes!

17.9.1 Computational Issues for Quaternions

One helpful metric we can compare different representations for rotation on is their computational efficiency. Below, we compare quaternions for rotation with orthonormal matrices for rotation in several important operations:

- **Operation: Composition of Rotations**: $\overset{\circ}{p}\overset{\circ}{q}$
  This is given by:

  $$\overset{\circ}{p}\overset{\circ}{q} = (p, \mathbf{p})(q, \mathbf{q}) = (pp - \mathbf{p}\mathbf{q}, p\mathbf{q} + q\mathbf{p} + \mathbf{p} \times \mathbf{q})$$

  Carrying this out naively requires **16 multiplications and 12 additions**.

  Compared with orthonormal matrices, composing quaternions is **faster** for this operation (orthonormal matrices require **27 multiplications and 18 additions**.

- **Operation: Rotating Vectors**: $\overset{\circ}{q}\overset{\circ}{r}\overset{\circ}{q}^*$
  This is given by:

  $$\begin{aligned}\overset{\circ}{q}\overset{\circ}{r}\overset{\circ}{q}^* \rightarrow & \mathbf{r}' = (q^2 - \mathbf{r} \cdot \mathbf{q})\mathbf{r} + 2(\mathbf{q} \cdot \mathbf{r})\mathbf{q} + 2q(\mathbf{q} \times \mathbf{r}) \\ & \mathbf{r}' = \mathbf{r} + 2q(\mathbf{q} \times \mathbf{r}) + 2\mathbf{q} \times (\mathbf{q} \times \mathbf{r}) \;\; \textbf{(More efficient implementation)}\end{aligned}$$

  Carrying this out naively requires **15 multiplications and 12 additions**.

  Compared with orthonormal matrices, composing quaternions is **slower** for this operation (orthonormal matrices require **9 multiplications and 6 additions**.

- **Operation: Renormalization** This operation is used when we compose many rotations, and the quaternion (if we are using a quaternion) or the orthonormal matrix is not quite an orthonormal matrix due to floating-point arithmetic. Since this operation requires matrix inversion (see below) for orthonormal matrices, it is much faster to carry out this operation with quaternions.

  Nearest Unit Quaternion: $\frac{\overset{o}{q}}{\sqrt{\overset{o}{q} \cdot \overset{o}{q}}}$

  Nearest Orthonormal Matrix: $M(M^T M)^{-\frac{1}{2}}$

17.9.2 Space of Rotations

We will conclude today's lecture by discussing the space of rotations, now that we have a representation for it:

- S^3 (the unit sphere $\in \mathbb{R}^3$) with antipodal points identified
- Projective space P^3
- **Sampling**: Sampling in this space can be done in regular and random intervals
- **Finite rotation groups**: These include the platonic solids with 12, 24, 60 elements - we can have rotation groups for (i) tetrahedron, (ii) hexahedron/octahedron, and (iii) dodecahedron/icosahedron
- **Finer-grade Sampling** can be achieved by sub-dividing the simplex in the rotation space
- If $\{\overset{o}{q}_i\}_{i=1}^N$ is a group, then so is $\{\overset{o}{q}'_i\}_{i=1}^N$, where $\overset{o}{q}'_i = \overset{o}{q}_0 \overset{o}{q}_i$.

17.10 References

1. Group, https://en.wikipedia.org/wiki/Group_(mathematics)
2. Manifold, https://en.wikipedia.org/wiki/Manifold

18 Lecture 19: Absolute Orientation in Closed Form, Outliers and Robustness, RANSAC

This lecture will continue our discussion of photogrammetry topics - specifically, covering more details with the problem of **absolute orientation**. We will also look at the effects of outliers on the robustness of closed-form absolute orientation, and how algorithms such as RANSAC can be leveraged as part of an absolute orientation, or, more generally, photogrammetry pipeline to improve the robustness of these systems.

18.1 Review: Absolute Orientation

Recall our four main problems of photogrammetry:

- **Absolute Orientation** $3D \longleftrightarrow 3D$
- **Relative Orientation** $2D \longleftrightarrow 2D$
- **Exterior Orientation** $2D \longleftrightarrow 3D$
- **Intrinsic Orientation** $3D \longleftrightarrow 2D$

In the last lecture, we saw that when solving **absolute orientation** problems, we are mostly interested in finding **transformations** (translation + rotation) between two coordinate systems, where these coordinate systems can correspond to objects or sensors moving in time (recall this is where we saw duality between objects and sensors).

Last time, we saw that one way we can find an optimal **transformation** between two coordinate systems in 3D is to decompose the optimal transformation into an optimal **translation** and an optimal **rotation**. We saw that we could solve for optimal translation in terms of rotation, and that we can mitigate the constraint issues with solving for an orthonormal rotation matrix by using **quaternions** to carry out rotation operations.

18.1.1 Rotation Operations

Relevant to our discussion of quaternions is identifying the critical operations that we will use for them (and for orthonormal rotation matrices). Most notably, these are:

1. **Composition of rotations**: $\mathring{p}\mathring{q} = (p, \mathbf{q})(q, \mathbf{q}) = (pq - \mathbf{q} \cdot \mathbf{q}, p\mathbf{q} + q\mathbf{q} + \mathbf{q} \times \mathbf{q})$
2. **Rotating vectors**: $\mathring{r}' = \mathring{q}\mathring{r}\mathring{q}^* = (q^2 - \mathbf{q} \cdot \mathbf{q})\mathbf{r} + 2(\mathbf{q} \cdot \mathbf{r})\mathbf{q} + 2q(\mathbf{q} \times \mathbf{r})$

Recall from the previous lecture that operation **(1) was faster** than using orthonormal rotation matrices, and operation **(2) was slower**.

18.1.2 Quaternion Representations: Axis-Angle Representation and Orthonormal Rotation Matrices

From our previous discussion, we saw that another way we can represent quaternions is through the axis-angle notation (known as the Rodrigues formula):

$$\mathbf{r}' = (\cos\theta)\mathbf{r} + (1 - \cos\theta)(\hat{\boldsymbol{\omega}} \cdot \mathbf{r})\hat{\boldsymbol{\omega}} + \sin\theta(\hat{\boldsymbol{\omega}} \times \mathbf{r})$$

Combining these equations from above, we have the following axis-angle representation:

$$\mathring{q} \Longleftrightarrow \hat{\boldsymbol{\omega}}, \theta, \;\; q = \cos\left(\frac{\theta}{2}\right), \; \mathbf{q} = \hat{\boldsymbol{\omega}} \sin\left(\frac{\theta}{2}\right) \implies \mathring{q} = \left(\cos\left(\frac{\theta}{2}\right), \hat{\boldsymbol{\omega}} \sin\left(\frac{\theta}{2}\right)\right)$$

We also saw that we can convert these quaternions to orthonormal rotation matrices. Recall that we can write our vector rotation operation as:

$$\mathring{q}\mathring{r}\mathring{q}^* = (\bar{Q}^T Q)\mathring{r}, \text{ where}$$

$$\bar{Q}^T Q = \begin{bmatrix} \mathring{q} \cdot \mathring{q} & 0 & 0 & 0 \\ 0 & q_0^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & q_0^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_z) & q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

The matrix $\bar{Q}^T Q$ has **skew-symmetric** components and **symmetric components**. This is useful for conversions. Given a **quaternion**, we can compute orthonormal rotations more easily. For instance, if we want an **axis** and **angle** representation, we can look at the lower right 3×3 submatrix, specifically its trace:

$$\text{Let } R = [\bar{Q}^T Q]_{3\times3,\text{lower}}, \text{ then :}$$

$$\begin{aligned}
\text{tr}(R) &= 3q_0^2 - (q_x^2 + q_y^2 + q_z^2) \\
&= 3\cos^2\left(\frac{\theta}{2}\right) - (\sin^2\left(\frac{\theta}{2}\right) \;\textbf{(Substituting our axis-angle representation)} \\
&- \cos^2\left(\frac{\theta}{2}\right) + \sin^2\left(\frac{\theta}{2}\right) - 1 \;\textbf{(Subtracting Zero)} \\
&\rightarrow 2\cos^2\left(\frac{\theta}{2}\right) - \sin^2\left(\frac{\theta}{2}\right) - 1 \\
&= 2\cos\theta - 1 \\
&\implies \cos\theta = \frac{1}{2}(\text{tr}(R) - 1)
\end{aligned}$$

While this is one way to get the angle (we can solve for θ through arccos of the expression above), it is not the best way to do so: we will encounter problems near $\theta \approx 0, \pi$. Instead, we can use the **off-diagonal** elements, which depend on $\sin\left(\frac{\theta}{2}\right)$ instead. Note that this works because at angles θ where $\cos\left(\frac{\theta}{2}\right)$ is "bad" (is extremely sensitive), $\sin\left(\frac{\theta}{2}\right)$ is "good" (not as sensitive), and vice versa.

18.2 Quaternion Transformations/Conversions

Next, let us focus on how we can convert between quaternions and orthonormal rotation matrices. Given a 3×3 orthonormal rotation matrix r, we can compute sums and obtain the following system of equations:

$$\begin{aligned}
1 + r_{11} + r_{22} + r_{33} &= 4q_0^2 \\
1 + r_{11} - r_{22} - r_{33} &= 4q_x^2 \\
1 - r_{11} + r_{22} - r_{33} &= 4q_y^2 \\
1 - r_{11} - r_{22} + r_{33} &= 4q_z^2
\end{aligned}$$

This equation can be solved by taking square roots, but due to the number of solutions (8 by Bezout's theorem, allowing for the flipped signs of quaternions, we should not use this set of equations alone to find the solution).

Instead, we can compute these equations, evaluate them, take the largest for numerical accuracy, arbitrarily select to use the positive version (since there is sign ambiguity with the signs of the quaternions), and solve for this. We will call this selected righthand side q_i.

For off-digaonals, which have **symmetric** and **non-symmetric** components, we derive the following equations:

$$\begin{aligned}
r_{32} - r_{23} &= 4q_0q_x \\
r_{13} - r_{31} &= 4q_0q_y \\
r_{21} - r_{12} &= 4q_0q_x \\
r_{21} + r_{12} &= 4q_xq_y \\
r_{32} + r_{23} &= 4q_yq_z \\
r_{13} + r_{31} &= 4q_zq_z
\end{aligned}$$

Adding/subtracting off-diagonals give us 6 relations, of which we only need 3 (since we have 1 relation from the diagonals). For instance, if we have $q_i = q_y$, then we pick off-diagonal relations involving q_y, and we solve the four equations given by:

$$\begin{aligned}
1 - r_{11} + r_{22} - r_{33} &= 4q_y^2 \\
r_{13} - r_{31} &= 4q_0q_y \\
r_{32} + r_{23} &= 4q_yq_z \\
r_{13} + r_{31} &= 4q_zq_z
\end{aligned}$$

This system of four equations gives us a direct way of going from quaternions to an orthonormal rotation matrix. Note that this could be 9 numbers that could be noisy, and we want to make sure we have best fits.

18.3 Transformations: Incorporating Scale

Thus far, for our problem of absolute orientation, we have considered **transformations** between two coordinate systems of being composed of **translation** and **rotation**. This is often sufficient, but in some applications and domains, such as satellite imaging fr topographic reconstruction, we may be able to better describe these transformations taking account not only **translation** and **rotation**, but also **scaling**.

Taking scaling into account, we can write the relationship between two point clouds corresponding to two different coordinate systems as:

$$\mathbf{r}'_r = sR(\mathbf{r}'_l)$$

Where rotation is again given by $R \in \mathbf{SO}(3)$, and the scaling factor is given by $s \in \mathbb{R}^+$ (where $\mathbb{R}^+ \triangleq \{x : x \in \mathbb{R}\ , x > 0\}$. Recall that $\mathbf{r}'_r$ and $\mathbf{r}'_l$ are the centroid-subtracted variants of the point clouds in both frames of reference.

18.3.1 Solving for Scaling Using Least Squares: Asymmetric Case

As we did before, we can write this as a least-squares problem over the scaling parameter s:

$$\min_s \sum_{i=1}^{n} ||\mathbf{r}'_{r,i} - sR(\mathbf{r}'_{l,i})||_2^2$$

As we did for translation and rotation, we can solve for an optimal scaling parameter:

$$\begin{aligned}
s^* &= \arg\min_s \sum_{i=1}^n ||\mathbf{r}'_{r,i} - sR(\mathbf{r}'_{l,i})||_2^2 \\
&= \arg\min_s \sum_{i=1}^n \Big(||\mathbf{r}'_{r,i}||_2^2\Big) - 2s\sum_{i=1}^n \Big(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\Big) + s^2\sum_{i=1}^n ||R(\mathbf{r}'_{l,i})||_2^2 \\
&= \arg\min_s \sum_{i=1}^n \Big(||\mathbf{r}'_{r,i}||_2^2\Big) - 2s\sum_{i=1}^n \Big(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\Big) + s^2\sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2 \quad \textbf{(Rotation preserves vector lengths)}
\end{aligned}$$

Next, let us define the following terms:

1. $s_r \triangleq \sum_{i=1}^n \Big(||\mathbf{r}'_{r,i}||_2^2\Big)$
2. $D \triangleq \sum_{i=1}^n \Big(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\Big)$
3. $s_l \triangleq \sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2$

Then we can write this objective for the optimal scaling factor s^* as:

$$s^* = \arg\min_s \{J(s) \triangleq s_r - 2sD + s^2 s_l\}$$

Since this is an unconstrained optimization problem, we can solve this by taking the derivative w.r.t. s and setting it equal to 0:

$$\begin{aligned}
\frac{dJ(s)}{ds} &= \frac{d}{ds}\Big(s_r - 2sD + s^2 s_l\Big) = 0 \\
&= -2D + s^2 s_l = 0 \implies s = \frac{D}{s_l}
\end{aligned}$$

As we also saw with rotation, this does not give us an exact answer without finding the orthonormal matrix R, but now we are able to remove scale factor and back-solve for it later using our optimal rotation.

18.3.2 Issues with Symmetry

Symmetry question: What if instead of going from the left coordinate system to the right one, we decided to go from right to left? In theory, this should be possible: we should be able to do this simply by **negating translation** and **inverting our rotation and scaling terms**. But in general, doing this in practice with our OLS approach above does not lead to $s_{\text{inverse}} = \frac{1}{s}$ - i.e. inverting the optimal scale factor does not give us the scale factor for the reverse problem.

Intuitively, this is the case because the version of OLS we used above "cheats" and tries to minimize error by shriking the scale by more than it should be shrunk. This occurs because it brings the points closer together, thereby minimizing, on average, the error term. Let us look at an alternative formulation for our error term that accounts for this optimization phenomenon.

18.3.3 Solving for Scaling Using Least Squares: Symmetric Case

Let us instead write our objective as:

$$\mathbf{e}_i = \frac{1}{\sqrt{s}}\mathbf{r}'_{r,i} = \sqrt{s}R(\mathbf{r}'_{l,i})$$

Then we can write our objective and optimization problem over scale as:

$$\begin{aligned}
s^* &= \arg\min_s \sum_{i=1}^n ||\frac{1}{\sqrt{s}}\mathbf{r}'_{r,i} - \sqrt{s}R(\mathbf{r}'_{l,i})||_2^2 \\
&= \arg\min_s \sum_{i=1}^n \Big(\frac{1}{s}||\mathbf{r}'_{r,i}||_2^2\Big) - 2\sum_{i=1}^n \Big(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\Big) + s\sum_{i=1}^n ||R(\mathbf{r}'_{l,i})||_2^2 \\
&= \arg\min_s \frac{1}{s}\sum_{i=1}^n \Big(||\mathbf{r}'_{r,i}||_2^2\Big) - 2\sum_{i=1}^n \Big(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\Big) + s\sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2 \quad \textbf{(Rotation preserves vector lengths)}
\end{aligned}$$

We then take the same definitions for these terms that we did above:

1. $s_r \triangleq \sum_{i=1}^n \left(||\mathbf{r}'_{r,i}||_2^2\right)$
2. $D \triangleq \sum_{i=1}^n \left(\mathbf{r}'_{r,i} R(\mathbf{r}'_{l,i})\right)$
3. $s_l \triangleq \sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2$

Then, as we did for the asymmetric OLS case, we can write this objective for the optimal scaling factor s^* as:

$$s^* = \arg\min_s \{J(s) \triangleq \frac{1}{s} s_r - 2D + s s_l\}$$

Since this is an unconstrained optimization problem, we can solve this by taking the derivative w.r.t. s and setting it equal to 0:

$$\begin{aligned}\frac{dJ(s)}{ds} &= \frac{d}{ds}\left(\frac{1}{s} s_r - 2D + s s_l\right) = 0 \\ &= -\frac{1}{s^2} s_r + s_l = 0 \implies s^2 = \frac{s_l}{s_r}\end{aligned}$$

Therefore, we can see that going in the reverse direction preserves this inverse (you can verify this mathematically and intuitively by simply setting $\mathbf{r}'_{r,i} \leftrightarrow \mathbf{r}'_{l,i} \; \forall \; i \in \{1, ..., n\}$ and noting that you will get $s^2_{\text{inverse}} = \frac{s_r}{s_l}$). Since this method better preserves symmetry, it is preferred.

Intuition: Since s no longer depends on correspondences (matches between points in the left and right point clouds), then the scale simply becomes the ratio of the point cloud sizes in both coordinate systems (note that s_l and s_r correspond to the summed vector lengths of the centroid-subtracted point clouds, which means they reflect the variance/spread/size of the point cloud in their respective coordinate systems.

We can deal with translation and rotation in a correspondence-free way, while also allowing for us to decouple rotation. Let us also look at solving rotation, which is covered in the next section.

18.4 Solving for Optimal Rotation in Absolute Orientation

Recall for rotation (see lecture 18 for details) that we switched from optimizing over orthonormal rotation matrices to optimizing over quaternions due to the lessened number of optimization constraints that we must adhere to. With our quaternion optimization formulation, our problem becomes:

$$\max_{\mathring{q}} \mathring{q}^T N \mathring{q}, \;\; \text{subject to} \;\; \mathring{q}^T \mathring{q} = 1$$

If this were an unconstrained optimization problem, we could solve by taking the derivative of this objective w.r.t. our quaternion $\mathring{q}$ and setting it equal to zero. Note the following helpful identities with matrix and vector calculus:

1. $\frac{d}{d\mathbf{a}}(\mathbf{a} \cdot \mathbf{b}) = \mathbf{b}$
2. $\frac{d}{d\mathbf{a}}(\mathbf{a}^T M \mathbf{b}) = 2M\mathbf{b}$

However, since we are working with quaternions, we must take this constraint into account. We saw in lecture 18 that we did this with using **Lagrange Multiplier** - in this lecture it is also possible to take this specific kind of vector length constraint into account using **Rayleigh Quotients**.

What are Rayleigh Quotients? The intuitive idea behind them: How do I prevent my parameters from becoming too large) positive or negative) or too small (zero)? We can accomplish this by dividing our objective by our parameters, in this case our constraint. In this case, with the Rayleigh Quotient taken into account, our objective becomes:

$$\frac{\mathring{q}^T N \mathring{q}}{\mathring{q}^T \mathring{q}} \quad \left(\textbf{Recall that } N \triangleq \sum_{i=1}^n R_{l,i}^T R_{r,i}\right)$$

How do we solve this? Since this is now an unconstrained optimization problem, we can solve this simply using the rules of calculus:

$$\begin{aligned}
J(\mathring{q}) \triangleq&= \frac{\mathring{q}^T N\mathring{q}}{\mathring{q}^T \mathring{q}} \\
\frac{dJ(\mathring{q})}{d\mathring{q}} &= \frac{d}{d\mathring{q}} \frac{\mathring{q}^T N\mathring{q}}{\mathring{q}^T \mathring{q}} = 0 \\
&= \frac{\frac{d}{d\mathring{q}}(\mathring{q}^T N\mathring{q})\mathring{q}^T \mathring{q} - \mathring{q}^T N\mathring{q}\frac{d}{d\mathring{q}}(\mathring{q}^T \mathring{q})}{(\mathring{q}^T \mathring{q})^2} = 0 \\
&= \frac{2N\mathring{q}}{\mathring{q}^T \mathring{q}} - \frac{2\mathring{q}}{(\mathring{q}^T \mathring{q})^2}(\mathring{q}^T N\mathring{q}) = 0
\end{aligned}$$

From here, we can write this first order condition result as:

$$N\mathring{q} = \frac{\mathring{q}^T N\mathring{q}}{\mathring{q}^T \mathring{q}}\mathring{q}$$

Note that $\frac{\mathring{q}^T N\mathring{q}}{\mathring{q}^T \mathring{q}} \in \mathbb{R}$ (this is our objective). Therefore, we are searching for a vector of quaternion coefficients such applying the rotation matrix to this vector simply produces a scalar multiple of it - i.e. an eigenvector of the matrix N. Letting $\lambda \triangleq \frac{\mathring{q}^T N\mathring{q}}{\mathring{q}^T \mathring{q}}$, then this simply becomes $N\mathring{q} = \lambda\mathring{q}$. Since this optimization problem is a maximization problem, this means that we can pick the **eigenvector** of N that corresponds to the **largest eigenvalue** (which in turn maximizes the objective consisting of the Rayleigh quotient $\frac{\mathring{q}^T N\mathring{q}}{\mathring{q}^T \mathring{q}}$, which is the eigenvalue.

Even though this quaternion-based optimization approach requires taking this Rayleigh Quotient into account, it is much easier to do this optimization than to solve for orthonormal matrices, which either require a complex Lagrangian (if we solve with Lagrange multipliers) or an SVD decomposition from Euclidean space to the $\mathbf{SO}(3)$ group (which also happens to be a manifold).

This approach raises a few questions:

- How many correspondences are needed to solve these optimization problems? Recall a correspondence is when we say two 3D points in different coordinate systems belong to the same point in 3D space, i.e. the same point observed in two separate frames of reference.
- When do these approaches fail?

We cover these two questions in more detail below.

18.4.1 How Many Correspondences Do We Need?

Recall that we are looking for 6 parameters (for translation and rotation) or 7 parameters (for translation, rotation, and scaling). Since each correspondence provides three constraints (since we equate the 3-dimensional coordinates of two 3D points in space), assuming non-redundancy, then we can solve this with two correspondences.

Let us start with two correspondences: if we have two objects corresponding to the correspondences of points in the 3D world, then if we rotate one object about axis, we find this does not work, i.e. we have an additional degree of freedom. Note that the distance between correspondences is fixed.

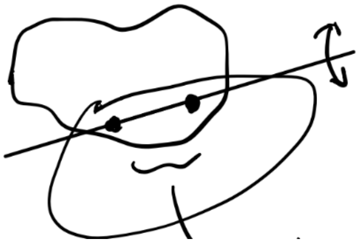


Figure 33: Using two correspondences leads to only satisfying 5 of the 6 needed constraints to solve for translation and rotation between two point clouds.

Because we have one more degree of freedom, this accounts for only 5 of the 6 needed constraints to solve for translation and rotation, so we need to have **at least 3 correspondences**.

With 3 correspondences, we get 9 constraints, which leads to some redundancies. We can add more constraints by incorporating **scaling** and generalizing the allowable transformations between the two coordinate systems to be the **generalized linear transformation** - this corresponds to allowing non-orthonormal rotation transformations. This approach gives us 9 unknowns!

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \end{bmatrix}$$

But we also have to account for translation, which gives us another 3 unknowns, giving us 12 in total and therefore requiring at least 4 non-redundant correspondences in order to compute the full general linear transformation. Note that this doesn't have any constraints as well!

On a practical note, this is often not needed, especially for finding the absolute orientation between two cameras, because oftentimes the only transformations that need to be considered due to the design constraints of the system (e.g. an autonomous car with two lidar systems, one on each side) are **translation** and **rotation**.

18.4.2 When do These Approaches Fail?

These approaches can fail when we do not have enough correspondences. In this case, the matrix N will become singular, and will produce eigenvalues of zero. A more interesting failure case occurs when the points of one or both of the point clouds are **coplanar**. Recall that we solve for the eigenvalues of a matrix (N in this case) using the characteristic equation given by:

$$\textbf{Characteristic Equation}: \ \det|N - \lambda \mathbf{I}| = 0$$
$$\textbf{Leads to 4th-order polynomial}: \ \lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0 = 0$$

Recall that our matrix N composed of the data has some special properties:

1. $c_3 = \text{tr}(N) = 0$ (This is actually a great feature, since usually the first step in solving 4th-order polynomial systems is eliminating the third-order term).
2. $c_2 = 2\text{tr}(M^T M)$, where M is defined as the sum of dyadic products between the points in the point clouds:

$$M \triangleq \left(\sum_{i=1}^{n} \mathbf{r}'_{l,i}\mathbf{r}'^{\,T}_{r,i}\right) \in \mathbb{R}^{3\times 3}$$

3. $c_1 = 8\det|M|$
4. $c_0 = \det|N|$

What happens if $\det|M| = 0$, i.e. the matrix M is singular? Then using the formulas above we must have that the coefficient $c_1 = 0$. Then this problem reduces to:

$$\lambda^4 + c_2\lambda^2 + c_0 = 0$$

This case corresponds to a special geometric case/configuration of the point clouds - specifically, when points are **coplanar**.

18.4.3 What Happens When Points are Coplanar?

When points are coplanar, we have that the matrix N, composed of the sum of dyadic products between the correspondences in the two point clouds, will be singular.

To describe this plane in space, we need only find a normal vector $\hat{\mathbf{n}}$ that is orthogonal to all points in the point cloud - i.e. the component of each point in the point cloud in the $\hat{\mathbf{n}}$ direction is 0. Therefore, we can describe the plane by the equation:

$$\mathbf{r}'_{r,i} \cdot \hat{\mathbf{n}} = 0 \; \forall \; i \in \{1, ..., n\}$$

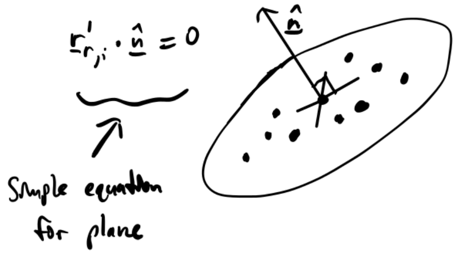

$\underbrace{\mathbf{r}'_{r,i} \cdot \hat{\mathbf{n}} = 0}$
Simple equation for plane
$\hat{\mathbf{n}}$

Figure 34: A coplanar point cloud can be described entirely by a surface normal of the plane $\hat{\mathbf{n}}$.

Note: In the absence of measurement noise, if one point cloud is coplanar, the the other point cloud must be as well (assuming that the transformation between the point clouds is a linear transformation). This does not necessarily hold when measurement noise is introduced.

Recall that our matrix M, which we used above to compute the coefficients of the characteristic polynomial describing this system, is given by:

$$M \triangleq \sum_{i=1}^{n} \mathbf{r}'_{r,i}\mathbf{r}'_{l,i}{}^{T}$$

Then, rewriting our equation for the plane above, we have:

$$\begin{aligned}
\mathbf{r}'_{r,i} \cdot \hat{\mathbf{n}} = 0 \implies M\hat{\mathbf{n}} &= \left(\sum_{i=1}^{n} \mathbf{r}'_{r,i}\mathbf{r}'_{l,i}{}^{T}\right)\hat{\mathbf{n}} \\
&= \sum_{i=1}^{n} \mathbf{r}'_{r,i}\mathbf{r}'_{l,i}{}^{T}\hat{\mathbf{n}} \\
&= \sum_{i=1}^{n} \mathbf{r}'_{r,i}0 \\
&= 0
\end{aligned}$$

Therefore, when a point cloud is coplanar, the **null space** of M is non-trivial (it is given by at least Span($\{\hat{\mathbf{n}}\}$), and therefore M is singular. Recall that a matrix $M \in \mathbb{R}^{n \times d}$ is singular if $\exists \, \mathbf{x} \in \mathbb{R}^d, \mathbf{x} \neq \mathbf{0}$ such that $M\mathbf{x} = \mathbf{0}$, i.e. the matrix has a non-trivial null space.

18.4.4 What Happens When Both Coordinate Systems Are Coplanar

Visually, when two point clouds are coplanar, we have:

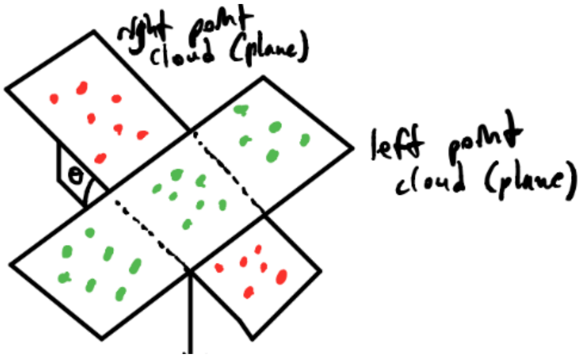


Figure 35: Two coplanar point clouds. This particular configuration allows us to estimate rotation in two simpler steps.

In this case, we can actually decompose finding the right rotation into two simpler steps!

1. Rotate one plane so it lies on top of the other plane. We can read off the **axis** and **angle** from the unit normal vectors of these two planes describing the coplanarity of these point clouds, given respectively by $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$:

   - **Axis**: We can find the axis by noting that the axis vector will be parallel to the cross product of $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$, simply scaled to a unit vector:

   $$\hat{\boldsymbol{\omega}} = \frac{\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2}{||\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}||_2}$$

   - **Angle**: We can also solve for the angle using the two unit vectors $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$:

   $$\cos\theta = \hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2$$
   $$\sin\theta = \hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2$$
   $$\theta = \arctan 2\left(\frac{\sin\theta}{\cos\theta}\right)$$

   We now have an axis angle representation for rotation between these two planes, and since the points describe each of the respective point clouds, therefore, a rotation between the two point clouds! We can convert this axis-angle representation into a quaternion with the formula we have seen before:

   $$\overset{o}{q} = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\boldsymbol{\omega}}\right)$$

2. Perform an in-plane rotation. Now that we have the quaternion representing the rotation between these two planes, we can orient two planes on top of each other, and then just solve a 2D least-squares problem to solve for our in-place rotation.

With these steps, we have a rotation between the two point clouds!

18.5 Robustness

In many methods in this course, we have looked at the use of **Least Squares** methods to solve for estimates in the presence of noise and many data points. Least squares produces an unbiased, minimum-variance estimate if (along with a few other assumptions) the dataset/measurement noise is Gaussian (Gauss-Markov Theorem) [1]. But what if the measurement noise is non-Gaussian? How do we deal with outliers in this case?

It turns out that **Least Squares** methods are not robust to outliers. One alternative approach is to use absolute error instead. Unfortunately, however, using absolute error does not have a closed-form solution. What are our other options for dealing with outliers? One particularly useful alternative is **RANSAC**.

RANSAC, or **Random Sample Consensus**, is an algorithm for robust estimation with **least squares** in the presence of outliers in the measurements. The goal is to find a least squares estimate that includes, within a certain threshold band, a set of inliers corresponding to the inliers of the dataset, and all other points outside of this threshold bands as outliers. The high-level steps of RANSAC are as follows:

1. **Random Sample**: Sample the minimum number of points needed to fix the transformation (e.g. 3 for absolute orientation; some recommend taking more).

2. **Fit random sample of points**: Usually this involves running least squares on the sample selected. This fits a line (or hyperplane, in higher dimensions), to the randomly-sampled points.

3. **Check Fit**: Evaluate the line fitted on the randomly-selected subsample on the rest of the data, and determine if the fit produces an estimate that is consistent with the "inliers" of your dataset. If the fit is good enough accept it, and if it is not, run another sample. Note that this step has different variations - rather than just immediately terminating once you have a good fit, you can run this many times, and then take the best fit from that.

   Furthermore, for step 3, we threshold the band from the fitted line/hyperplane to determine which points of the dataset are inliers, and which are outliers (see figure below). This band is usually given by a 2ϵ band around the fitted line/hyperplane. Typically, this parameter is determined by knowing some intrinsic structure about the dataset.

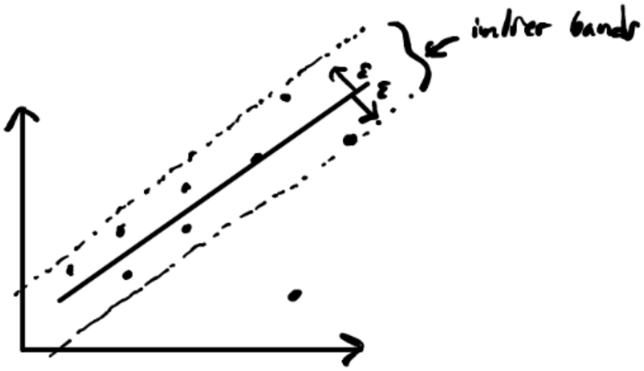


Figure 36: To evaluate the goodness of fit of our sampled points, as well as to determine inliers and outliers from our dataset, we have a 2ϵ thick band centered around the fitted line.

Another interpretation of RANSAC: counting the "maximimally-occupied" cell in Hough transform parameter space! Another way to find the best fitting line that is robust to outliers:

1. Repeatedly sample subsets from the dataset/set of measurements, and fit these subsets of points using least squares estimates.

2. For each fit, map the points to a discretized Hough transform parameter space, and have an accumulator array that keeps track of how often a set of parameters falls into a discretized cell. Each time a set of parameters falls into a discretized cell, increment it by one.

3. After N sets of random samples/least squares fits, pick the parameters corresponding to the cell that is "maximally-occupied", aka has been incremented the most number of times! Take this as your outlier-robust estimate.

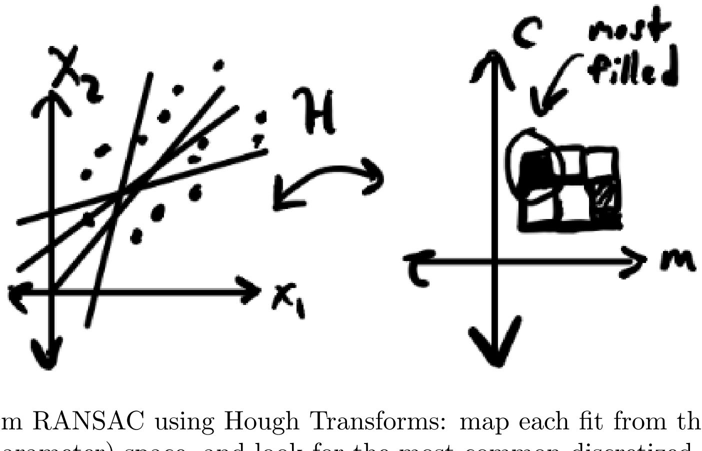


Figure 37: Another way to perform RANSAC using Hough Transforms: map each fit from the subsamples of measurements to a discretized Hough Transform (parameter) space, and look for the most common discretized cell in parameter space to use for an outlier-robust least-squares estimate.

18.6 Sampling Space of Rotations

Next, we will shift gears to discuss the **sampling space of rotations**.

Why are we interested in this space? Many orientation problems we have studied so far do not have a closed-form solution and may require sampling. How do we sample from the space of rotations?

18.6.1 Initial Procedure: Sampling from a Sphere

Let us start by **sampling from a unit sphere** (we will start in 3D, aiming eventually for 4D, but our framework will generalize easily from 3D to 4D). Why a sphere? Recall that we are interested in sampling for the coefficients of a unit quaternion $\overset{o}{q} = (q_0, q_x, q_y, q_z), ||\overset{o}{q}||_2^2 = 1$.

One way to sample from a sphere is with latitude and longitude, given by (θ_i, ϕ_i), respectively. The problem with this approach, however, is that we sample points that are close together at the poles. Alternatively, we can generate random longitude θ_i and ϕ_i, where:

- $-\frac{\pi}{2} \leq \theta_i \leq \frac{\pi}{2} \; \forall \; i$
- $-\pi \leq \phi_i \leq \pi \; \forall \; i$

But this approach suffers from the same problem - it samples too strongly from the poles. Can we do better?

18.6.2 Improved Approach: Sampling from a Cube

To achieve more uniform sampling from a sphere, what if we sampled from a unit cube (where the origin is given the center of the cube), and map the sampled points to an enscribed unit sphere within the cube?

Idea: Map all points (both inside the sphere and outside the sphere/inside the cube) onto the sphere by connecting a line from the origin to the sampled point, and finding the point where this line intersects the sphere.

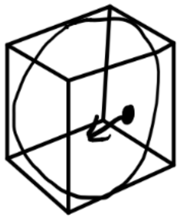


Figure 38: Sampling from a sphere by sampling from a cube and projecting it back to the sphere.

Problem with this approach: This approach disproportionately samples more highly on/in the direction of the cube's edges. We could use sampling weights to mitigate this effect, but better yet, we can simply discard any samples that fall outside the sphere. To avoid numerical issues, it is also best to discard points very close to the sphere.

Generalization to 4D: As we mentioned above, our goal is to generalize this from 3D to 4D. Cubes and spheres simply become 4-dimensional - enabling us to sample quaternions.

18.6.3 Sampling From Spheres Using Regular and Semi-Regular Polyhedra

We saw the approach above requires discarding samples, which is computationally-undesirable because it means we will probabilistically have to generate more samples than if we were able to sample from the sphere alone. To make this more efficient, let us consider shapes that form a "tighter fit" around the sphere - for instance: **polyhedra**! Some polyhedra we can use:

- **Tetrahedra** (4 faces)
- **Hexahedra** (6 faces)
- **Octahedra** (8 faces)
- **Dodecahedra** (12 faces)

- **Icosahedra** (20 faces)

These polyhedra are also known as the **regular solids**.

As we did for the cube, we can do the same for polyhedra: to sample from the sphere, we can sample from the polyhedra, and then **project** onto the point on the sphere that intersects the line from the origin to the sampled point on the polyhedra. From this, we get **great circles** from the edges of these polyhedra on the sphere when we project.

Fun fact: Soccer balls have 32 faces! More related to geometry: soccer balls are part of a group of **semi-regular** solids, specifically an **icosadodecahedron**.

18.6.4 Sampling in 4D: Rotation Quaternions and Products of Quaternions

Now we are ready to apply these shapes for sampling quaternions in 4D. Recall that our goal with this sampling task is to find the rotation between two point clouds, e.g. two objects. We need a uniform way of sampling this spae. We can start with the hexahedron. Below are 10 elementary rotations we use (recall that a quaternion is given in axis-angle notation by $\overset{o}{q} = (\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\boldsymbol{\omega}})$):

1. **Identity rotation**: $\overset{o}{q} = (1, 0)$
2. π **about** $\hat{\mathbf{x}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\mathbf{x}}) = (0, \hat{\mathbf{x}})$
3. π **about** $\hat{\mathbf{y}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\mathbf{y}}) = (0, \hat{\mathbf{y}})$
4. π **about** $\hat{\mathbf{z}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\mathbf{z}}) = (0, \hat{\mathbf{z}})$
5. $\boldsymbol{\frac{\pi}{2}}$ **about** $\hat{\mathbf{x}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\hat{\mathbf{x}}) = \frac{1}{\sqrt{2}}(1, \hat{\mathbf{x}})$
6. $\boldsymbol{\frac{\pi}{2}}$ **about** $\hat{\mathbf{y}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\hat{\mathbf{y}}) = \frac{1}{\sqrt{2}}(1, \hat{\mathbf{y}})$
7. $\boldsymbol{\frac{\pi}{2}}$ **about** $\hat{\mathbf{z}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\hat{\mathbf{z}}) = \frac{1}{\sqrt{2}}(1, \hat{\mathbf{z}})$
8. $-\boldsymbol{\frac{\pi}{2}}$ **about** $\hat{\mathbf{x}}$: $\overset{o}{q} = (\cos\left(-\frac{\pi}{4}\right), \sin\left(-\frac{\pi}{4}\right)\hat{\mathbf{x}}) = \frac{1}{\sqrt{2}}(1, -\hat{\mathbf{x}})$
9. $-\boldsymbol{\frac{\pi}{2}}$ **about** $\hat{\mathbf{y}}$: $\overset{o}{q} = (\cos\left(-\frac{\pi}{4}\right), \sin\left(-\frac{\pi}{4}\right)\hat{\mathbf{y}}) = \frac{1}{\sqrt{2}}(1, -\hat{\mathbf{y}})$
10. $-\boldsymbol{\frac{\pi}{2}}$ **about** $\hat{\mathbf{z}}$: $\overset{o}{q} = (\cos\left(-\frac{\pi}{4}\right), \sin\left(-\frac{\pi}{4}\right)\hat{\mathbf{z}}) = \frac{1}{\sqrt{2}}(1, -\hat{\mathbf{z}})$

These 10 rotations by themselves give us 10 ways to sample the rotation space. How can we construct more samples? We can do so by **taking quaternion products**, specifically, products of these 10 quaternions above. Let us look at just a couple of these products:

1. $(0, \hat{\mathbf{x}})(0, \hat{\mathbf{y}})$:

$$\begin{aligned}(0, \hat{\mathbf{x}})(0, \hat{\mathbf{y}}) &= (0 - \hat{\mathbf{x}} \cdot \hat{\mathbf{y}}, 0\hat{\mathbf{x}} + 0\hat{\mathbf{y}} + \hat{\mathbf{x}} \times \hat{\mathbf{y}}) \\ &= (-\hat{\mathbf{x}} \cdot \hat{\mathbf{y}}, \hat{\mathbf{x}} \times \hat{\mathbf{y}}) \\ &= (0, \hat{\mathbf{z}})\end{aligned}$$

   We see that this simply produces the third axis, as we would expect. This does not give us a new rotation to sample from. Next, let us look at one that does.

2. $\frac{1}{\sqrt{2}}(1, \hat{\mathbf{x}})\frac{1}{\sqrt{2}}(1, \hat{\mathbf{y}})$:

$$\begin{aligned}\frac{1}{\sqrt{2}}(1, \hat{\mathbf{x}})\frac{1}{\sqrt{2}}(1, \hat{\mathbf{y}}) &= \frac{1}{2}(1 - \hat{\mathbf{x}} \cdot \hat{\mathbf{y}}, \hat{\mathbf{y}} + \hat{\mathbf{x}} + \hat{\mathbf{x}} \times \hat{\mathbf{y}}) \\ &= \frac{1}{2}(1, \hat{\mathbf{x}} + \hat{\mathbf{y}} + \hat{\mathbf{x}} \times \hat{\mathbf{y}})\end{aligned}$$

   This yields the following axis-angle representation:

   - Axis: $\frac{1}{\sqrt{3}}(1\ 1\ 1)$

- Angle: $\cos\left(\frac{\theta}{2}\right) = \frac{1}{2} \implies \frac{\theta}{2} = \frac{\pi}{3} \implies \theta = \frac{2\pi}{3}$

Therefore, we have produced a new rotation that we can sample from!

These are just a few of the pairwise quaternion products we can compute. It turns out that these pairwise quaternion products produce a total of **24 new rotations** from the original 10 rotations. These are helpful for achieving greater sampling granularity when sampling the rotation space.

18.7 References

1. Gauss-Markov Theorem, https://en.wikipedia.org/wiki/Gauss%E2%80%93Markov_theorem

19 Lecture 20: Space of Rotations, Regular Tessellations, Critical Surfaces in Motion Vision and Binocular Stereo

In this lecture, we will transition from solving problems of **absolute rotation** (which, if you recall, finds the transformation between two 3D coordinate systems) into **relative orientation**, which finds the transformation between two 2D coordinate systems. We will start by covering the problem of **binocular stereo**, and in the process talk about tessellations from solids, critical surfaces.

19.1 Tessellations of Regular Solids

As a brief review from the last lecture, recall that we saw we can encode rotations as 4D points on the unit sphere, where the coordinates of these 4D points correspond to our coefficients for the unit quaternion.

What are tessellations? "A filling or tessellations of a flat surface is the covering of a plane using one or more geometric shapes (polygons)." [1]. **Tessellations** of the surface of the sphere can be based on **platonic solids**, with 4, 6, 8, 12, and 20 faces. Each of the tessellations from the platonic solids results in equal area projections on the sphere, but the division is somewhat coarse.

For **greater granularity**, we can look at the 14 **Archimedean solids**. This allows for having multiple polygons in each polyhedra (e.g. squares and triangles), resulting in unequal area in the tessellations on the unit sphere.

Related, we are also interested in the **rotation groups** (recall **groups** are mathematical sets that obey certain algebras, e.g. the **Special Orthogonal** group) of these regular polyhedra:

- 12 elements in rotation group for **tetrahedron**.
- 24 elements in rotation group for **hexahedron**.
- 60 elements in rotation group for **dodecahedron**.
- The **octahedron** is the dual of the **cube** and therefore occupies the same rotation group as it.
- The **icosahedron** is the dual of the **dodecahedron** and therefore occupies the same rotation group as it.

A few other notes on tessellations:

- One frequently-used method for creating tessellations is to divide each face into many **triangular** or **hexagonal** areas.
- Histograms can be created in tessellations in planes by taking square sub-divisions of the region.
- **Hexagonal tessellations** are a dual of **triangular tessellations**.

19.2 Critical Surfaces

What are Critical Surfaces? Critical surfaces are geometric surfaces - specifically, hyperboloids of one sheet, that lead to ambiguity in the solution space for relative orientation problems.

Why are Critical Surfaces important? Critical surfaces can negatively impact the performance of relative orientation systems, and understanding their geometry can enable us to avoid using strategies that rely on these types of surfaces in order to find the 2D transformation, for instance, between two cameras.

We will discuss more about these at the end of today's lecture. For now, let's introduce **quadrics** - geometric shapes/surfaces defined by second-order equations in a 3D Cartesian coordinate system:

1. **Ellipsoid**: $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$

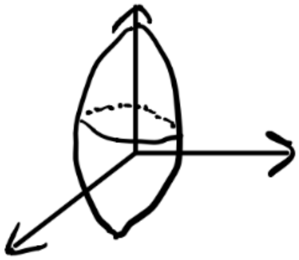


Figure 39: 3D geometric depiction of an ellipsoid, one member of the quadric family.

2. **Sphere**: $x^2 + y^2 + z^2 = 1$ (or more generally, $= r \in \mathbb{R}$)

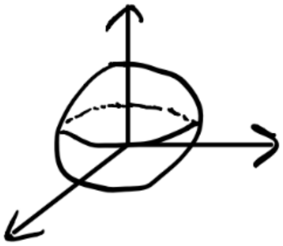


Figure 40: 3D geometric depiction of a sphere, another member of the quadric family and a special case of the ellipse.

3. **Hyperboloid of One Sheet**: $x^2 + y^2 - z^2 = 1$

This quadric surface is **ruled**: we can embed straight lines in the surface, despite its quadric function structure.

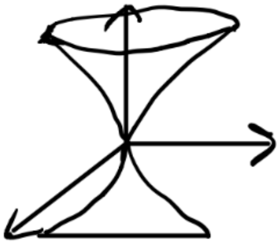


Figure 41: 3D geometric depiction of a hyperboloid of one sheet, another member of the quadric family and a special case of the ellipse.

4. **Hyperboloid of Two Sheets**: $x^2 - y^2 - z^2 = 1$

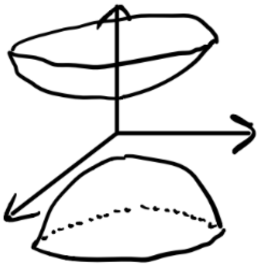


Figure 42: 3D geometric depiction of a hyperboloid of one sheet, another member of the quadric family and a special case of the ellipse.

5. **Cone**: $\frac{z^2}{c^2} = \frac{x^2}{a^2} + \frac{y^2}{b^2}$

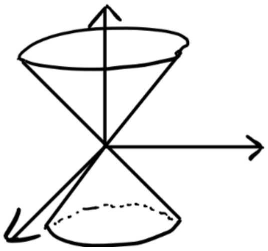


Figure 43: 3D geometric depiction of a cone, another member of the quadric family and a special case of the hyperboloid of one sheet.

6. **Elliptic Paraboloid**: $\frac{z}{c} = \frac{x^2}{a^2} + \frac{y^2}{b^2}$

   Note that this quadric surface has a linear, rather than quadratic dependence, on z.

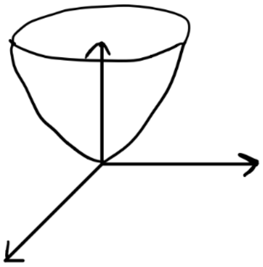


Figure 44: 3D geometric depiction of a elliptic paraboloid, another member of the quadric family and a special case of the hyperboloid of one sheet.

7. **Hyperbolic Paraboloid**: $\frac{z}{c} = \frac{x^2}{a^2} - \frac{y^2}{b^2}$

   Note that this quadric surface also has a linear, rather than quadratic dependence, on z.

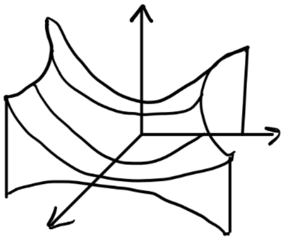


Figure 45: 3D geometric depiction of a hyperbolic paraboloid, another member of the quadric family and a special case of the hyperboloid of one sheet.

Another special case is derived through **planes**. You may be wondering - how can we have a quadratic structure from planes? We can derive a surface with quadratic terms by considering the intersection of two planes. This intersection of planes is computed analytically as a product of two linear equations, resulting in a quadratic equation.

19.3 Relative Orientation and Binocular Stereo

Problem of interest: Computing 3D from 3D using two cameras - a problem known as **binocular stereo**. We found this problem was easy to solve if the geometry of the two cameras, in this case known as the **two-view geometry**, is known and calibrated (usually this results in finding a calibration matrix K). To achieve high-performing binocular stereo systems, we need to find the relative orientation between the two cameras. A few other notes on this:

- Calibration is typically for **binocular stereo** using **baseline calibration**.
- Recall that like absolute orientation, we have **duality** in te problems we solve: we are able to apply the same machine vision framework regardless of "if the camera moved of it the world moved".
- Therefore, the dual problem to **binocular stereo** is **structure from motion**, also known as **Visual Odometry (VO)**. This involves finding transformations over time from camera (i.e. one moving camera at different points in time).

19.3.1 Binocular Stereo

Recall that our goal with binocular stereo is to find the **translation** (known as the **baseline**, given by $\mathbf{b} \in \mathbb{R}^3$. The diagram below illustrates our setup with two cameras, points in the world, and the translation between the two cameras given as this aforementioned baseline $\mathbf{b}$.

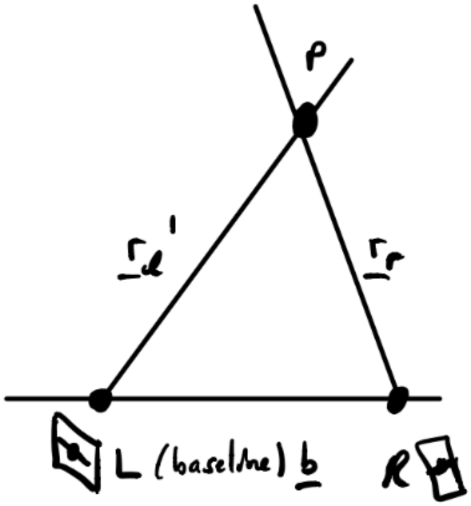


Figure 46: Binocular stereo system set up. For this problem, recall that one of our objectives is to measure the translation, or baseline, between the two cameras.

When we search for correspondences, we need only search along a line, which gives us a measure of **binocular disparity** that we can then use to measure distance between the cameras.

The lines defined by these correspondences are called **epipolar lines**. Places that pass through **epipolar lines** are called **epipolar planes**, as given below:

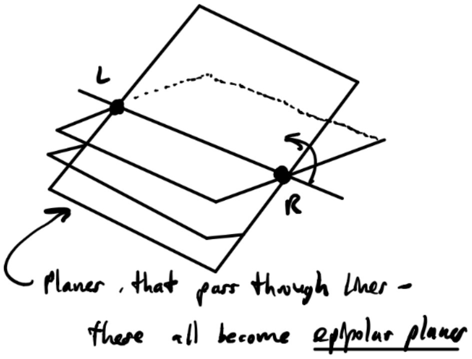

L
R
Planes, that pass through lines -
these all become epipolar planes

Figure 47: Epipolar planes are planes that pass through epipolar lines.

Next, in the image, we intersect the image plane with our set of epipolar planes, and we look at the intersections of these image planes (which become lines). The figure below illustrates this process for the left and right cameras in our binocular stereo system.

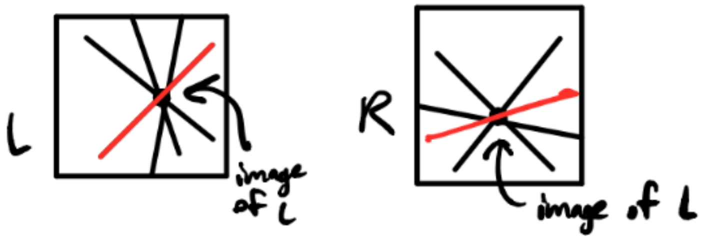


Figure 48: After finding the epipolar planes, we intersect these planes with the image plane, which gives us a set of lines in both the left and righthand cameras/coordinate systems of our binocular stereo system.

A few notes on this process:

- For system convergence, we want as much overlap as possible between the two sets of lines above, depicted in Figure 10.
- The entire baseline **b** projects to a single point in each image.
- As we stated before, there are line correspondences if we already know the geometry, i.e. if we want correspondences between the lines in red above, we need only look along the red lines in each image.
- Recall that our goal is to find the **transformation** between two cameras - i.e. **baseline** (or **translation**, in dual structure from motion problem).
- Recall, as we saw for absolute orientation, that we also solve for **rotation** in addition to the **baseline/translation**. Together, these translation + rotation variables lead us to think that we are solving a problem of 6 degrees of freedom (DOF), but because of **scale factor ambiguity** (we cannot get absolute sizes of objects/scenes we image), we cannot get absolute length of the baseline **b**, and therefore, we treat the baseline as a unit vector. Treating the baseline as a unit vector results in one less DOF and 5 DOF for the system overall (since we now only have 2 DOF for the unit vector baseline).

19.3.2 How Many Correspondences Do We Need?

As we have done so for other systems/problems, we consider the pertinent question: **How many correspondences do we need to solve the binocular stereo/relative orientation problem?**. To answer this, let us consider what happens when we are given different numbers of correspondences:

- With 1 correspondence, we have many degrees of freedom left - i.e. not enough constraints for the number of unknowns remaining. This makes sense considering that each correspondence imposes a maximum of 3 DOF for the system.

- With 2 correspondences, we still have the ability to rotate one of the cameras without changing the correspondence, which implies that only 4 out of the 5 constraints are satisfied. This suggests we need more constraints.

It turns out we need **5 correspondences** needed to solve the binocular stereo/relative orientation problem - each correspondence gives you **1 constraint**. Why? Each image has a **disparity**, with two components (this occurs in practice). There are two types of disparity, each corresponding to each of the 2D dimensions. Note that **disparity** computes pixel discrepancies between correspondences between images.

- **Horizontal disparity** corresponds to depth.
- **Vertical disparity** corresponds to differences in orientation, and actually takes the constraints into account. In practice, vertical disparity is tuned out first using an iterative sequence of 5 moves.

In practice, we would like to use **more correspondences for accuracy**. With more correspondences, we no longer try to find exact matches, but instead minimize a sum of squared errors using least squares methods. We minimize sum of squares of error in **image position**, not in the 3D world.

However, one issue with this method is **nonlinearity**:

- This problem setup results in 7 second-order equations, which, by Bezout's Theorem, gives $2^7 = 128$ different solutions to this problem.
- Are there really 128 solutions? There are methods that have gotten this down to 20. With more correspondences and more background knowledge of your system, it becomes easier to determine what the true solution is.

19.3.3 Determining Baseline and Rotation From Correspondences

To determine baseline and rotation for this binocular stereo problem, we can begin with the figure below:

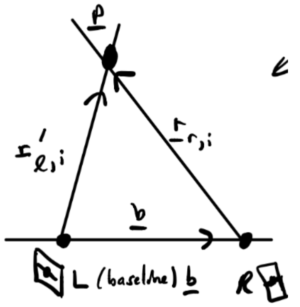


Figure 49: Epipolar plane for the binocular stereo system.

The vectors $\mathbf{r}'_{l,i}$ (the left system measurement after the rotation transformation has been applied), $\mathbf{r}_{r,i}$, and $\mathbf{b}$ are all coplanar in a perfect system. Therefore, if we consider the triple product of these 3 vectors, the volume of the parallelipiped, which can be constructed from the **triple product**, should have **zero volume** because these vectors are coplanar (note that this is the ideal case). This is known as the **coplanarity condtion**:

$$V = [{}'_{\mathrm{l,i}}\ \mathbf{r}_{\mathrm{r,i}}\ \mathbf{b}] = 0$$

A potential solution to find the optimal baseline (and rotation): we can use least squares to minimize the volume of the parallelipiped corresponding to the triple product of these three (hopefully near coplanar) vectors. This is a feasible approach, but also have a **high noise gain/variance**.

This leads us to an important question: What, specifically, are we trying to minimize? Recall that we are **matching correspondences in the image, not in the 3D world.** Therefore, the volume of the parallelipiped is **proportional to the error**, but does not match it exactly. When we have measurement noise, the rays from our vectors in the left and righthand systems no longer intersect, as depicted in the diagram below:

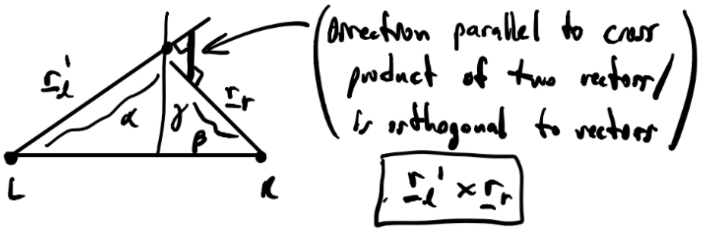

(Direction parallel to cross product of two vectors/ is orthogonal to vectors)
r_l' × r_r
r_l'
r_r
α
γ
β
L
R

Figure 50: With measurement noise, our rays do not line up exactly. We will use this idea to formulate our optimization problem to solve for optimal baseline and rotation.

We can write that the error is perpendicular to the cross product between the rotated left and the right rays:

$$\mathbf{r}'_{l,i} \times \mathbf{r}_r$$

And therefore, we can write the equation for the "loop" (going from the rotated left coordinate system to the right coordinate system) as:

$$\alpha \mathbf{r}'_l + \gamma(\mathbf{r}'_l \times \mathbf{r}_r) = \mathbf{b} + \beta \mathbf{r}_r$$

Where the error we seek to minimize $(\mathbf{r}'_l \times \mathbf{r}_r)$ is multiplied by a parameter γ.

To solve for our parameters α, β, and γ, we can transform this vector equation into 3 scalar equations by taking dot products. We want to take dot products that many many terms drop to zero, i.e. where orthogonality applies. Let's look at these 3 dot products:

1. $\cdot(\mathbf{r}'_l \times \mathbf{r}_r)$: This yields the following:

$$\begin{aligned}
&\textbf{Lefthand side}: \ (\alpha \mathbf{r}'_l + \gamma(\mathbf{r}'_l \times \mathbf{r}_r)) \cdot (\mathbf{r}'_l \times \mathbf{r}_r) = \gamma ||\mathbf{r}'_l \times \mathbf{r}_r||_2^2 \\
&\textbf{Righthand side}: \ (\mathbf{b} + \beta \mathbf{r}_r) \cdot (\mathbf{r}'_l \times \mathbf{r}_r) = \mathbf{b} \cdot \mathbf{r}'_l \times \mathbf{r}_r + 0 = [\mathbf{b} \ \mathbf{r}'_l \ \mathbf{r}_r] \\
&\textbf{Combining}: \ \gamma ||\mathbf{r}'_l \times \mathbf{r}_r||_2^2 = [\mathbf{b} \ \mathbf{r}'_l \ \mathbf{r}_r]
\end{aligned}$$

Intuitively, this says that the error we see is proportional to teh triple product (the volume of the parallelipiped). Taking our equation with this dot product allows us to isolate γ.

2. $\cdot(\mathbf{r}'_l \times (\mathbf{r}'_l \times \mathbf{r}_r))$: This yields the following:

$$\begin{aligned}
&\textbf{Lefthand side}: \ (\alpha \mathbf{r}'_l + \gamma(\mathbf{r}'_l \times \mathbf{r}_r)) \cdot ((\mathbf{r}'_l \times (\mathbf{r}'_l \times \mathbf{r}_r)) = \beta ||\mathbf{r}'_l \times \mathbf{r}_r||_2^2 \\
&\textbf{Righthand side}: \ (\mathbf{b} + \beta \mathbf{r}_r) \cdot ((\mathbf{r}'_l \times (\mathbf{r}'_l \times \mathbf{r}_r)) = (\mathbf{b} \times \mathbf{r}'_l) \cdot (\mathbf{r}'_l \times \mathbf{r}_r) \\
&\textbf{Combining}: \ \beta ||\mathbf{r}'_l \times \mathbf{r}_r||_2^2 = (\mathbf{b} \times \mathbf{r}'_l) \cdot (\mathbf{r}'_l \times \mathbf{r}_r)
\end{aligned}$$

Taking this dot product with our equation allows us to find β. We can repeat an analogous process to solve for α.

3. $\cdot(\mathbf{r}_r \times (\mathbf{r}'_l \times \mathbf{r}_r))$: This yields the following:

$$\begin{aligned}
&\textbf{Lefthand side}: \ (\alpha \mathbf{r}'_l + \gamma(\mathbf{r}'_l \times \mathbf{r}_r)) \cdot ((\mathbf{r}_r \times (\mathbf{r}'_l \times \mathbf{r}_r)) = \alpha ||\mathbf{r}'_l \times \mathbf{r}_r||_2^2 \\
&\textbf{Righthand side}: \ (\mathbf{b} + \beta \mathbf{r}_r) \cdot ((\mathbf{r}_r \times (\mathbf{r}'_l \times \mathbf{r}_r)) = (\mathbf{b} \times \mathbf{r}_r) \cdot (\mathbf{r}'_l \times \mathbf{r}_r) \\
&\textbf{Combining}: \ \alpha ||\mathbf{r}'_l \times \mathbf{r}_r||_2^2 = (\mathbf{b} \times \mathbf{r}_r) \cdot (\mathbf{r}'_l \times \mathbf{r}_r)
\end{aligned}$$

Taking this dot product with our equation allows us to find α.

With this, we have now isolated all three of our desired parameters. We can then take these 3 equations to solve for our 3 unknowns α, β, and γ. We want $|\gamma|$ to be as small as possible. We also require α and β to be non-negative, since these indicate the scalar multiple of the direction along the rays in which we (almost) get an intersection between the left and right coordinate systems. Typically, a negative α and/or β results in intersections behind the camera, which is often not physically feasible.

It turns out that one of the ways discard some of the 20 solutions produced by this problem is to throw out solutions that result in negative α and/or β.

Next, we can consider the distance this vector corresponding to the offset represents. This distance is given by:

$$d = \gamma ||\mathbf{r}'_l \times \mathbf{r}_r||_2 = \frac{[\mathbf{b} \ \mathbf{r}'_l \ \mathbf{r}_r]}{||\mathbf{r}'_l \times \mathbf{r}_r||_2}$$

Closed-Form Solution: Because this is a system involving 5 second-order equations, and the best we can do is reduce this to a single 5th-order equation, which we do not (yet) have the closed-form solutions to, we cannot solve for this problem in closed-form. However, we can still solve for it numerically. We can also look at solving this through a weighted least-squares approach below.

19.3.4 Solving Using Weighted Least Squares

Because our distance d can get very large without a huge error in image position, d is not representative of our error of interest. However, it is still proportional. We can account for this difference by using a weighting factor w_i, which represents the conversion factor from 3D error to 2D image error). We will denote the i$^{\text{th}}$ weight as w_i.

Therefore, incorporating this weighting factor, our least squares optimization problem becomes:

$$\min_{\mathbf{b},R(\cdot)} \sum_{i=1}^{n} w_i[\mathbf{b}\ \mathbf{r}_l'\ \mathbf{r}_r]^2, \text{ subject to } \mathbf{b}\cdot\mathbf{b} = ||\mathbf{b}||_2^2 = 1$$

How do we solve this? Because w_i will change as our candidate solution changes, we will solve this problem iteratively and in an alternating fashion - we will alternate between updated our conversion weights w_i and solving for a new objective given the recent update of weights. Therefore, we can write this optimization problem as:

$$\begin{aligned}
\mathbf{b}^*, R^* &= \arg\min_{\mathbf{b},||\mathbf{b}||_2^2=1,R(\cdot)} \sum_{i=1}^{n} w_i[\mathbf{b}\ \mathbf{r}_l'\ \mathbf{r}_r]^2 \\
&= \arg\min_{\mathbf{b},||\mathbf{b}||_2^2=1,R(\cdot)} \sum_{i=1}^{n} w_i\left((\mathbf{r}_{r,i}\times\mathbf{b})\cdot\mathbf{r}_{l,i}'\right)^2
\end{aligned}$$

Intuition for these weights: Loosely, we can think of the weights as being the conversion factor from 3D to 2D error, and therefore, they can roughly be thought of as $w = \frac{f}{Z}$, where f is the focal length and Z is the 3D depth.

Now that we have expressed a closed-form solution to this problem, we are ready to build off of the last two lectures and apply our unit quaternions to solve this problem. Note that because we express the points from the left coordinate system in a rotated frame of reference (i.e. with the rotation already applied), then we can incorporate the quaternion into this definition to show how we can solve for our optimal set of parameters given measurements from both systems:

$$\overset{o'}{r}_l = \overset{o}{q}\overset{o}{r}_l\overset{o^*}{q}$$

Then we can solve for this as t:

$$\begin{aligned}
t &= (\mathbf{r}_r\times\mathbf{b})\cdot\mathbf{r}_l' \\
&= \overset{o}{r}_r\overset{o}{b}\cdot\overset{o}{q}\overset{o}{r}_l\overset{o^*}{q} \\
&= \overset{o}{r}_r(\overset{o}{b}\overset{o}{q})\cdot\overset{o}{q}\overset{o}{r}_l \\
&= \overset{o}{r}_d\overset{o}{d}\cdot\overset{o}{q}\overset{o}{r}_l, \ \text{ where } \overset{o}{d} \triangleq \overset{o}{b}\overset{o}{q}, \text{ which we can think of as a product of baseline and rotation.}
\end{aligned}$$

Recall that our goal here is to find the **baseline** $\overset{o}{b}$ - this can be found by solving for our quantity $\overset{o}{d}$ and multiplying it on the righthand side by $\overset{o^*}{q}$:

$$\begin{aligned}
&\overset{o}{d}\overset{o^*}{q} = \overset{o}{b}\overset{o}{q}\overset{o^*}{q} = \overset{o}{b}\overset{o}{e} = \overset{o}{b} \\
&(\text{Recall that } \overset{o}{e} = (1,0) = \overset{o}{q}\overset{o^*}{q})
\end{aligned}$$

At first glance, it appears we have 8 unknowns, with 5 constraints. But we can add additional constraints to the system to make the number of constraints equal the number of DOF:

1. **Unit quaternions**: $||\overset{o}{q}||_2 = \overset{o}{q}\cdot\overset{o}{q} = 1$

2. **Unit baseline**: $||\overset{o}{b}||_2 = \overset{o}{b}\cdot\overset{o}{b} = 1$

3. **Orthogonality of $\overset{o}{q}$ and $\overset{o}{d}$**: $\overset{o}{q}\cdot\overset{o}{d} = 0$

Therefore, with these constraints, we are able to reach a system with 8 constraints. Note that we have more constraints to enforce than with the absolute orientation problem, making the relative orientation problem more difficult to solve.

19.3.5 Symmetries of Relative Orientation Approaches

We can interchange the left and right coordinate system rays. We can do this for this problem because we have **line intersections** rather than **line rays**. These symmetries can be useful for our numerical approaches. The equation below further demonstrates this symmetry by showing we can interchange the order of how $\overset{o}{d}$ and $\overset{o}{q}$ interact with our measurements to produce the same result.

$$\begin{aligned} t &= \overset{o}{r}_r \overset{o}{d} \cdot \overset{o}{q} \overset{o}{r}_l \\ &= \overset{o}{r}_r \overset{o}{q} \cdot \overset{o}{d} \overset{o}{r}_l \end{aligned}$$

Given these symmetries, we can come up with some sample solutions:

1. $\{\overset{o}{q}, \overset{o}{d}\}$
2. $\{-\overset{o}{q}, \overset{o}{d}\}$ $(\times 2)$
3. $\{\overset{o}{q}, -\overset{o}{d}\}$ $(\times 2)$
4. $\{\overset{o}{d}, \overset{o}{q}\}$ $(\times 2)$

How do we avoid having to choose from all these solutions? One approach: Assume/fix one of the two unknowns $\overset{o}{d}$ or $\overset{o}{q}$. This results in a linear objective and a linear problem to solve, which, as we know, can be solved with **least squares** approaches, giving us a closed-form solution:

- **Option 1**: Assume $\overset{o}{q}$ is known and fix it $\longrightarrow$ solve for $\overset{o}{d}$
- **Option 2**: Assume $\overset{o}{d}$ is known and fix it $\longrightarrow$ solve for $\overset{o}{q}$

Note that both of the approaches above can be solved with least squares!

While this is one approach, a better approach is to use **nonlinear optimization**, such as **Levenberg-Marquadt** (often called LM in nonlinear optimization packages). An optional brief intro to Levenberg-Marquadt is presented at the end of this lecture summary.

Levenberg-Marquadt (LM) optimization requires a non-redundant parameterization, which can be achieved with either **Gibbs vectors** or **quaternions** (the latter of which we can circumvent the redundancies of Hamilton's quaternions by treating the redundancies of this representation as extra constraints. Recall that Gibbs vectors, which are given as $(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\hat{\boldsymbol{\omega}})$, have a singularity at $\theta = \pi$.

19.3.6 When Does This Fail? Critical Surfaces

Like all the other machine vision approaches we have studied so far, we would like to understand when and why this system fails:

- Recall that we need at least 5 correspondences to solve for the **baseline** and **rotation**. With less correspondences, we don't have enough constraints to find solutions.
- **Gefärliche Flaächen**, also known as **"dangerous or critical surfaces"**: There exist surfaces that make the relative orientation problem difficult to solve due to the additional ambiguity that these surfaces exhibit. One example: **Plane flying over a valley with landmarks:**

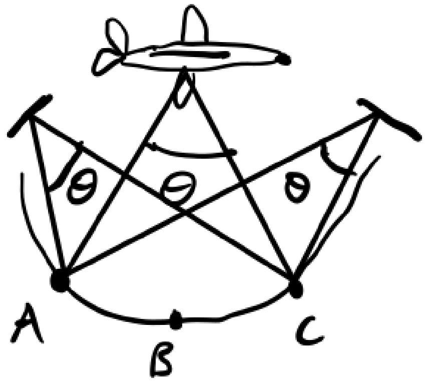


Figure 51: A plane flying over a valley is an instance of a critical surface. This is because we can only observe angles, and not points. Therefore, locations of two landmarks are indistinguishable.

To account for this, pilots typically plan flight paths over ridges rather than valleys for this exact reason:

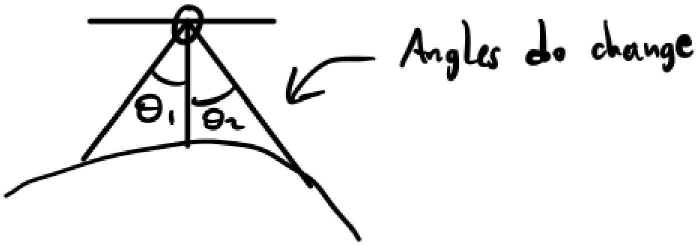


Figure 52: When planes fly over a ridge rather than a valley, the angles between two landmarks do change, allowing for less ambiguity for solving relative orientation problems.

How do we generalize this case to 3D? We will see that this type of surface in 3D is a **hyperboloid of one sheet**. We need to ensure sections of surfaces you are looking at do not closely resemble sections of hyperboloids of one sheet.

There are also other types of **critical surfaces** that are far more common that we need to be mindful of when considering relative orientation problems - for instance, the intersection of two planes. In 2D, this intersection of two planes can be formed by the product of their two equations:

$$(ax + by + c)(dx + ey + f) = adx^2 + aexy + afx + bdxy + bey^2 + bfy + cdx + cey + cf = 0$$

We can see that this equation is indeed second order with respect to its spatial coordinates, and therefore belongs to the family of quadric surfaces and critical surfaces.

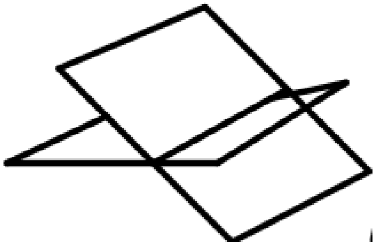


Figure 53: The intersection of two planes is another type of critical surface we need to be mindful of. It takes a second-order form because we multiply the two equations of the planes together to obtain the intersection.

19.3.7 (Optional) Levenberg-Marquadt and Nonlinear Optimization

Levenberg-Marquadt (LM) and Gauss-Newton (GN) are two nonlinear optimization procedures used for deriving solutions to nonlinear least squares problems. These two approaches are largely the same, except that LM uses an additional **regularization**

term to ensure that a solution exists by making the closed-form matrix to invert in the normal equations positive semidefinite. The normal equations, which derive the closed-form solutions for GN and LM, are given by:

1. **GN**: $(J(\theta)^T J(\theta))^{-1}\theta = J(\theta)^T e(\theta) \implies \theta = (J(\theta)^T J(\theta))^{-1} J(\theta)^T e(\theta)$
2. **LM**: $(J(\theta)^T J(\theta) + \lambda I)^{-1}\theta = J(\theta)^T e(\theta) \implies \theta = (J(\theta)^T J(\theta) + \lambda I)^{-1} J(\theta)^T e(\theta)$

Where:

- θ is the vector of parameters and our solution point to this nonlinear optimization problem.
- $J(\theta)$ is the Jacobian of the nonlinear objective we seek to optimize.
- $e(\theta)$ is the residual function of the objective evaluated with the current set of parameters.

Note the λI, or regularization term, in Levenberg-Marquadt. If you're familiar with ridge regression, LM is effectively ridge regression/regression with L2 regularization for nonlinear optimization problems. Often, these approaches are solved iteratively using gradient descent:

1. **GN**: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha(J(\theta^{(\mathbf{t})})^T J(\theta^{(\mathbf{t})}))^{-1} J(\theta^{(\mathbf{t})})^T e(\theta^{(\mathbf{t})})$
2. **LM**: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha(J(\theta^{(\mathbf{t})})^T J(\theta^{(\mathbf{t})}) + \lambda I)^{-1} J(\theta^{(\mathbf{t})})^T e(\theta^{(\mathbf{t})})$

Where α is the step size, which dictates how quickly the estimates of our approaches update.

19.4 References

1. Tesselation, https://en.wikipedia.org/wiki/Tessellation

20 Lecture 21: Relative Orientation, Binocular Stereo, Structure from Motion, Quadrics, Camera Calibration, Reprojection

Today, we will continue discussing **relative orientation**, one of our main problems in photogrammetry. We concluded our previous lecture with a discussion of critical surfaces - that is, surfaces which make solving the relative rotation problem difficult. In general, these critical surfaces are **quadric surfaces**, which take the form:

$$\pm\left(\frac{x}{a}\right)^2 \pm \left(\frac{y}{b}\right)^2 \pm \left(\frac{z}{c}\right)^2 = 1$$

This form is simple: (i) Centered at the origin, (ii) Axes line up. In general, the form of these quadric surfaces may be more complicated, with both linear and constant terms. The signs of these qudratic terms determine the shape, as we saw in the previous lecture:

1. $+++$ (ellipsoid)
2. $++-$ (hyperboloid of one sheet) - this is a class of critical surfaces
3. $+--$ (hyperboloid of two sheets)
4. $---$ (imaginary ellipsoid)

For the solutions to the polynomial $R^2 + \cdots - R = 0$ (the class of **hyperboloids of one sheets**), the solutions tell us why these critical surfaces create solution ambiguity:

1. $\mathbf{R} = \mathbf{0}$: I.e. the solution is on the surface - the origin of the righthand system is on the surface. Two interpretations from this; the first is for binocular stereo, and the second is for Structure from Motion (SfM):
   (a) **Binocular Stereo**: The righthand system lies on the surface.
   (b) **Structure from Motion**: In Structure from Motion, we "move ont the surface" in the next time step.
2. $\mathbf{R} = -\mathbf{b}$: This solution is the origin of the lefthand system (we move the solution left along the baseline from the righthand system to the lefthand system). Two interpretations from this; the first is for binocular stereo, and the second is for Structure from Motion (SfM):
   (a) **Binocular Stereo**: "The surface has to go through both eyes".

(b) **Structure from Motion**: We must start and end back on the surface.

3. $\mathbf{R} = k\mathbf{b}$: Because this system (described by the polynomial in $\mathbf{R}$, then we have no scaling, giving us a solution. This suggests that the entire baseline in fact lies on the surface, which in turn suggests:
   - This setting/surface configuration is rare.
   - This surface is **ruled** - we can draw lines inscribed in the surface. This suggests that our critical surface is a hyperboloid of one sheet. It turns out that we have two rulings for the hyperboloid at one sheet (i.e. at every point on this surface, we can draw two non-parallel lines through the surface that cross at a point).

Hyperboloids of one sheet, as we saw in the previous lecture, are not the only types of critical surfaces. **Intersections of planes** are another key type of critical surface, namely because they are much more common in the world than hyperboloids of one sheet. Analytically, the intersection of planes is given by the product of planes:

$$(a_1X + b_1Y + c_1Z + d_1)(a_2X + b_2Y + c_2Z + d_2) = 0$$

This analytic equation describes the intersection of two planes - gives us a quadric surface.

In order for this intersection of planes to not have any constants, one of the planes must be the **epipolar plane**, whose image is a line. The second plane is arbitrary and can take any form.

Practically, we may not run into this problem in this exact analytic form (i.e. a surface that is an exact instance of a hyperboloid of one sheet), but even as we venture near this condition, the noise amplification factor increases. As it turns out, using a higher/wider Field of View (FOV) helps to mitigate this problem, since as FOV increases, it becomes less and less likely that the surface we are imaging is locally similar to a critical surface.

One way that we can increase the FOV is through the use of **spider heads**, which consist of 8 cameras tied together into a rigid structure. Though it requires calibration between the cameras, we can "stitch" together the images from the 8 cameras into a single "mosaic image".

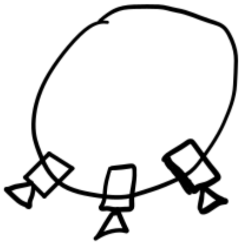


Figure 54: A spider heads apparatus, which is one technique for increasing the FOV, thereby reducing the probability of surfaces resembling critical surfaces.

Next, we will switch our discussion to another of our photogrammetry problems - interior orientation.

20.1 Interior Orientation

Recall that interior orientation was another one of the 4 photogrammetry problems we study, and involves going from 3D (world coordinate) to 2D (image coordinates). Specifically, we focus on **camera calibration**.

Have we already touched on this? Somewhat - with **vanishing points**. Recall that the goal with vanishing points was to find (x_0, y_0, f) using calibration objects. This method:

- Is not very accurate, nor general across different domains/applications
- Does not take into account radial distortion, which we need to take into account for high-quality imaging such as aerial photography.

What is radial distortion? We touch more on this in the following section.

20.1.1 Radial Distortion

This type of distortion leads to a discrepancy between what should be the projected location of an object in the image, and what it actually projects to. This distortion is radially-dependent.

This type of distortion becomes more apparent when we image lines/edges. It manifests from having a **center of distortion**. The image of an object does not appear exactly where it should, and the error/discrepancy depends on the radius of the point in the image relative to the center of distortion. This radial dependence is typically approximated as a polynomial in the radius r:

$$\begin{aligned}
r &= ||\mathbf{r}||_2 \\
\delta_x &= x(k_1 r^2 + k_2 r^4 + \cdots) \\
\delta_y &= y(k_1 r^2 + k_2 r^4 + \cdots)
\end{aligned}$$

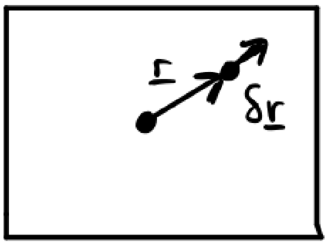
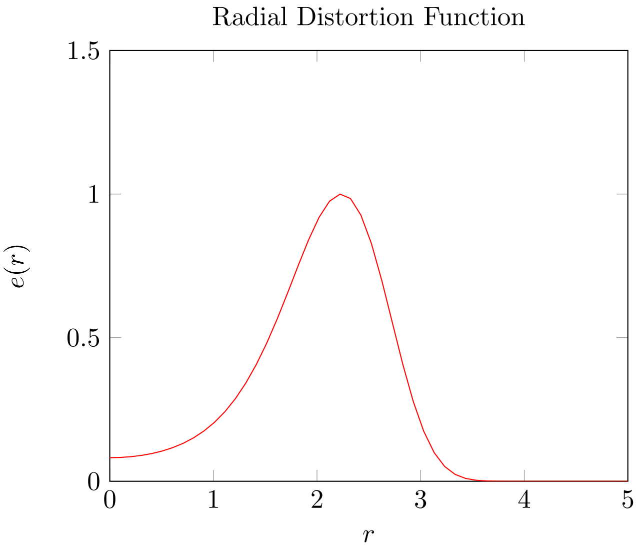


Figure 55: Radial distortion is in the direction of the radius emanating from the center of projection to the point in the image plane where a point projects to in the absence of radial distortion.

Note that the error vector $\delta\mathbf{r}$ is parallel to the vector $\mathbf{r}$ - i.e. the distortion is in the direction of the radial vector.

Many high-quality lenses have a defined radial distortion function that gives the distortion as a function of r, e.g. the figure below:



These functions, as we stated above, are typically approximated as polynomial functions of r. **How do we measure radial distortion?** One way to do so is through the use of **plumb lines**. These lines are suspended from the ceiling via weights, which allows us to assume that they are straight and all parallel to one another in the 3D world. Then, we take an image of these lines, with the image centered on the center lines. We can estimate/assess the degree of radial distortion by identifying the curvature, if any, of the lines as we move further away from the center lines of the image.

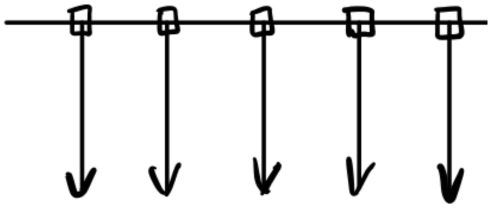


Figure 56: Plumb lines are a useful way of estimating radial distortion, because they allow us to estimate the degree of curvature between lines that should be straight and parallel.

Plumb lines allow us to estimate two types of radial distortion: (i) **Barrel Distortion** and (ii) **Pincushion Distortion**, depicted below. The radius of curvature from these lines allows us to measure k_1 (the first polynomial coefficient for r^2):

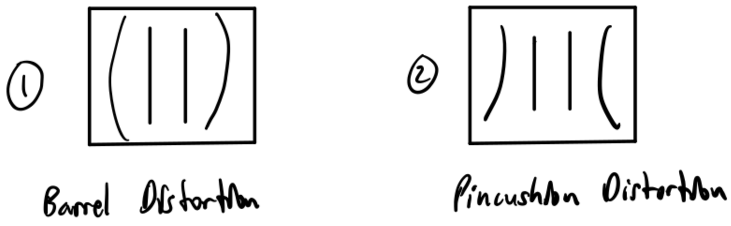


Figure 57: Barrel and pincushion distortion, which we can absorb from plumb lines.

A more subtle question: is it better to go from)a) **undistorted** → **distorted** coordinates, or from (b) **distorted** → **undistorted** coordinates?

It is often desirable to take approach (b) (**distorted** → **undistorted**) because we can measure the distorted coordinates. Wse can use **series inversion** to relate the distorted and undistorted coodinates. This affects the final coordinate system that we do optimization in, i.e. we optimize error in the image plane.

20.1.2 Tangential Distortion and Other Distortion Factors

There are other types of distortion that we can analyze and take into account:

1. **Tangential Distortion**: This is a problem we have solved, but to contrast this with radial distortion, rather than the distortion acting radially, the distortion acts tangentially (see figure below). With tangential distortion, our offsets take the form:

$$\delta_x = -y(\epsilon_1 r^2 + \epsilon_2 r^4 + \cdots)$$
$$\delta_y = x(\epsilon_1 r^2 + \epsilon_2 r^4 + \cdots)$$

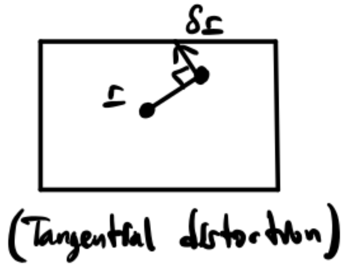


Figure 58: Tangential distortion acts tangentially relative to the radius - in this case, in a counterclockwise fashion (as we rotate θ).

2. **Decentering**: If the center of distortion is not the principal point (center of the image with perspective projection), then we get an offset that depends on position. This offset is typically small, but we still need to take it into account for high-quality imaging such as aerial photography.

3. **Tilt of Image Plane**: If the image plane is tilted (see the figure below), then magnification will not be uniform and the focus will not be perfect. In practice, to fix this, rather than changing the tilt of the image plane, you can instead insert a **compensating element** to offset this tilt.

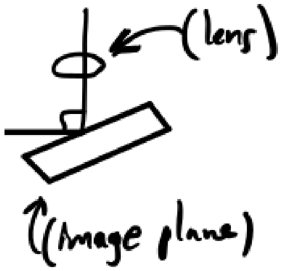


Figure 59: Tilt of the image plane is another factor that can affect distortion of images, as well as variable magnification/scaling across the image. Oftentimes, it can be corrected via a compensating element that offsets this tilt, rather than realigning the image plane.

Taking these effects into account allows for creating more sophisticated distortion models, but sometimes at the expense of overfitting and loss of generalizability (when we have too many degrees of freedom, and our estimates for them become so precariously configured that our model is not applicable for a slight change of domain).

These distortion models can also be used for nonlinear optimization protocols such as Levenberg-Marquadt (LM) or Gauss-Newton (GN) (see lecture 20 for a review of these).

20.2 Tsai's Calibration Method

This calibration method relies on using a calibration object for which we have precise knowledge of the points on our 3D points of the object.

In camera calibration, **correspondences** are defined between **points in the image** and **points on the calibration object**.

Here, we encounter the same reason that vanishing point-based methods are difficult - it is hard to directly relate the camera to calibration objects. When we take **exterior orientation** into account, we generally perform much better and get higher-performing results. Exterior orientation (2D → 3D) seeks to find the calibration object in space given a camera image.

Adding **exterior orientation** also increases the complexity of this problem:

- **Interior orientation** has 3 Degrees of Freedom
- **Exterior orientation** has 6 Degrees of Freedom (translation and rotation)

Combined, therefore our problem now possesses 9 DOF. We can start solving this problem with perspective projection and interior orientation.

20.2.1 Interior Orientation

Beginning with perspective projection, we have:

$$\frac{x_I - x_O}{f} = \frac{X_c}{Z_c}$$
$$\frac{x_I - x_O}{f} = \frac{X_c}{Z_c}$$

Where:

1. $(X_c, Y_c, Z_c)^T$ are the camera coordinates (world coordinate units)
2. (x_I, y_I) denote image position (row, column/grey level units)
3. (x_O, y_O, f) denotie interior orientation/principal point (column/grey level and pixel units)

Can we modify the measurements so we are able to take radial distortion into account?

We need a good initial guess, because, numerical, iterative approaches that are used to solve this problem precisely exhibit multiple mimima, and having a good initial guess/estimate to serve as our "prior" will help us to find the correct minima.

From here, we can add exterior orientation.

20.2.2 Exterior Orientation

As we are solving for translation and rotation, our equation for exterior orientation equation becomes (written as a matrix-vector problem):

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$
$$\mathbf{X}_c = \mathbf{R}\mathbf{X}_s + \mathbf{t}$$

The figure below illustrates the desired transformation between these two reference frames:

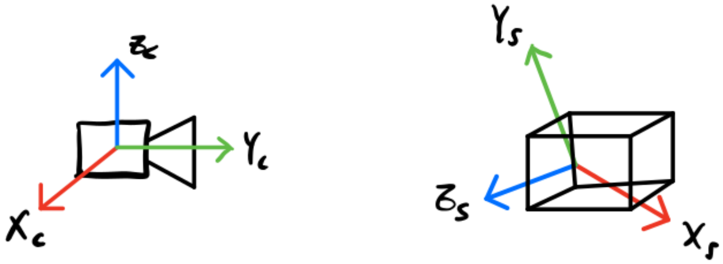


Figure 60: Exterior orientation seeks to find the transformation between a camera and a calibration object.

Where the matrix $\mathbf{R}$ and the translation $\mathbf{t}$ are our unknowns. From this, we can combine these equations with our equations for interior orientation to find our solutions.

20.2.3 Combining Interior and Exterior Orientation

Combining our equations from interior and exterior orientation:

- **Interior Orientation**:

$$\frac{x_I - x_O}{f} = \frac{X_c}{Z_c}$$
$$\frac{x_I - x_O}{f} = \frac{X_c}{Z_c}$$

- **Exterior Orientation**:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$
$$\mathbf{X}_c = \mathbf{R}\mathbf{X}_s + \mathbf{t}$$

Then we can combine these equations into:

$$\frac{x_I - x_O}{f} = \frac{X_c}{Z_c} = \frac{r_{11}X_s + r_{12}Y_s + r_{13}Z_s + t_x}{r_{31}X_s + r_{32}Y_s + r_{33}Z_s + t_z}$$

$$\frac{y_I - y_O}{f} = \frac{Y_c}{Z_c} = \frac{r_{21}X_s + r_{22}Y_s + r_{23}Z_s + t_y}{r_{31}X_s + r_{32}Y_s + r_{33}Z_s + t_z}$$

Written this way, we can map directly from **image coordinates** to **calibration objects**.

How can we find f and t_z, as well as mitigate radial distortion?

Let's start by looking at this problem in polar coordinates. Radial distortion only changes radial lengths (r), and not angle (θ). Changing f or Z_c changes only radius/magnification. Let's use this factor to "forget" about the angle.

We can also remove additional DOF by **dividing our combined equations** by one another:

$$\frac{x_I - x_O}{y_I - y_0} = \frac{\frac{r_{11}X_s + r_{12}Y_s + r_{13}Z_s + t_x}{r_{31}X_s + r_{32}Y_s + r_{33}Z_s + t_z}}{\frac{r_{21}X_s + r_{22}Y_s + r_{23}Z_s + t_y}{r_{31}X_s + r_{32}Y_s + r_{33}Z_s + t_z}}$$

$$= \frac{r_{11}X_s + r_{12}Y_s + r_{13}Z_s + t_x}{r_{21}X_s + r_{22}Y_s + r_{23}Z_s + t_y}$$

We can see that combining these constraints removes our DOF t_z, giving us 8 constraints rather than 9. Cross-multiplying and term consolidation gives:

$$(X_S y_I')r_{11} + (Y_S y_I')r_{12} + (Z_S y_I')r_{13} + y_I' t_x - (X_S x_I')r_{21} - (Y_S x_I')r_{22} - (Z_S x_I')r_{23} - x_I' t_y = 0$$

A few notes on this consolidated homogeneous equation:

- Since our unknowns are the r_{ij} values, we have a linear combination/equation of our unknowns! The coefficients they are multiplied by are either our observed image positions or the positions of the correspondences/keypoints on the calibration object.
- Note that we subtract out an estimate of the center of projection. The estimation error of this center of projection affects adjacent points the most, as a slight miscalculation can greatly affect the calculations above - and as a result we throw out all correspondences that are ϵ-close (for some $\epsilon > 0$ to the center of the image.
- Each correspondence between image coordinates and the points on the calibration object gives one of these equations: "This point in the image corresponds to another point on the calibration object".

As we have asked for the other frameworks and systems we have analyzed: **How many correspondences do we need?** Let's consider which unknowns appear in the equations, and which do not:

- DOF/unknowns in the equations: $r_{11}, r_{12}, r_{13}, r_{21}, r_{22}, r_{23}, t_x, t_y$ (8 unknowns)
- DOF/unknowns not in the equations: $r_{31}, r_{32}, r_{33}, t_z, f$ (5 unknowns)

Can this inform of how we can solve this problem? Let's consider the following points:

- We are not yet enforcing orthonormality of the rotation matrices, i.e. $\mathbf{R}^T\mathbf{R} = \mathbf{I}$.
- We can solve for the last row of the rotation matrix containing the unknowns r_{31}, r_{32}, r_{33} through finding the cross product of the first two rows of this matrix.
- Having 8 unknowns suggests that we need at least 8 correspondences, but because this equation is homogeneous, there is **scale factor ambiguity** (i.e. doubling all the variables does not change the righthand side value of the equation). The solution to account for this scale factor ambiguity is to set/fix one of these variables to a constant value. Fixing an unknown means we now have 7 DOF (rather than 8), and therefore, **we actually need 7 correspondences**. Solving with 7 (or more) correspondences gives us a multiple of the solution. Solving with exactly 7 correspondences gives us 7 equations and 7 unknowns, which we can solve with, for instance, Gauss-Jordan elimination. More correspondences are desirable because they allow us to estimate both solutions and the error associated with these solutions.
- These solutions give us values for the 8 unknowns in the equation above: $r_{11}', r_{12}', r_{13}', r_{21}', r_{22}', r_{23}', t_x, t_y = 1$ (where t_y is the fixed parameter to account for scale ambiguity). However, as aforementioned these values are scaled versions of the true solution, and we can compute this scale factor by noting that because the rows of our (soon to be) orthonormal rotation matrix must have unit length - therefore the scale factor is given by computing the length of the rows of this matrix:

$$s = \frac{1}{\sqrt{{r_{11}'}^2 + {r_{12}'}^2 + {r_{13}'}^2}}$$

OR

$$s = \frac{1}{\sqrt{{r_{21}'}^2 + {r_{22}'}^2 + {r_{23}'}^2}}$$

> A good sanity check: these two variables should evaluate to approximately the same value - if they do not, this is often one indicator of correspondence mismatch.

With this scale factor computed we can find the true values of our solution by multiplying the scaled solutions with this scale factor:

$$s(r'_{11}, r'_{12}, r'_{13}, r'_{21}, r'_{22}, r'_{23}, t_x, t_y = 1) \rightarrow r_{11}, r_{12}, r_{13}, r_{21}, r_{22}, r_{23}, t_x, t_y$$

Note that we also have not (yet) enforced orthonormality of the first two rows in the rotation matrix, i.e. our goal is to have (or minimize):

$$r'_{11}r'_{21} + r'_{12}r'_{22} + r'_{13}r'_{23} = 0$$

To accomplish this, we will use the idea of "squaring up".

20.2.4 "Squaring Up"

Squaring up is one method for ensuring that vectors are numerically/approximately orthogonal to one another. This technique answers the question of "Given a set of orthogonal vectors, what is the nearest set of orthogonal vectors?"

The figure below highlights that given vectors **a** and **b**, we can find the nearest set of rotated vectors **a**′ and **b**′ using the following, which states that adjustments to each vector in the pair are made in the direction of the other vector:

$$\begin{aligned}\mathbf{a}' &= \mathbf{a} + k\mathbf{b}\\ \mathbf{b}' &= \mathbf{b} + k\mathbf{a}\end{aligned}$$

Since we want this dot product to be zero, we can solve for the scale k by taking the dot product of **a**′ and **b**′ and setting it equal to zero:

$$\begin{aligned}\mathbf{a}' \cdot \mathbf{b}' &= (\mathbf{a} + k\mathbf{b})(\mathbf{b} + k\mathbf{a}) = 0\\ &= \mathbf{a} \cdot \mathbf{b} + (\mathbf{a} \cdot \mathbf{a} + \mathbf{b} \cdot \mathbf{b})k + k^2 + \mathbf{a} \cdot \mathbf{b} = 0\end{aligned}$$

This equation produces a quadratic equation in k, but unfortunately, as we approach our solution point, this sets the second and third terms, which k depends on, to values near zero, creating numerical instability and consequently making this a poor approach for solving for k. A better approach is to use the approximation:

$$k \approx -\left(\frac{\mathbf{a} \cdot \mathbf{b}}{2}\right)$$

It is much easier to solve for this and iterate over it a couple of times. I.e. instead of using the approach above, we use:

$$x = \frac{2c}{-b \pm \sqrt{b^2 - 4ac}}$$

Where we note that the "standard" quadratic solution is given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

It is important to consider both of these approaches because of floating-point precision accuracy - in any case, as we approach the solution point, one of the two forms of these solutions will not perform well, and the other form will perform substantially better.

With this, we are able to iteratively solve for k until our two vectors have a dot product approximately equal to zero. Once this dot product is approximately zero, we are done! We have enforced orthonormality of the first two rows of the rotation matrix, which allows us to find an orthonormal third row of the rotation matrix by taking a cross product of the first two rows. Therefore, we then have an orthonormal rotation matrix.

20.2.5 Planar Target

Planar target is an example calibration object that possesses desirable properties, namely, as we will see below, that it reduces the number of correspondences we need from 7 to 5. Additionally, planar targets are generally easy to make, store, and allow for high camera calibration accuracy. They are typically mounted on the side of wheels so that they can be rotated. They are typically given analytically by the plane:

$$Z_s = 0, \text{ i.e. the XY-plane where Z is equal to 0.}$$

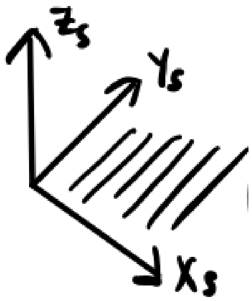


Figure 61: A geometric view of a planar target, which is a calibration target whose shape is geometrically described by a plane.

As a result of this, we no longer need to determine the rotation matrix coefficients r_{13}, r_{23}, and r_{33} (i.e. the third column of the rotation matrix, which determines how Z_s affects X_c, Y_c, and Z_c in the rotated coordinate system. Mathematically:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_s \\ Y_s \\ 0 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

As a result of this, our equations become:

$$\frac{x_I - x_O}{f} = \frac{X_c}{Z_c} = \frac{r_{11}X_s + r_{12}Y_s + t_x}{r_{31}X_s + r_{32}Y_s + t_z}$$

$$\frac{y_I - y_O}{f} = \frac{Y_c}{Z_c} = \frac{r_{21}X_s + r_{22}Y_s + t_y}{r_{31}X_s + r_{32}Y_s + t_z}$$

And, as we saw before, dividing these equations by one another yields a simplified form relative to our last division for the general case above:

$$\frac{x_I - x_O}{y_I - y_0} = \frac{\frac{r_{11}X_s + r_{12}Y_s + t_x}{r_{31}X_s + r_{32}Y_s + t_z}}{\frac{r_{21}X_s + r_{22}Y_s + t_y}{r_{31}X_s + r_{32}Y_s + t_z}}$$

$$= \frac{r_{11}X_s + r_{12}Y_s + t_x}{r_{21}X_s + r_{22}Y_s + t_y}$$

As we did before for the general case, cross-multiplying and rearranging gives:

$$(X_S y_I')r_{11} + (Y_S y_I')r_{12} + y_I' t_x - (X_S x_I')r_{21} - (Y_S x_I')r_{22} - x_I' t_y = 0$$

Now, rather than having 8 unknowns, we have 6: $r_{11}, r_{12}, r_{21}, r_{22}, t_x, t_y$. Because this is also a homogeneous equation, we again can account for scale factor ambiguity by setting one of DOF/parameters to a fixed value. This in turn reduces the DOF from 6 to 5, which means we only need 5 correspondences to solve for now (compared to 7 in the general case). As before, if we have more than 5 correspondences, this is still desirable as it will reduce estimation error and prevent overfitting.

One potential issue: if the parameter we fix, e.g. t_y, evaluates to 0 in our solution, this can produce large and unstable value estimates of other parameters (since we have to scale the parameters according to the fixed value). If the **fixed parameter** is close to 0, then we should fix a different parameter.

20.2.6 Aspect Ratio

This is no longer a common issue in machine vision now, but when aspect ratios were set differently in analog with photo-diode arrays for stepping in the x and y directions, this required setting an additional parameter to analytically describe the scale of x relative to y, and this parameter cannot be solved without a planar target.

20.2.7 Solving for t_z and f

Now that we have solved for our other parameters, we can also solve for t_z and f:

- These unknowns do not appear in other equations, but we can use our estimates from our other parameters to solve for t_z and f.
- We can again use the same equations that combine interior and exterior orientation:

$$
\begin{aligned}
(r_{11}X_s + r_{12}Y_s + r_{13}Z_s + t_x)f - x'_I t_z &= (r_{31}X_s + r_{32}Y_s + r_{33}Z_s)x'_I \\
(r_{21}X_s + r_{22}Y_s + r_{23}Z_s + t_y)f - y'_I t_z &= (r_{31}X_s + r_{32}Y_s + r_{33}Z_s)y'_I
\end{aligned}
$$

  Since we only have two unknowns (since we have found estimates for our other unknowns at this point), this now becomes a much simpler problem to solve. With just **1 correspondence**, we get both of the **2 equations** above, which is sufficient to solve for our **2 unknowns** that remain. However, as we have seen with other cases, using more correspondences still remains desirable, and can be solved via least squares to increase robustness and prevent overfitting.

- One problem with this approach: We need to have **variation in depth**. Recall that perspective projection has multiplication by f and division by Z; therefore, doubling f and Z results in no change, which means we can only determine t_z and f as a ratio $\frac{f}{t_z}$, rather than separately. To remove this consequence of scale ambiguity, we need **variation in depth**, i.e. the calibration object, such as a plane, **cannot be orthogonal to the optical axis**.

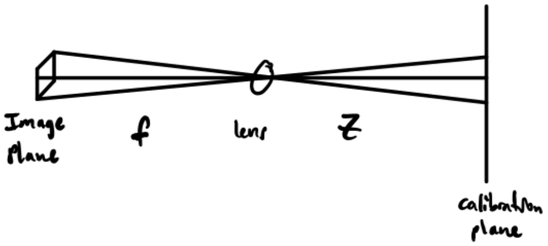


Figure 62: In order to obtain f and t_z separately, and not just up to a ratio, it is crucial that the calibration object does not lie completely orthogonal to the optical axis - else, these parameters exhibit high noise sensitivity and we can determine them separately.

It turns out this problem comes up in wheel alignment - if machine vision is used for wheel alignment, the calibration object will be placed at an angle of 45 or 60 degrees relative to the optical axis.

20.2.8 Wrapping it Up: Solving for Principal Point and Radial Distortion

To complete our joint interior/exterior orientation camera calibration problem, we need to solve for **principal point** and **radial distortion**.

As it turns out, there is unfortunately no closed-form solution for finding the principal point. Rather, we **minimize image error** using nonlinear optimization (e.g. Levenberg-Marquadt or Gauss-Newton). This error is the 2D error in the (x, y) image plane that describes the error between predicted and observed points in the image plane:

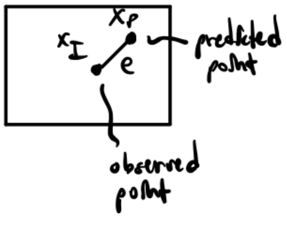


Figure 63: Image error that we minimize to solve for our principal point and radial distortion. We can formulate the total error from a set of observations as the squared errors from each correspondence. Note that x_I refers to the observed point, and x_P refers to the predicted point after applying our parameter estimate transformations to a point on the calibration object corresponding to an observed point.

Ideally, this error is zero after we have found an optimal set of parameters, i.e. for each i:

$$\begin{aligned}x_{I_i} - x_{P_i} &= 0\\ y_{I_i} - y_{P_i} &= 0\end{aligned}$$

Where do x_{P_i} and y_{P_i} come from? They come from applying:

- **Rotation matrix estimate ($\mathbf{R}$)**
- **Translation vector estimate ($\mathbf{t}$)**
- **Principal Point Estimate** (x_0, y_0)
- **Radial Distortion Polynomial** $(k_1, k_2, \cdots)$

Using these equations, we can take a sum of squared error terms and solve for our parameters using nonlinear optimization with Levenberg-Marquadt (LM). This can be implemented using **LMdif** in the **MINPACK** package (Fortran), as well as other packages in other languages. Recall that LM performs nonlinear optimization with L2 regularization.

One small problem with this approach: LM only works for unconstrained optimization, which is not the case when we only use orthonormal rotation matrices. To overcome this, **Tsai's Calibration Method** used **Euler Angles**, and we can use either **Gibb's vector** or **unit quaternions**. Comparing/contrasting each of these 3 rotation representations to solve this problem:

- **Euler Angles**: This representation is non-redundant, but "blows up" if we rotate through 180 degrees.
- **Gibb's Vector**: This representation is also non-redundant, but exhibits a singularity at $\theta = \pi$.
- **Unit Quaternions**: This representation is redundant but exhibits no singularities, and this redundancy can be mitigated by adding an additional equation to our system:

$$\overset{o}{q} \cdot \overset{o}{q} - 1 = 0$$

Finally, although LM optimization finds **local**, and not **global** extrema, we have already developed an initial estimate of our solution through the application of the combined interior/exterior orientation equations above, which ensures that extrema that LM optimization finds are indeed not just locally optimal, but globally optimal.

20.2.9 Noise Sensitivity/Noise Gain of Approach

As we have asked for other systems, it is important to consider how robust this approach is to noise and perturbations. A few notes/concluding thoughts:

- As we saw above, this approach does not produce good estimates for our parameters t_z and f if the calibration object is orthogonal to the optical axis. In addition to not being able to separate the estimates for these parameters, both of these parameters (because they are defined by a ratio) have a very high/infinite noise gain.
- Since these methods are largely based off of numerical, rather than analytical methods, we can best evaluate noise gain using **Monte Carlo** methods, in which we use noise that is well-defined by its statistical properties, e.g. zero-mean Gaussian noise with finite variance denoted by σ^2. We can analyze the estimated noise gain of the system by looking at the parameter distribution in the parameter space after applying noise to our observed values and making solution predictions.
- Higher-order coefficients of radial distortion are highly sensitive to noise/exhibit high noise gain.

21 Lecture 22: Exterior Orientation, Recovering Position and Orientation, Bundle Adjustment, Object Shape

Today, we will conclude our discussion with photogrammetry with more discussion on **exterior orientation**. We will focus on how exterior orientation can be combined with other photogrammetry techniques we've covered for recovering position and orientation, solving the bundle adjustment problem, and determining optimal representations for object shape for solving alignment and recognition problems.

21.1 Exterior Orientation: Recovering Position and Orientation

Consider the problem of a drone flying over terrain:

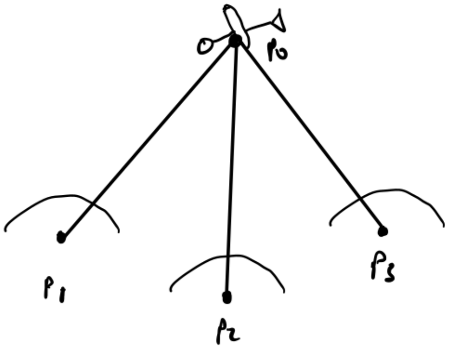


Figure 64: Example problem of recovering position and orientation: a drone flying over terrain that observes points for which it has correspondences of in the image plane. The ground points are given by p_1, p_2, and p_3, and the point of interest is p_0. This photogrammetry problem is sometimes referred to as Church's tripod.

Problem Setup:

- Assume that the ground points p_1, p_2, and p_3 are known, and we want to solve for p_0.
- We also want to find the attitude of the drone in the world: therefore, we solve for **rotation** + **translation** (6 DOF).
- We have a mix of 2D-3D correspondences: 2D points in the image plane that correspond to points in the image, and 3D points in the world.
- Assume that the **interior orientation** of the camera is known (x_0, y_0, f).
- Connect image to the center of projection.

How many correspondences do we Need? As we have seen with other frameworks, this is a highly pertinent question that is necessary to consider for photogrammetric systems.

Since we have 6 DOF with solving for **translation** + **rotation**, we need to have at least 3 correspondences.

21.1.1 Calculating Angles and Lengths

Next, we need to figure out how to solve for the angles between ground points, and the lengths to these ground points from the plane:

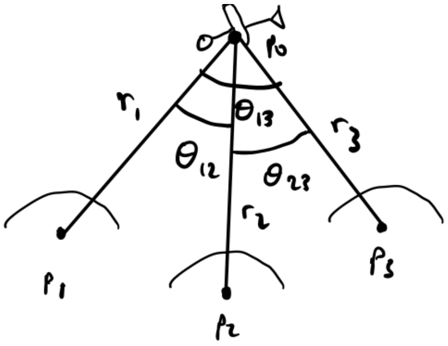


Figure 65: As a next step of solving this exterior orientation problem, we calculate the angles between the ground points relative to the plane, as well as the lengths from the plane to the ground points. The angles of interest are given by: θ_{12}, θ_{13}, and $\theta_{2,3}$, and the lengths of interest are r_1, r_2, and r_3.

- Given the problem setup above, if we have rays from the points on the ground to the points in the plane, we can calculate angles between the ground points using dot products (cosines), cross products (sines), and arc tangents (to take ratios of sines and cosines).

- We also need to know the lengths of the tripod - i.e. we need to find r_1, r_2, and r_3.
- From here, we can find the 3D point p_0 by finding the intersection of the 3 spheres corresponding to the ground points (center of spheres) and the lengths from the ground points to the plane (radii):

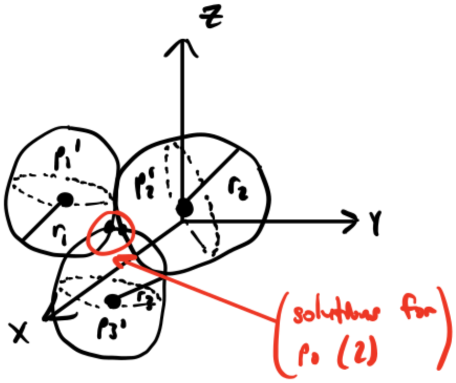


Figure 66: We can find the location of the plane p_0 by finding the intersection of 3 spheres using the point information p_i and the length information r_i.

> Note that with this intersection of spheres approach, there is ambiguity with just three points/correspondences (this gives 2 solutions, since the intersection of 3 spheres gives 2 solutions). Adding more points/correspondences to the system reduces this ambiguity and leaves us with 1 solution.
>
> Another solution issue that can actually help us to reduce ambiguity with these approaches is that mirror images have a mirror cyclical ordering of the points that is problematic. This allows us to find and remove "problematic solutions".

What if I only care about attitude? That is, can I solve for only rotation parameters? Unfortunately not, since the variables/parameters we solve for are coupled to one another.

Some laws of trigonometry are also helpful here, namely the law of sines and cosines:

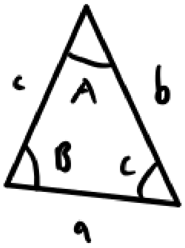


Figure 67: For any triangle, we can use the law of sines and law of cosines to perform useful algebraic and geometric manipulations on trigonometric expressions.

Law of Sines: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$

Law of Cosines: $a^2 = b^2 + c^2 - 2bc\cos A$

We do not use these approaches here, but it turns out we can solve for the lengths between the different ground points (r_{12}, r_{23}, and r_{13}) using these trigonometric laws.

21.1.2 Finding Attitude

From here, we need to find **attitude** (rotation of the plane relative to the ground). We can do this by constructing three vectors with the translation subtracted out (as we have seen with previous approaches):

$$\mathbf{a}_1 \iff \mathbf{b}_1 = \mathbf{p}_1 - \mathbf{p}_0 \tag{54}$$

$$\mathbf{a}_2 \iff \mathbf{b}_2 = \mathbf{p}_2 - \mathbf{p}_0 \tag{55}$$

$$\mathbf{a}_3 \iff \mathbf{b}_3 = \mathbf{p}_3 - \mathbf{p}_0 \tag{56}$$

Note that the vectors $\mathbf{a}_1, \mathbf{a}_2$, and $\mathbf{a}_3$ are our correspondences in the **image plane**. This means that we can now relate these two coordinate systems. We can relate these via a rotation transformation:

$$R(\hat{\mathbf{a}}_i) = \hat{\mathbf{b}}_i \;\forall\; i \in \{1, 2, 3\}$$

We can first represent R as an orthonormal rotation matrix. It turns out we can actually solve this by considering all three correspondences at once:

$$R(\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{a}}_3) = (\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3)$$

We can solve for R using matrix inversion, as below. Question to reflect on: Will the matrix inverse result be orthonormal?

$$\begin{aligned}
&A \triangleq (\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{a}}_3) \\
&B \triangleq (\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3) \\
&RA = B \implies R = BA^{-1}
\end{aligned}$$

In practice, as we saw with other photogrammetry frameworks, we would like to have more than 3 correspondences in tandem with least squares approach. Can use estimates with first 3 correspondence to obtain initial guess, and then iteration to solve for refined estimate. May reduce probability of converging to local minima.

Note that errors are in image, not in 3D, and this means we need to optimize in the image plane, and not in 3D.

21.2 Bundle Adjustment

Bundle Adjustment (sometimes abbreviated as BA) is a related photogrammetry problem in which, in the most general case, consists of K landmarks (points in the image observed by cameras) and N cameras.

Goal of BA: Determine the 3D locations of landmark objects and cameras in the scene relative to some global coordinate system, as well as determine the orientation of cameras in a global frame of reference.

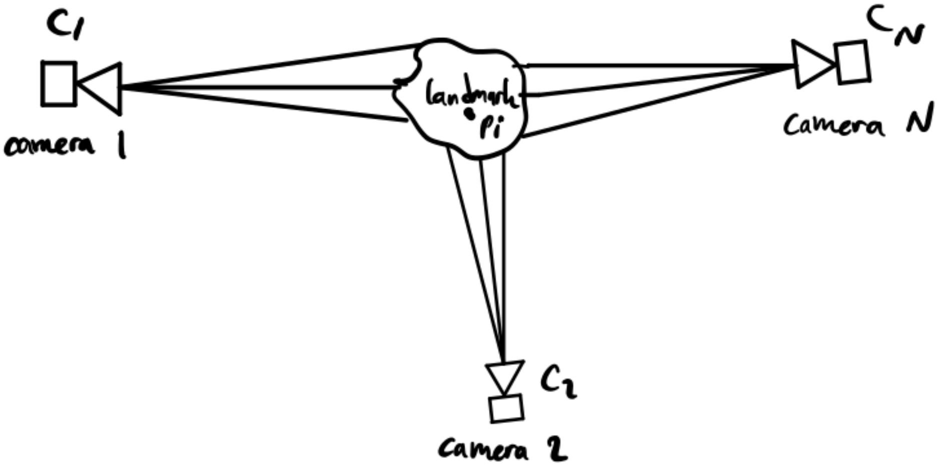


Figure 68: Bundle Adjustment (BA). In the general case, we can have any number of K landmark points ("interesting" points in the image) and N cameras that observe the landmarks.

What are our unknowns for Bundle Adjustment

- **Landmarks** ($\{p_i\}$): A set of points that are observed by multiple views, of which we want to find the coordinates of in a global coordinate system.
- **Cameras** ($\{c_i\}$): This is the set of camera locations in space with respect to some global coordinate system.
- **Camera Attitudes** ($\{R_i\}$): These are the orientations of the cameras in space with respect to some global coordinate system.
- **Intrinsic Orientations** ($\{x_{0_i}, y_{0_i}, f_i, K_i\}$: These refer to the camera intrinsics for each camera in our problem.

This approach is typically solved using **Levenberg-Marquadt (LM)** nonlinear optimization. Although there are many parameters to solve for, we can make an initial guess (as we did with our previous camera calibration approaches to avoid local minima) to ensure that we converge to **global**, rather than **local** minima.

How do we find "interesting points" in the image? One way to do this is to use descriptors (high-dimensional vectors corresponding to image gradients), as is done with SIFT (Scale-Invariant Feature Transforms). Some other, more recent approaches include:

- SURF (Speeded Up Robust Features)
- ORB (Oriented FAST and rotated BRIEF)
- BRIEF (Binary Robust Independent Elementary Features)
- VLAD (Vector of Locally Aggregated Descriptors)

21.3 Recognition in 3D: Extended Gaussian 3D

Recall our previous discussions with recognition in 2D - let's now aim to extend this to 3D!

Goal of 3D Recognition: Describe a 3D object.

Let's consider several representations for effective **alignment estimation** and **recognition**:

- **Polyhedra**: When this is our object representation, we often consider this to be a class of "block world problems". We can describe these polyhedra objects semantically with edges, faces, and linked data structures. Because this is not a realistic representation of the world, for systems in practice, we look for more complicated representations.
- **Graphics/curves**: These can be done with a mesh. Here, we consider any curved surface. Meshes can be thought of as polyhedral objects with many facets (polygon faces). This representation is well-suited for outputting pictures.

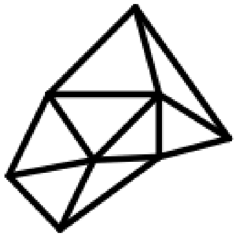


Figure 69: Example of a mesh. Note that this can be constructed as many facets, where each facet is a polygon.

  What kind of tasks are we interested in for meshes, and more generally, the rest of our object representations?

  - Find **alignment/pose** (position/orientation): For alignment, we can accomplish this task by assigning correspondences between vertices, but this is not a very effective approach because there is no meaning behind the vertices (i.e. they are not necessarily deemed "interesting" points), and the vertex assignments can change each time the shape is generated. Can do **approximate alignment** by reducing the distance to the nearest vertex iteratively and solving.
  - Object **recognition**: We cannot simply compare numbers/types of facets to models in a library, because the generated mesh can change each time the mesh is generated. It turns out that this is a poor representation for recognition.

Since neither of these representations lend themselves well for these two tasks, let us consider alternative representations. First, let us consider what characteristics we look for in a representation.

21.3.1 What Kind of Representation Are We Looking For?

We are looking for an object representation that exhibits the following properties:

- **Translation Invariance**: Moving an object in a coordinate system does not change (at least not substantially) the object's representation.
- **Rotation Consistency**: Rotating the object changes the representation of the object in an understandable and efficient way that can be applied for alignment problems.

Representations to consider given these properties:

- **Generalized Cylinders**:
  - These can be thought of as extruding a circle along a line.
  - In the generalized case, we can extrude an arbitrary cross-section along a line, as well as allow the line to be a general curve and to allow the cross-section (generator) to vary in size.

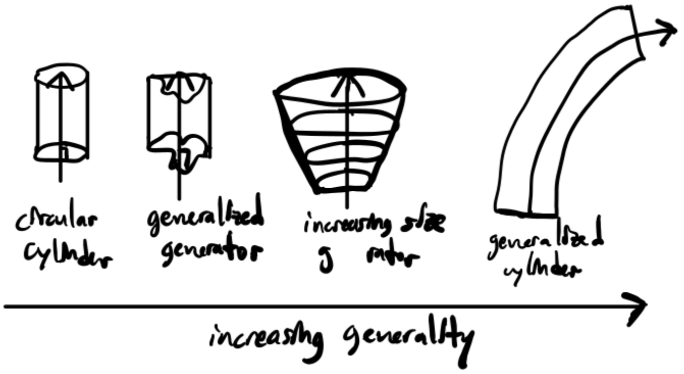


Figure 70: Representations of increasing generality for generalized cylinders.

Example: Representing a sphere as a generalized cylinder:

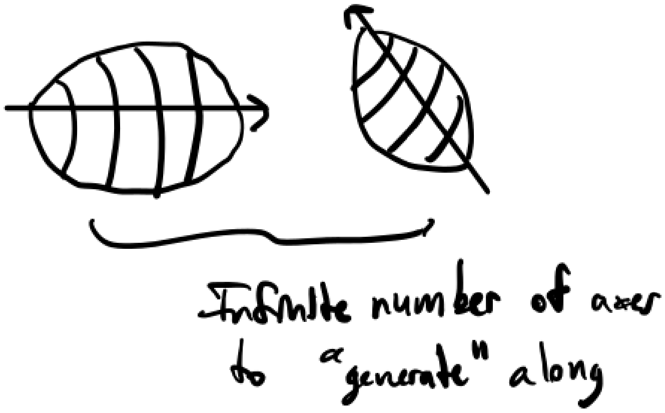


Figure 71: Representing a sphere as a generalized cylinder - unfortunately, there are an infinite number of axes we can use to "generate" the cylinder along.

  - This same problem shows up elsewhere, especially when we allow for inaccuracies in data.
  - This approach has not been overly successful in solving problems such as alignment, in practice.
  - This leads us to pursue an alternative representation.

- **Polyhedra Representation**:
  - Let's briefly revisit this approach to get a good "starting point" for our object representation that we can solve **alignment** and **recognition** problems.
  - One way, as we have seen before, was to take edges and faces, and to build a graph showing connectedness of polyhedra.
  - Instead, we take the unit normal vector of faces and multiply them by the area of the face/polygon.

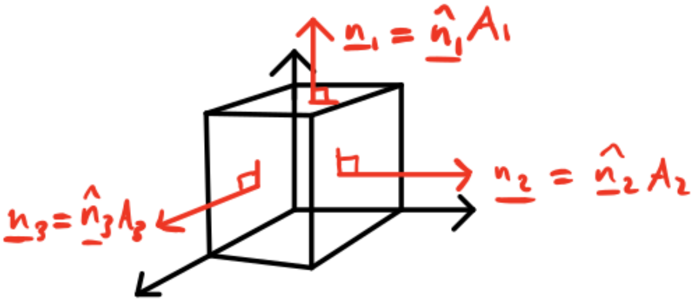

$\underline{n}_1 = \underline{\hat{n}}_1 A_1$
$\underline{n}_2 = \underline{\hat{n}}_2 A_2$
$\underline{n}_3 = \underline{\hat{n}}_3 A_3$

Figure 72: Polyhedra approach in which we represent faces by the unit normal vector multiplied by the area of the face of the polyhedra representation.

- Despite throwing away information about the connectedness of faces, Minkowski's proof shows that this generates a unique representation for **convex polyhedra**.
- Note that this proof is non-constructive, i.e. there was no algorithm given with the proof that actually solves the construction of this representation. There is a construction algorithm that solves this, but it is quite slow, and, as it turns out, are **not needed for recognition tasks**.
- Further, note that the sum of vectors form a closed loop:

$$\sum_{i=1}^{N} \mathbf{n}_i = 0$$

Therefore, any object that does not satisfy this is **non-convex polyhedra**.

Let's consider what this looks like for the following cylindrical/conic section. Consider the surface normals on the cylindrical component $(A_i\hat{\mathbf{n}}_i)$ and the surface normals on the conic component $(B_i\hat{\mathbf{n}}_i)$. Note that we still need to be careful with this representation, because each mesh generated from this shape can be different.

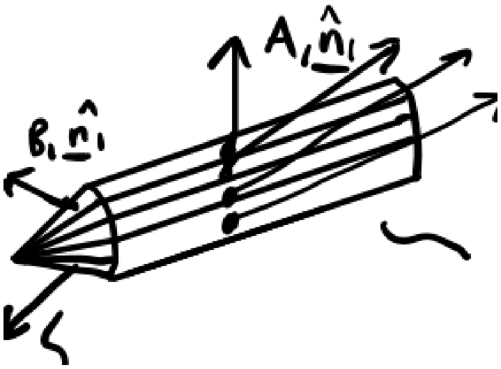


Figure 73: A shape represented by a mesh that we will convert to an extended Gaussian sphere, as we will see below. Note that the surface normals $A_i\hat{\mathbf{n}}_i$ and $B_i\hat{\mathbf{n}}_i$ each map to different great circles on the sphere, as can be seen in the figure below.

Idea: Combine facets that have similar surface normals and represent them on the same great circle when we convert to a sphere.

What does this look like when we map these scaled surface normals to a unit sphere?

Idea: Place masses in great circle corresponding to the plane spanned by the unit normals of the **cylindrical** and **conic** sections above. Note that the mass of the back plate area of our object occupies a single point on the mapped sphere, since every surface normal in this planar shape is the same.

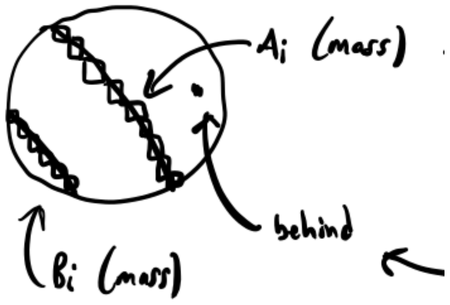


Figure 74: Extended Gaussian sphere mapping, which maps the surface normals of arbitrary (convex) surfaces to the sphere such that the surface vector corresponds to a scaled unit normal in both the original surface and the sphere.

Therefore, this sphere becoems our representation! This works much better for **alignment** and **recognition**. Note that we work in a sphere space, and not a high-dimensional planar/Euclidean space. Therefore:

- Comparisons (e.g. for our recognition task) can be made by comparing masses using distance/similarity functions.
- Alignment can be adjusted by rotating these spheres.

How well does this representation adhere to our desired properties above?

- **Translation Invariance**: Since surface normals and areas do not depend on location, translation invariance is preserved.
- **Rotation Equivariance**: Since rotating the object corresponds to just rotating the unit sphere normals without changing the areas of the facets, rotation of the object simply corresponds to an equal rotation of the sphere.

Loosely, this representation corresponds to **density**. This density is dependent on the degree of curvature of the surface we are converting to a sphere representation:

- Low density ↔ High curvature
- High density ↔ Low curvature

To better understand this representation in 3D, let us first understand it in 2D.

21.3.2 2D Extended Circular Image

The idea of the extended Gaussian Image: what do points on an object and points on a sphere have in common? They have the same surface normals.

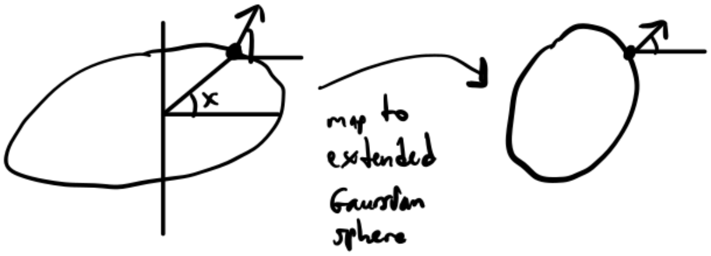


Figure 75: Representation of our extended Gaussian image in 2D. The mapping between the surface and the sphere is determined by the points with the same surface normals.

Gauss: This extended Gaussian image representation is a way to map arbitrary convex objects to a sphere by mapping points with the same surface normals.

We can make this (**invertible**) mapping in a **point-point** fashion, and subsequently generalize this to mapping entire **convex shapes** to a sphere. There are issues with non-convex objects and invertibility because the forward mapping becomes a **surjective mapping**, since multiple ponts can have the same surface normal, and therefore are mapped to the same location on the circle.

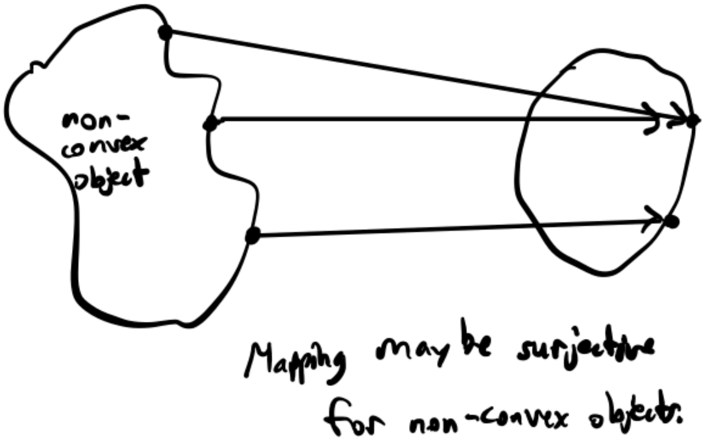


Figure 76: When surfaces are non-convex, we run into issues with inverting our mapping, because it may be surjective, i.e. some points on the surface may map to the same point on the sphere due to having the same surface normal.

Idea: Recall that our idea is to map from some convex 2D shape to a circle, as in the figure below.

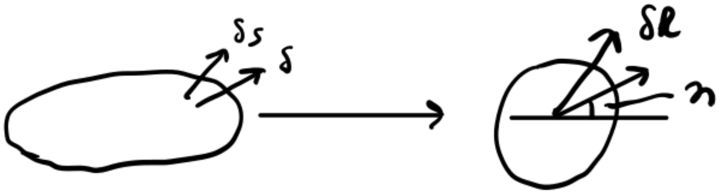


Figure 77: Our forward mapping from our surface to a circle.

Note that in this case, density depends on which region (and consequently the degree of curvature) of the object above. We are interested in this as a function of angle η. Analytically, our **curvature** and **density** quantities are given by:

- **Curvature**: $\kappa = \frac{d\eta}{dS} = \frac{1}{R_{\text{curvature}}}$ (The "turning rate")
- **Density**: $G = \frac{1}{\kappa} = \frac{dS}{d\eta}$ (inverse of curvature)

These are the only quantities we need for our representation mapping! We just map the density $G = \frac{dS}{d\eta}$ from the object to the unit sphere. This is invertible in 2D for convex objects, though this invertibility is not necessarily the case in 3D.

21.3.3 Analyzing Gaussian Curvature in 2D Plane

Let us analyze this mapping in the 2D plane, using the figure below as reference:

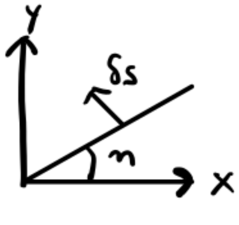


Figure 78: Analyzing Gaussian curvature in the 2D plane.

Mathematically, then, we can say:

$$\delta_x = -\sin(\eta)$$
$$\delta_y = \cos(\eta)$$

Then we can integrate to find x and y:

$$
\begin{aligned}
&x = x_0 + \int -\sin(\eta)dS \\
&x = x_0 + \int -\sin(\eta)\frac{dS}{d\eta}d\eta \\
&x = x_0 + \int -\sin(\eta)G(\eta)d\eta \\
&\text{And similarly for } y: \\
&y = y_0 + \int \cos(\eta)G(\eta)d\eta
\end{aligned}
$$

What happens if we integrate over the entire unit circle in 2D? Then we expect these integrals to evaluate to 0:

$$
\begin{aligned}
&\int_0^{2\pi} -\sin(\eta)G(\eta)d\eta = 0 \\
&\int_0^{2\pi} \cos(\eta)G(\eta)d\eta = 0
\end{aligned}
$$

These conditions require that the centroid of **density** $G(\eta)$ is at the origin, i.e. that the weighted sum of $G(\eta) = 0$.

21.3.4 Example: Circle of Radius R

In the case of a circle with constant radius, it turns out our curvature and density will be constant:

$$
\begin{aligned}
K &= \frac{d\eta}{dS} = \frac{1}{R} \\
G(\eta) &= \frac{dS}{d\eta} = R
\end{aligned}
$$

Note that while this holds for spheres, we can actually generalize it beyond spheres too (making it of more practical value). This is because we can fit curves of arbitrary shape by fitting a circle locally and finding the radius of the best-fitting circle.

Note that because density is equal to R, this density increases with increasing radius. Because density is the same everywhere, we cannot recover orientation as there is not uniqueness of each point. Let's try using an ellipse in 2D.

21.3.5 Example: Ellipse in 2D

Recall that there are several ways to parameterize an ellipse:

- **Implicitly**: $\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$

- **"Squashed circle"**: $x = a\cos\theta,\ y = b\sin\theta$ (note that θ is the variable we parameterize these parametric equations over. Because this takes a parametric form, this form is better for generating ellipses than the first implicit form because we can simply step through our parameter θ.

Next, let's compute the normal vector to the curve to derive a surface normal. We can start by computing the tangent vector. First, note that the vector $\mathbf{r}$ describing this parametric trajectory is given by:

$$\mathbf{r} = (a\cos\theta, b\sin\theta)^T$$

Then the tangent vector is simply the derivative of this curve with respect to our parameter θ:

$$\mathbf{t} = \frac{d\mathbf{r}}{d\theta} = (-a\sin\theta, b\cos\theta)^T$$

Finally, in 2D we can compute the normal vector to the surface by switching x and y of the tangent vector and inverting the sign of y (which is now in x's place):

$$\mathbf{n} = (b\cos\theta, a\sin\theta)T$$

We want this vector to correspond to our normal in the sphere:

$$\mathbf{n} = (b\cos\theta, a\sin\theta)T \leftrightarrow \hat{\mathbf{n}} = (\cos\eta, \sin\eta)^T$$

Then we have:

$$\begin{aligned}
&b\cos\theta = n\cos\eta \\
&a\sin\theta = n\sin\eta \\
&n^2 = b^2\cos^2\theta + a^2\sin^2\theta \\
&\mathbf{s} = (a\cos\eta, b\sin\eta)^T \\
&K = \left(\frac{\mathbf{s}\cdot\mathbf{s}}{ab}\right)^{\frac{3}{2}} \\
&n^2((a\cos\eta)^2 + (b\sin\eta)^2) = (ab)^2
\end{aligned}$$

These derivations allow us to find the extrema of curvature: ellipses will have 4 of them, spaced $\frac{\pi}{2}$ apart from one another (alternating minimums and maximums): $\eta = 0, \eta = \frac{\pi}{2}, \eta = \pi, \eta = \frac{3\pi}{2}$.

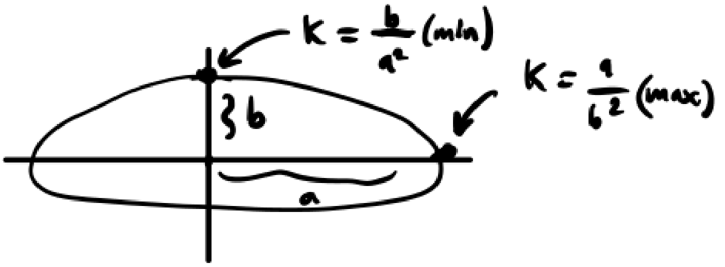


Figure 79: Curvature extrema of an ellipse: An ellipse has 4 extrema, spaced $\frac{\pi}{2}$ apart from one another. Extrema depend on the major and minor axes of the ellipse.

If a is the major axis and b is the minor axis, then extrema of curvature are given by:

- **Maximum**: $\kappa = \frac{a}{b^2}$
- **Minimum**: $\kappa = \frac{b}{a^2}$

Without loss of generality, we can flip a and b if the major and minor axes switch.

For alignment, we can rotate these ellipses until their orientations match. If the alignment is not constructed well, then the object is not actually an ellipse.

Geodetic Identity: Related, this identity relates geocentric angles to angles used for computing latitudes:

$$\begin{aligned}
&b\cos\theta = n\cos\eta \\
&a\sin\theta = n\sin\eta \\
&a(\tan\theta) = b(\tan\eta)
\end{aligned}$$

What are some other applications of this in 2D? We can do (i) **circular filtering** and **circular convolution**!

21.3.6 3D Extended Gaussian Images

Fortunately, many of the results in 2D Gaussian images generalize well to 3D. As with previous cases, we can begin with the Gauss mapping (starting with points, but generalizing to different convex shapes in the surface). From here, we can compute **curvature** and **density**:

- **Curvature**: $\kappa = \frac{\delta S}{\delta O} = \lim_{\delta \to 0} \frac{\delta S}{\delta O} = \frac{dS}{dO}$
- **Density**: $G(\eta) = \frac{\delta O}{\delta S} = \lim_{\delta \to 0} \frac{\delta O}{\delta S} = \frac{dO}{dS}$

Although curvature in 3D is more complicated, we can just use our notion of Gaussian curvature, which is a single scalar.

As long as our object is convex, going around the shape in one direction in the object corresponds to going around the sphere in the same direction (e.g. counterclockwise).

Example of non-convex object: Saddle Point: As we mentioned before, we can run into mapping issues when we have non-convex objects that we wish to map, such as the figure below:

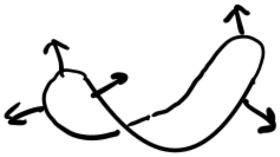


Figure 80: An example of a non-convex object: a saddle point. Notice that some of the surface normals are the same, and therefore would be mapped to the same point on the sphere.

Note further that traveling/stepping an object in the reverse order for a non-convex object inverts the curvature κ, and therefore allows for a negative curvature $\kappa < 0$.

Example of 3D Gaussian Curvature: Sphere of Radius R: This results in the following (constant) curvature and density, as we saw in the 2D case:

- **Curvature**: $\kappa = \frac{1}{R^2}$
- **Density**: $G(\eta) = R^2$

Extensions: Integral Curvature: Next time, we will discuss how Gaussian curvature allows for us to find the integral of Gaussian curvature for recognition tasks.

22 Lecture 23: Gaussian Image and Extended Gaussian Image, Solids of Revolution, Direction Histograms, Regular Polyhedra

In this lecture, we cover more on the Gaussian images and extended Gaussian images (EGI), as well as integral Gaussian images, some properties of EGI, and applications of these frameworks.

22.1 Gaussian Curvature and Gaussian Images in 3D

Let's start with defining what a Gaussian image is:

Gaussian Image: A correspondence between points on an image and corresponding points on the unit sphere based off of equality of the surface normals.

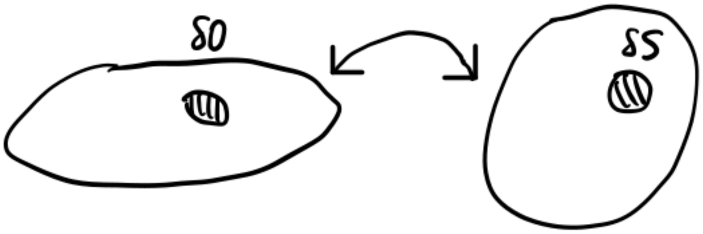


Figure 81: Mapping from object space the Gaussian sphere using correspondences between surface normal vectors. Recall that this method can be used for tasks such as image recognition and image alignment.

Recall that we defined the **Gaussian curvature** of an (infinitesimal) patch on an object as a scalar that corresponds to the ratio of the areas of the object and the Gaussian sphere we map the object to, as we take the limit of the patch of area on the object to be zero:

$$\begin{aligned}\kappa_{\text{Gaussian}} &= \lim_{\delta \to 0} \frac{\delta S}{\delta O} \\ &= \text{ The ratio of areas of the image and the object}\end{aligned}$$

Rather than use Gaussian curvature, however, we will elect to use the inverse of this quantity (which is given by density) $\frac{1}{\kappa}$, which roughly corresponds to the amount of surface that contains that surface normal.

We can also consider the integral of Gaussian curvature, and the role this can play for dealing with object discontinuities:

22.1.1 Gaussian Integral Curvature

The integral of Gaussian curvature is given by:

$$\iint_O K dO = \iint_O \frac{dS}{dO} dO = \iint_O dS = S \text{ (The area of a corresponding patch on the sphere)}$$

Note: One nice feature of **integral curvature**: This can be applied when curvature itself cannot be applied, e.g. at discontinuities such as edges and corners.

We can also apply the integral over the Gaussian sphere:

$$\iint_S \frac{1}{K} dS = \iint_S \frac{dO}{dS} dS = \iint_S dO = O \text{ (The area of a corresponding path on the object)}$$

This second derivation is used more often in practice.

22.1.2 How Do We Use Integral Gaussian Curvature?

As we briefly mentioned above, integral Gaussian curvature can be used for "smoothing out" discontinuities and providing continuity when there is none guaranteed by the curvature function itself. Discontinuities can be caused by features such as corners, edges, and other abrupt changes in surface orientation.

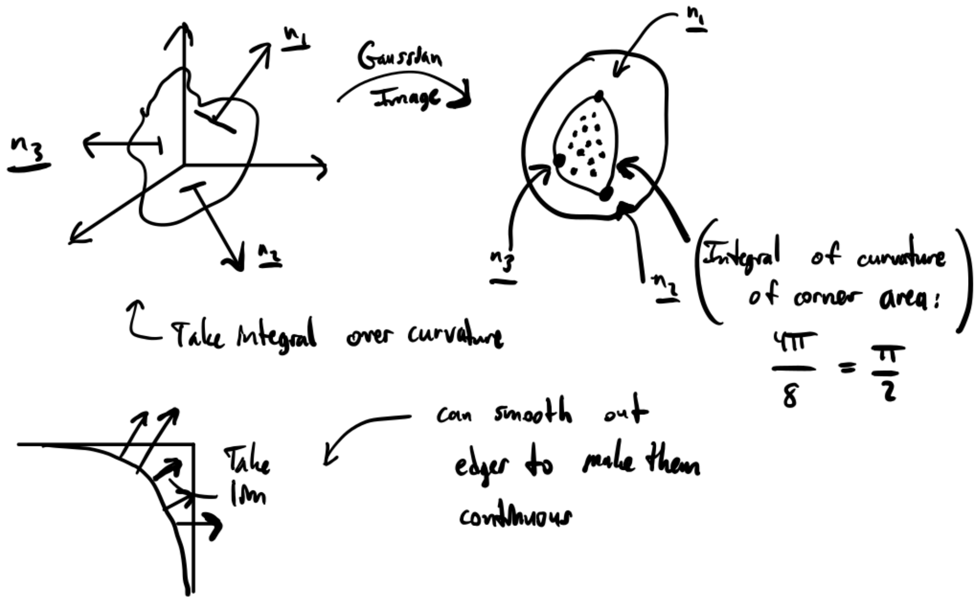

n_1
Gaussian Image
n_3
n_2
n_1
n_3
n_2
(Integral of curvature of corner area: 4π/8 = π/2)
Take Integral over curvature
Take lim
can smooth out edges to make them continuous

Figure 82: An application in which integral Gaussian curvature can be used in lieu of the curvature function. The smoothing properties of integration enable for the mitigation of discontinuity affects.

Next, let's consider the different forms our mass density distribution G can take.

22.1.3 Can We Have Any Distribution of G on the Sphere?

It is important to understand if we need to impose any restrictions on our mass density distribution. Let's analyze this in both the discrete and continuous cases. The results from the discrete case extend to the continuous case.

- **Discrete Case**: Consider an object that we have discretized to a set of polygon facets, and consider that we have a camera that observes along an optical axis $\hat{\mathbf{v}}$, and a mirror image of this camera that observes along the optical axis $-\hat{\mathbf{v}}$. Note that any surface normals that are not parallel to these optical axes will have foreshortening effects imposed.

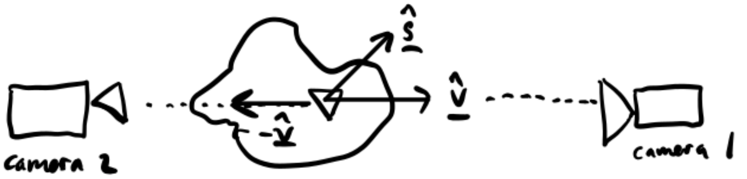


Figure 83: Problem setup in the discrete case. Consider our object parameterized by a discrete number of finite polygon facets, and a camera and its mirror image observing the different surface normals of these facets from two opposite vantage points.

First, consider only taking the facets with positive dot product (i.e. 90 degrees or less) away from the vector $\hat{\mathbf{v}}$, and sum the product of these facet area multiplied by normal and the unit vector corresponding to the optical axis $\hat{\mathbf{v}}$:

$$\sum_{\hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}}$$

Now, consider the other facets, that is, those for which we take the facets with positive dot product (i.e. 90 degrees or less) away from the vector $-\hat{\mathbf{v}}$, which is 180 degrees rotated from $\hat{\mathbf{v}}$, and applying the same operation:

$$\sum_{\hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}}) \geq 0} A_i \hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}})$$

We must have equality between these two sums:

$$\sum_{\hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} = \sum_{\hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}}) \geq 0} A_i \hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}})$$

And therefore, if we combine the sums to the same side:

$$\sum_{\hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} - (\sum_{\hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}}) \geq 0} A_i \hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}})) = \mathbf{0}$$
$$\sum_{\hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} - (-\sum_{\hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}}) \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}}) = \mathbf{0}$$
$$\sum_{\hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} + \sum_{\hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}}) \geq 0} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} = \mathbf{0}$$
$$\sum_{\{\hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} \geq 0\} \cup \{\hat{\mathbf{s}}_i \cdot (-\hat{\mathbf{v}}) \geq 0\}} A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} = \mathbf{0}$$
$$\sum_i A_i \hat{\mathbf{s}}_i \cdot \hat{\mathbf{v}} = \mathbf{0}$$
$$(\sum_i A_i \hat{\mathbf{s}}_i) \cdot \hat{\mathbf{v}} = \mathbf{0} \; \forall \; \hat{\mathbf{v}}$$

Therefore, from this it must be the case that $\sum_i A_i \hat{\mathbf{s}}_i = \mathbf{0}$, i.e. that the center of mass of the mapping must be at the center of the sphere. This also means that the Gaussian density G must have a "zero-mean" mass distribution for any closed object. This does not generally hold for any non-closed objects.

- **Continuous Case**: This same result holds in the continuous case - the mean must be at the center of the sphere for closed objects. Mathematically:

$$\iint_S G(\hat{\mathbf{s}}) \hat{\mathbf{s}} dS = \mathbf{0}$$

Therefore, for both the discrete and continuous cases, we can think of the Extended Gaussian Image (EGI) as a mass distribution over the sphere that objects are mapped to according to their surface normals.

Next, let's look at some discrete applications. Continuous EGI can be useful as well, especially if we are using an analytical model (in which case we can compute the exact EGI).

22.2 Examples of EGI in 3D

Let's start with some examples of EGI in 3D.

22.2.1 Sphere: EGI

We have seen this before for the circle, and can simply generalize this from 2D to 3D:

- **Curvature**: $\kappa = R^2$
- **Density**: $G = \frac{1}{K} = \frac{1}{R^2}$

We saw that because the curvature and density are the same everywhere, that this imposes limitations on the effectiveness of this surface for recognition and alignment tasks. Let's next consider something with more complicated structure.

22.2.2 Ellipsoid: EGI

Recall our two main ways to parameterize ellipses/ellipsoids: implicit (the more common form) and "squashed circle/sphere" (in this case, the more useful form:

- **Implicit parameterization**: $\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1$
- **"Sphere Squashing" parameterization**: These are parameterized by θ and ϕ:

$$\begin{aligned} x &= a\cos\theta\cos\phi \\ y &= b\sin\theta\cos\phi \\ z &= c\sin\phi \end{aligned}$$

We will elect to use the second of these representations because it allows for easier and more efficient generation of points by parameterizing θ and ϕ and stepping these variables.

As we did last time for the 2D ellipse, we can express these parametric coordinates as a vector parameterized by θ and ϕ:

$$\mathbf{r} = (a\cos\theta\cos\phi, b\sin\theta\cos\phi, c\sin\phi)^T$$

And similarly to what we did before, to compute the **surface normal** and the **curvature**, we can first find (in this case, two) tangent vectors, which we can find by taking the derivative of $\mathbf{r}$ with respect to θ and ϕ:

$$\begin{aligned} \mathbf{r}_\theta &\triangleq \frac{d\mathbf{r}}{d\theta} = (-a\sin\theta\cos\phi, b\cos\theta\cos\phi, 0)^T \\ \mathbf{r}_\phi &\triangleq \frac{d\mathbf{r}}{d\phi} = (-a\cos\theta\sin\phi, -b\sin\theta\sin\phi, c\cos\phi)^T \end{aligned}$$

Then we can compute the normal vector by taking the cross product of these two tangent vectors of the ellipsoid. We do not carry out this computation, but this cross product gives:

$$\begin{aligned} \mathbf{n} = \mathbf{r}_\theta \times \mathbf{r}_\phi &= \cos\phi(b\cos\theta\cos\phi, ac\sin\theta\cos\phi, ab\sin\phi)^T \\ &= (b\cos\theta\cos\phi, ac\sin\theta\cos\phi, ab\sin\phi)^T \end{aligned}$$

Where we have dropped the $\cos\phi$ factor because we will need to normalize this vector anyway. Note that this vector takes a similar structure to that of our original parametric vector $\mathbf{r}$.

With our surface normals computed, we are now equipped to match surface normals on the object to the surface normals on the unit sphere. To obtain **curvature**, our other quantity of interest, we need to differentiate again. We will also change our parameterization from $(\theta, \phi) \rightarrow (\xi, \eta)$, since (θ, ϕ) parameterize the object in its own space, and we want to parameterize the object in unit sphere space (ξ, η).

$$\begin{aligned}
\hat{\mathbf{n}} &= (\cos\xi\cos\eta, \sin\xi\cos\eta, \sin\eta)^T \\
\bar{\mathbf{s}} &= (a\cos\xi\cos\eta, b\sin\xi\cos\eta, \sin\eta)^T \\
\kappa &= \left(\frac{\mathbf{s}\cdot\mathbf{s}}{abc}\right)^2 \\
G &= \frac{1}{K} = \left(\frac{abc}{\mathbf{s}\cdot\mathbf{s}}\right)^2
\end{aligned}$$

With the surface normals and curvature now computed, we can use the distribution G on the sphere for our desired tasks of **recognition** and **alignment**!

Related, let us look at the **extrema** of curvature for our ellipsoid. These are given as points along the axes of the ellipsoids:

1. $\left(\frac{bc}{a}\right)^2$
2. $\left(\frac{ac}{b}\right)^2$
3. $\left(\frac{ab}{c}\right)^2$

For these extrema, there will always be one maxima, one minima, and one saddle point. The mirror images of these extrema (which account for the mirrors of each of these three extrema above) exhibit identical behavior and reside, geometrically, on the other side of the sphere we map points to.

How Well Does this Ellipsoid Satisfy Our Desired Representation Properties?

- **Translation invariance**: As we saw with EGI last time, this representation is robust to translation invariance.
- **Rotation "Equivariance"**: This property works as desired as well - rotations of the ellipsoid correspond to **equivariant rotations** of the EGI of this ellipse.

22.3 EGI with Solids of Revolution

We saw above that the EGI for an ellipsoid is more mathematically-involved, and is, on the other end, "too simple" in the case of a sphere. Are there geometric shapes that lend themselves well for an "intermediate representation"? It turns out there are, and these are the **solids of revolution**. These include:

- Cylinders
- Spheres
- Cones
- Hyperboloids of one and two sheets

How do we compute the EGI of solids of revolution? We can use **generators** that produce these objects to help.

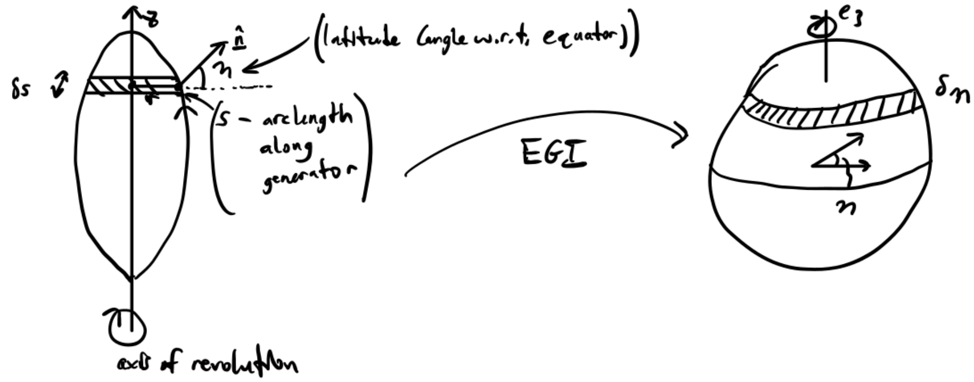


Figure 84: EGI representation of a generalized solid of revolution. Note that bands in the object domain correspond to bands in the sphere domain.

As we can see from the figure, the bands "map" into each other! These solids of revolution are symmetric in both the object and transform space. Let's look at constructing infinitesimal areas so we can then compute Gaussian curvature κ and density G:

- **Area of object band**: $\delta O = 2\pi r \delta s$
- **Area of sphere band**: $\delta S = 2\pi \cos(\eta)\delta n$

Then we can compute the curvature as:

$$\begin{aligned}
\kappa &= \frac{\delta S}{\delta O} = \frac{2\pi\cos(\eta)\delta\eta}{2\pi r\delta s} = \frac{\cos(\eta)\delta\eta}{r\delta s} \\
G &= \frac{1}{\kappa} = \frac{\delta O}{\delta S} = r\sec(\eta)\frac{\delta s}{\delta\eta}
\end{aligned}$$

Then in the limit of $\delta \to 0$, our curvature and density become:

$$\begin{aligned}
&\kappa = \lim_{\delta\to O}\frac{\delta S}{\delta O} = \frac{\cos\eta}{r}\frac{d\eta}{ds} \text{ (Where } \frac{d\eta}{ds} \text{ is the rate of change of surface normal direction along the arc, i.e. curvature)} \\
&G = \lim_{\delta\to 0}\frac{\delta O}{\delta S} = r\sec\eta\frac{ds}{d\eta} \text{ (Where } \frac{ds}{d\eta} \text{ is the rate of change of the arc length w.r.t. angle)} \\
&\kappa = \frac{\cos\eta}{r}\kappa
\end{aligned}$$

22.4 Gaussian Curvature

Using the figure below, we can read off the trigonometric terms from this diagram to compute the curvature κ in a different way:

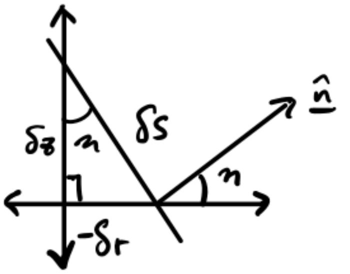


Figure 85: Our infinitesimal trigonometric setup that we can use to calculate Gaussian curvature.

We can read off the trigonometric terms as:

$$\begin{aligned}
\sin\eta &= \frac{-\delta r}{\delta s} = -r_s \text{ (First derivative)} \\
\cos\eta\frac{d\eta}{ds} &= \frac{d}{ds}(-r_s) = -r_{\text{ss}} \text{ (Second derivative)}
\end{aligned}$$

Then the curvature for the solid of revolution is given by:

$$\kappa = \frac{-r_{\text{ss}}}{r}$$

Let's look at some examples to see how this framework applies:

22.4.1 Example: Sphere

Take the following cross-section of a sphere in 2D:

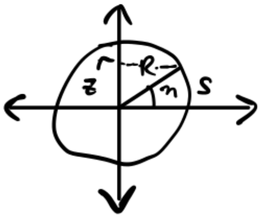


Figure 86: Cross-section of a 3D sphere in 2D. We can apply our framework above to derive our same result for Gaussian curvature.

Mathematically:

$$\begin{aligned}
r &= R\cos\left(\frac{S}{R}\right) \\
r_{\text{ss}} &= R\left(-\frac{1}{R^2}\right)\cos\left(\frac{S}{R}\right) = -\frac{1}{R}\cos\left(\frac{S}{R}\right)
\end{aligned}$$

Then curvature is given by:

$$\kappa = \frac{-r_{\text{ss}}}{r} = \frac{-\left(-\frac{1}{R}\right)\cos\left(\frac{S}{R}\right)}{R\cos\left(\frac{S}{R}\right)} = \frac{1}{R^2}$$

Alternatively, rather than expressing this in terms of R and S, we can also express as a function of z (with respect to the diagram above). Let's look at some of the quantities we need for this approach:

$$\begin{aligned}
&\tan\eta = -\frac{dr}{dz} = -r_z \text{ (First derivative)} \\
&\sec^2\frac{d\eta}{ds} = \frac{d}{ds}(-r_z) = -r_{zz}\frac{dz}{ds} \text{ (By Chain Rule)} \\
&\sec^2\eta = 1 + \tan^2\eta \\
&\cos\eta = \frac{dz}{d\eta} = -z_s
\end{aligned}$$

Putting all of these equations together:

$$\begin{aligned}
&\kappa = \frac{-r_{zz}}{r(1+z^2)^2} \\
&r = \sqrt{R^2 - Z^2} \\
&r_z = -\frac{Z}{\sqrt{R^2 - Z^2}} \\
&r_{zz} = -\frac{R^2}{(R^2 - Z^2)^{\frac{3}{2}}} \\
&1 + r_z^2 = \frac{R^2}{R^2 - Z^2} \\
&\text{Then: } \kappa = \frac{-r_{zz}}{r(1+z^2)^2} = \frac{1}{R^2}
\end{aligned}$$

One particular EGI, and the applications associated with it, is of strong interest - the **torus**.

22.4.2 EGI Example Torus

Geometrically, the torus can be visualized as a cross-section in the diagram below, with small radius ρ and large radius R.

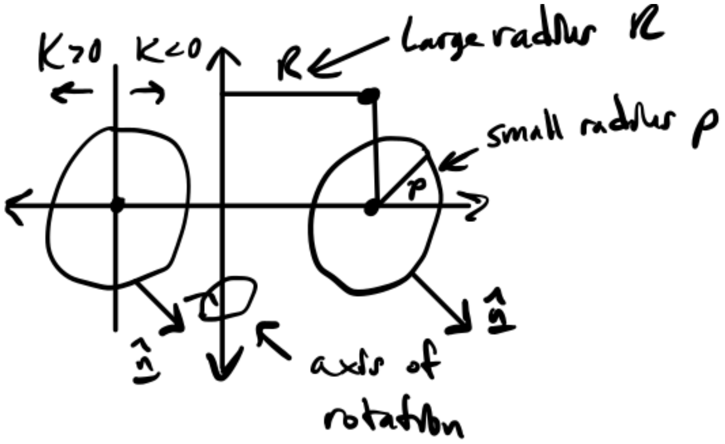


Figure 87: Geometric cross-section of the torus, along with its associated parameters and their significance that describe the torus.

What problems might we have with this object? It is **non-convex**. This Gaussian image may not be invertible when objects are non-convex. We lose uniqueness of mapping, as well as some other EGI properties, by using this non-convex object.

For the torus, the mapping from the object to the sphere is **not invertible** - each point in the EGI maps to two points on the object, which means the **surface normal** of the object is not unique.

The torus is convex into the board. Convex shapes/objects have non-negative curvature, and above we have a saddle point with negative curvature.

Keeping these potential issues in mind, let's press on. Mathematically:

$$\begin{aligned} r &= R + \rho \cos \eta \\ &= R + \rho \cos\left(\frac{S}{\rho}\right) \end{aligned}$$

From here, we can take the second derivative r_{ss}:

$$r_{\text{ss}} = \frac{d^2 r}{ds^2} = -\frac{1}{\rho}\cos\left(\frac{s}{\rho}\right)$$

Combining/substituting to solve for curvature κ:

$$\kappa = \frac{-r_{\text{ss}}}{r} = \frac{1}{\rho}\frac{\cos\left(\frac{S}{\rho}\right)}{R + \rho\cos\left(\frac{S}{\rho}\right)}$$

We will also repeat this calculation for the corresponding surface normal on the other side. We will see that this results in different sign and magnitude:

$$\begin{aligned} r &= R - \rho \cos \eta \\ &= R - \rho \cos\left(\frac{S}{\rho}\right) \end{aligned}$$

From here, we can take the second derivative r_{ss}:

$$r_{\text{ss}} = \frac{d^2 r}{ds^2} = \frac{1}{\rho}\cos\left(\frac{s}{\rho}\right)$$

Combining/substituting to solve for curvature κ:

$$\kappa = \frac{-r_{\text{ss}}}{r} = -\frac{1}{\rho}\frac{\cos\left(\frac{S}{\rho}\right)}{R + \rho\cos\left(\frac{S}{\rho}\right)}$$

Since we have two different Gaussian curvatures, and therefore two separate densities for the same surface normal, what do we do? Let's discuss two potential approaches below.

1. **Approach 1: Adding Densities**: For any non-convex object in which we have multiple points with the same **surface normals**, we can simply add these points together in density:

$$G = \frac{1}{\kappa_+} + \frac{1}{\kappa_-} = 2\rho^2$$

   Even though this approach is able to cancel many terms, it creates a constant density, which means we will not be able to solve alignment/orientation problems. Let's try a different approach.

2. **Approach 2: Subtracting Densities**: In this case, let us try better taking into account local curvature by subtracting instead of adding densities, which will produce a non-constant combined density:

$$G = \frac{1}{\kappa_+} - \frac{1}{\kappa_-} = 2R\rho \sec(\eta)$$

   This result is non-constant (better for solving alignment/orientation problems), but we should note that because of the secant term, it has a singularity at the pole. We can conceptualize this singularity intuitively by embedding a sphere within an infinite cylinder, and noting that as we approach the singularity, the mapped location climbs higher and higher on the cylinder.

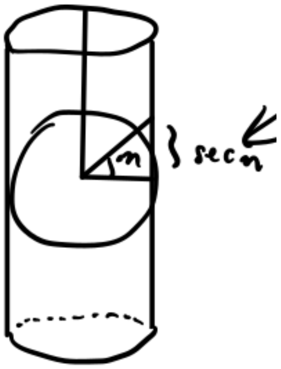


Figure 88: Embedding our sphere within a cylinder geometrically illustrates the singularity we observe for $\sec\eta$ as $\eta \to \frac{\pi}{2}$.

Note: Our **alignment** and **recognition** tasks are done in sphere space, and because of this, we do not need to reconstruct the original objects after applying the EGI mapping.

22.4.3 Analyzing the Density Distribution of the Sphere (For Torus)

Whereas we had a direct mapping from "bands" to "bands" in the previous case, in this case our "bands" map to "crescents" on the sphere when computing the EGI of the torus.

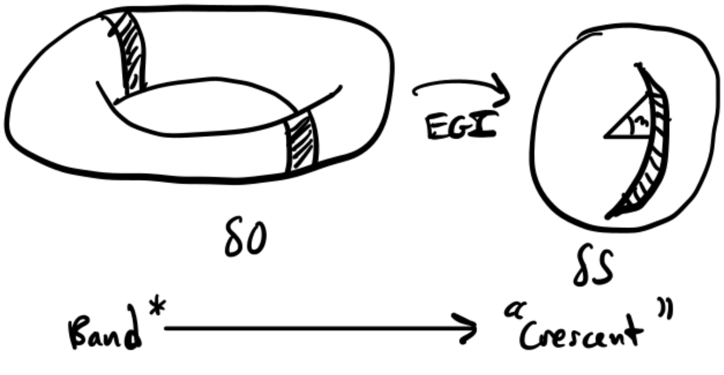


Figure 89: Computing the EGI of a torus, and demonstrating how we do not have the same "band to band" correspondence/mapping that we had in the previous case, suggesting that this mapping may not be invertible.

Note: On a torus, the band is slightly narrower on the inner side. Can balance out this asymmetry by using the band on the opposite side.

We can also consider the area of a torus:

$$\begin{aligned} A_{\text{torus}} &= (2\pi\rho)(2\pi R) \\ &= 4\pi^2 R\rho \\ &= \text{“A circle of circles”} \end{aligned}$$

As a result of how this area is structured, two “donuts” (tori) of different shape/structure but with the same area correspond/map to the same EGI, since EGI captures a ratio of areas. This loss of uniqueness is due to the fact that this torus representation still remains non-convex - in a sense this is the “price we pay” for using a non-convex representation.

Some other issues to be mindful of with a torus:

- It is hard to extract all surface normals since it is impossible to image all surfaces of a torus with just a single camera/camera pose. This phenomenon is also due to the non-convexity of the torus.

22.5 How Can We Compute EGI Numerically?

How can we compute EGI numerically through applications/frameworks such as **photometric stereo** data or with discretized **meshes**? We can look at the facets/pixels of objects.

Let us first suppose we have a triangular facet:

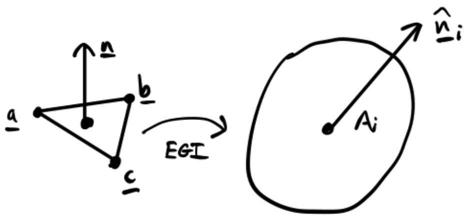


Figure 90: A triangular facet, which we can imagine we use to compute our EGI numerically.

As we have done before, we can find the **surface normal** and **area** of this facet to proceed with our EGI computation:

- **Surface Normal**: This can be computed simply by taking the cross product between any of the two edges of the triangular facet, e.g.

$$\begin{aligned} \mathbf{n} &= (\mathbf{b}-\mathbf{a}) \times (\mathbf{c}-\mathbf{b}) \\ &= (\mathbf{a}\times\mathbf{b}) + (\mathbf{b}\times\mathbf{c}) + (\mathbf{c}\times\mathbf{a}) \end{aligned}$$

- **Area**: This can be computed by recalling the area of a triangular ($\frac{1}{2}$base × height):

$$\begin{aligned} A &= \frac{1}{2}(\mathbf{b}-\mathbf{a})\cdot(\mathbf{c}-\mathbf{a}) \\ &= \frac{1}{6}((\mathbf{a}\cdot\mathbf{a}+\mathbf{b}\cdot\mathbf{b}+\mathbf{c}\cdot\mathbf{c}) - (\mathbf{a}\cdot\mathbf{b}+\mathbf{b}\cdot\mathbf{c}+\mathbf{c}\cdot\mathbf{a})) \end{aligned}$$

We can repeat this area and surface normal computation on all of our facets/elements to compute the mass distribution over the sphere by adding different facets from all over surface. We add rather than subtract components to ensure we have a non-constant EGI, and therefore have an EGI representation that is suitable for alignment tasks.

22.5.1 Direction Histograms

Direction histograms are another way of thinking about how we build an EGI. These are used in different applications outside of machine vision as well, e.g. finding direction histograms in (i) muscle fibers, (ii) neural images, (iii) analyzing blood vessels.

Typically, these **direction histograms** can be used to ascertain what the most common orientations are amongst a set of orientations.

For histograms in higher dimensions, is subdividing the region into squares/cubes the most effective approach? As it turns out, no, and this is because the regions are not round. We would prefer to fill in the plane with disks.

To contrast, suppose the tessellation is with triangles - you are now combining things that are pretty far away, compared to a square. Hexagons alleviate this problem by minimizing distance from the center of each cell to the vertives, while also preventing overlapping/not filled in regions. **Other notes**:

- We also need to take "randomness" into account, i.e. when points lie very close to the boundaries between different bins. We can account for this phenomena by constructing and counting not only the original grids, but additionally, shifted/offset grids that adjust the intervals, and consequently the counts in each grid cell.

  The only issue with this approach is that this "solution" scales poorly with dimensionality: As we increase dimensions, we need to take more grids (e.g. for 2D, we need our (i) Original plane, (ii) Shifted x, (iii) Shifted y, and (iv) Shifted x and shifted y). However, in light of this, this is a common solution approach for mitigating "randomness" in 2D binning.

22.5.2 Desired Properties of Dividing Up the Sphere/Tessellations

To build an optimal representation, we need to understand what our desired optimal properties of our tessellations will be:

1. **Equal areas of cells/facets**
2. **Equal shapes of cells/facets**
3. **"Rounded" shapes of cells/facets**
4. **Regular pattern**
5. **Allows for easy "binning"**: As an example a brute force way to do binning for spheres is to compute the dotr product with all surface normals and take the largest of these. This is very impractical and computationally-inefficient because it requires us to step through all data each time. For this reason, it is important we consider implementations/representations that make binning efficient and straightforward.
6. **Alignment of rotation**: The idea with this is that we need to bring one object into alignment with the other in order to do effective recognition, one of the key tasks we cited.

   For instance, with a dodecahedron, our **orientation/direction histogram** is represented by 12 numbers (which correspond to the number of faces on the dodecahedron):

   $$[A_1, A_2, \cdots, A_{12}]$$

   When we bring this object into alignment during, for instance, a recognition task, we merely only need to permute the order of these 12 numbers - all information is preserved and there is no loss from situations such as overlapping. This is an advantage of having alignment of rotation.

   $$[A_7, A_4, \cdots, A_8]$$

Platonic and **Archimedean** solids are the representations we will use for these direction histograms.

23 Quiz 1 Review

Here you will find a review of some of the topics covered so far in Machine Vision. These are as follows in the section notes:

1. **Mathematics review** - Unconstrained optimization, Green's Theorem, Bezout's Theorem, Nyquist Sampling Theorem
2. **Projection** - Perspective vs. Orthographic, Object vs. Image space
3. **Optical Flow** - BCCE/optical flow, Time to Contact, vanishing points
4. **Photometry** - Radiance, irradiance, illumination, geometry, BRDF, Helmholtz Reciprocity
5. **Different surface types** - Lambertian, Hapke
6. **Shape from Shading (SfS)** - Hapke case, general case, reflectance maps, image irradiance equation

7. **Photometric Stereo** - least squares, multiple measurement variant, multiple light sources variant

8. **Computational molecules** - Sobel, Robert, Silver operators, finite differences (forward, backward, and average), Laplacians

9. **Lenses** - Thin lenses, thick lenses, telecentric lens, focal length, principal planes, pinhole model

10. **Patent Review** - Edge detector, Object Detection

23.1 Quick Mathematics Review

Let's quickly touch on some important mathematical theorems that are useful for machine vision.

- **Unconstrained optimization**: Suppose we are trying to optimize an objective $J(a, b, c)$ by some set of parameters $\{a, b, c\}$. We can find the best set of parameters $\{a^*, b^*, c^*\}$ using first-order conditions:

$$
\begin{aligned}
&a^*, b^*, c^* = \arg\min_{a,b,c} J(a, b, c) \\
&\rightarrow \frac{\partial J}{\partial a} = 0 \\
&\rightarrow \frac{\partial J}{\partial b} = 0 \\
&\rightarrow \frac{\partial J}{\partial c} = 0
\end{aligned}
$$

  The solution a^*, b^*, c^* is the set of a, b, c that satisfies these first-order conditions. Note that this procedure is identical with maximization.

- **Green's Theorem** relates the line integral of a contour to the area located within that contour:

$$\oint_L (Ldx + Mdy) = \iint_D \Big(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\Big) dxdy$$

- **Bézout's Theorem**: *The maximum number of solutions is the product of the polynomial order of each equation in the system of equations*:

$$\text{number of solutions} = \prod_{e=1}^{E} o_e$$

- **Nyquist Sampling Theorem**: We must sample at twice the frequency of the highest-varying component of our image to avoid aliasing and consequently reducing spatial artifacts.

- **Taylor Series**: We can expand any analytical, continuous, infinitely-differentiable function into its Taylor Series form according to:

$$f(x + \delta x) = f(x) + \delta x f'(x) + \frac{(\delta x)^2}{2!} f''(x) + \frac{(\delta x)^3}{3!} f'''(x) + \frac{(\delta x)^4}{24} f^{(4)}(x) + ... = \sum_{i=0}^{\infty} \frac{(\delta x)^i f^{(i)}(x)}{i!}, \quad \text{where } 0! \triangleq 1$$

  For a single variable function, and:

$$f(x + \delta_x, y + \delta_y) = f(x, y) + \delta_x \frac{\partial f(x, y)}{\partial x} + \delta_y \frac{\partial f(x, y)}{\partial y} + \cdots$$

  For a multivariable function.

23.2 Projection: Perspective and Orthographic

A few important notes to remember:

- We use CAPITAL LETTERS for WORLD SPACE, and lowercase letters for image space.

- Orthographic projection is essentially the limit of perspective projection as we move the point we are mapping to the image plane far away from the optical axis.

- We can derive perspective projection from the pinhole model and similar triangles.

Perspective Projection:

$$\frac{x}{f} = \frac{X}{Z}, \frac{y}{f} = \frac{Y}{Z} \textbf{ (component form)}$$

$$\frac{1}{f}\mathbf{r} = \frac{1}{\mathbf{R} \cdot \hat{\mathbf{z}}}\mathbf{R} \textbf{ (vector form)}$$

Orthographic Projection:

$$x = \frac{f}{Z_0}X, y = \frac{f}{Z_0}y$$

23.3 Optical Flow

Optical flow captures the motion of an image over time in image space. Therefore, we will be interested in how brightness in the image changes with respect to (i) time, (ii) in the x-direction, and (iii) in the y-direction. We will use derivatives to describe these different quantities:

- $u \triangleq \frac{dx}{dt}$ - i.e. the velocity in the image space in the x direction.
- $v \triangleq \frac{dx}{dt}$ - i.e. the velocity in the image space in the y direction.
- $\frac{\partial E}{\partial x}$ - i.e. how the brightness changes in the x direction.
- $\frac{\partial E}{\partial y}$ - i.e. how the brightness changes in the y direction.
- $\frac{\partial E}{\partial t}$ - i.e. how the brightness changes w.r.t. time.

If $x(t)$ and $y(t)$ both depend on t, then the chain rule gives us the following for the total derivative:

$$\frac{dE(x,y,t)}{dt} = \frac{dx}{dt}\frac{\partial E}{\partial x} + \frac{dy}{dt}\frac{\partial E}{\partial y} + \frac{\partial E}{\partial t} = 0$$

Rewriting this in terms of u, v from above:

$$uE_x + vE_y + E_t = 0$$

This equation above is known as the **Brightness Change Constraint Equation (BCCE)**. This is also one of the most important equations in 2D optical flow.

Normalizing the equation on the right by the magnitude of the brightness derivative vectors, we can derive the **brightness gradient**:

$$(u,v) \cdot \left(\frac{E_x}{\sqrt{E_x^2 + E_y^2}}, \frac{E_y}{\sqrt{E_x^2 + E_y^2}}\right) = (u,v) \cdot \hat{\mathbf{G}} = -\frac{E_t}{\sqrt{E_x^2 + E_y^2}}$$

$$\textbf{Brightness Gradient}: \hat{\mathbf{G}} = \left(\frac{E_x}{\sqrt{E_x^2 + E_y^2}}, \frac{E_y}{\sqrt{E_x^2 + E_y^2}}\right)$$

What is the **brightness gradient**?

- A unit vector given by: $\left(\frac{E_x}{\sqrt{E_x^2+E_y^2}}, \frac{E_y}{\sqrt{E_x^2+E_y^2}}\right) \in \mathbb{R}^2$.
- Measures spatial changes in brightness in the image in the image plane x and y directions.

We are also interested in contours of constant brightness, or **isophotes**. These are curves on an illuminated surface that connects points of equal brightness (source: Wikipedia).

Finally, we are also interested in solving for optimal values of u and v for multiple measurements. In the ideal case without noise:

$$\begin{bmatrix} U \\ V \end{bmatrix} = \frac{1}{(E_{x_1}E_{y_2} - E_{y_1}E_{x_2})} \begin{bmatrix} E_{y_2} & -E_{y_1} \\ -E_{x_2} & E_{x_1} \end{bmatrix} \begin{bmatrix} -E_{t_1} \\ -E_{t_2} \end{bmatrix}$$

When there is noise we simply minimize our objective, instead of setting it equal to zero:

$$J(u,v) \triangleq \int_{x \in \mathbb{X}} \int_{y \in \mathbb{Y}} (uE_x + vE_y + E_t)^2 dxdy$$

We solve for our set of optimal parameters by finding the set of parameters that minimizes this objective:

$$u^*, v^* = \arg\min_{u,v} J(u,v) = \arg\min_{u,v} \int_{x \in \mathbb{X}} \int_{y \in \mathbb{Y}} (uE_x + vE_y + E_t)^2 dxdy$$

Since this is an unconstrained optimization problem, we can solve by finding the minimum of the two variables using two First-Order Conditions (FOCs):

- $\frac{\partial J(u,v)}{\partial u} = 0$
- $\frac{\partial J(u,v)}{\partial v} = 0$

Vanishing points: These are the points in the image plane (or extended out from the image plane) that parallel lines in the world converge to. Applications include:

- Multilateration
- Calibration Objects (Sphere, Cube)
- Camera Calibration

We will also look at **Time to Contact (TTC)**:

$$\frac{Z}{W} \triangleq= \frac{Z}{\frac{dZ}{dt}} = \frac{\text{meters}}{\frac{\text{meters}}{\text{seconds}}} = \text{seconds}$$

Let us express the inverse of this **Time to Contact (TTC)** quantity as C, which can be interpreted roughly as the number of frames until contact is made:

$$C \triangleq \frac{W}{Z} = \frac{1}{\text{TTC}}$$

23.4 Photometry

Here, we will mostly focus on some of the definitions we have encountered from lecture:

- **Photometry**: Photometry is the science of measuring visible radiation, light, in units that are weighted according to the sensitivity of the human eye. It is a quantitative science based on a statistical model of the human visual perception of light (eye sensitivity curve) under carefully controlled conditions.
- **Radiometry**: Radiometry is the science of measuring radiation energy in any portion of the electromagnetic spectrum. In practice, the term is usually limited to the measurement of ultraviolet (UV), visible (VIS), and infrared (IR) radiation using optical instruments.
- **Irradiance**: $E \triangleq \frac{\delta P}{\delta A}$ (W/m^2). This corresponds to light falling on a surface. When imaging an object, irradiance is converted to a grey level.
- **Intensity**: $I \triangleq \frac{\delta P}{\delta W}$ (W/ster). This quantity applied to a point source and is often directionally-dependent.
- **Radiance**: $L \triangleq \frac{\delta^2 P}{\delta A \delta \Omega}$ (W/m$^2\times$ ster). This photometric quantity is a measure of how bright a surface appears in an image.

- **BRDF (Bidirectional Reflectance Distribution)**: $f(\theta_i, \theta_e, \phi_i, \phi_e) = \frac{\delta L(\theta_e, \phi_e)}{\delta E(\theta_i, \phi_i)}$. This function captures the fact that oftentimes, we are only interested in light hitting the camera, as opposed to the total amount of light emitted from an object. Last time, we had the following equation to relate image irradiance with object/surface radiance:

  $$E = \frac{\pi}{4} L \Big(\frac{d}{f}\Big)^2 \cos^4 \alpha$$

  Where the irradiance of the image E is on the lefthand side and the radiance of the object/scene L is on the right. The BRDF must also satisfy **Helmholtz reciprocity**, otherwise we would be violating the 2nd Law of Thermodynamics. Mathematically, recall that Helmholtz reciprocity is given by:

  $$f(\theta_i, \theta_e, \phi_i, \phi_e) = f(\theta_e, \theta_i, \phi_e, \phi_i) \; \forall \; \theta_i, \theta_e, \phi_i, \phi_e$$

23.5 Different Surface Types

With many of our photometric properties established, we can also discuss photometric properties of different types of surfaces. Before we dive in, it is important to also recall the definition of surface orientation:

- $p \triangleq \frac{\partial z}{\partial x}$
- $q \triangleq \frac{\partial z}{\partial y}$
- **Lambertian Surfaces**:
  - Ideal Lambertian surfaces are equally bright from all directions, i.e.

    $$\begin{gathered} f(\theta_i, \theta_e, \phi_i, \phi_e) = f(\theta_e, \theta_i, \phi_e, \phi_i) \; \forall \; \theta_i, \theta_e, \phi_i, \phi_e \\ \text{AND} \\ f(\theta_i, \theta_e, \phi_i, \phi_e) = K \in \mathbb{R} \text{ with respect to } \theta_e, \phi_e \end{gathered}$$

  - **"Lambert's Law"**:

    $$E_i \propto \cos \theta_i = \hat{\mathbf{n}} \cdot \hat{\mathbf{s}}_i$$

- **Hapke Surfaces**:
  - The BRDF of a Hapke surface is given by:

    $$f(\theta_i, \phi_i, \theta_e, \phi_e) = \frac{1}{\sqrt{\cos \theta_e \cos \theta_i}}$$

  - Isophotes of Hapke surfaces are **linear** in spatial gradient, or (p, q) space.
- What do the isophotes of the moon look like, supposing the moon can fall under some of the different types of surfaces we have discussed?
  - **Lambertian**: We will see circles and ellipses of isophotes, depending on the angle made between the viewer and the the moon.
  - **Hapke**: Because of the BRDF behavior, isophotes will run **longitudinally** along the moon in the case in which it is a Hapke surface.

23.6 Shape from Shading (SfS)

This is one of the most fundamental problems in machine vision in which we try to estimate the surface of an object from brightness measurements. Several variations of this problem we consider:

- Hapke surfaces - this leads to linear isophotes in (p, q) space, and allows us to solve for one of the dimensions of interest.
- General case: There are several techniques we can apply to solve for this:
  - Green's Theorem (see lecture notes handout)
  - Iterative Discrete Optimization (see lecture notes handout)

For these problems, we considered:

- Characteristic strips (x, y, z, p, q)
- Initial curves/base characteristics
- Normalizing with respect to constant step sizes
- A system of 5 ODEs
- Stationary points and estimates of surfaces around them for initial points

Next, let us review reflectance maps. A **reflectance map** $R(p, q)$ is a lookup table (or, for simpler cases, a parametric function) that stores the brightness for particular surface orientations $p = \frac{\partial z}{\partial x}, q = \frac{\partial z}{\partial y}$.

The **Image Irradiance Equation** relates the reflectance map to the brightness function in the image $E(x, y)$ and is the first step in many Shape from Shading approaches.

$$E(x, y) = R(p, q)$$

23.7 Photometric Stereo

Photometric stereo is a technique in machine vision for estimating the surface normals of objects by observing that object under different lighting conditions. This problem is quite common in machine vision applications.

One way we can solve photometric stereo is by taking multiple brightness measurements from a light source that we move around. This problem becomes:

$$\begin{bmatrix} -s_1^T \\ -s_2^T \\ -s_3^T \end{bmatrix} \mathbf{n} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}$$

Written compactly:

$$\mathbf{Sn} = \mathbf{E} \longrightarrow \mathbf{n} = \mathbf{S}^{-1}\mathbf{E}$$

Note that we we need **S** to be invertible to compute this, which occurs when the light source vectors are not coplanar.

Other variations of this problem:

- Using light sources at different electromagnetic frequencies
- Using a camera with different light filters
- Moving the object to different locations

23.8 Computational Molecules

These are crucial components for applying machine concepts to real-life problems.

1. $E_x = \frac{1}{\epsilon}(E(x, y) - E(x-1, y))$ (Backward Difference), $\frac{1}{\epsilon}(z(x+1, y) - z(x, y))$ (Forward Difference)
2. $E_y = \frac{1}{\epsilon}(E(x, y) - E(x, y-1))$ (Backward Difference), $\frac{1}{\epsilon}(E(x, y+1) - E(x, y))$ (Forward Difference)
3. $\Delta E = \nabla^2 E = \frac{1}{\epsilon^2}(4E(x, y) - (E(x-1, y) + E(x+1, y) + E(x, y-1) + E(x, y+1)))$
4. $E_{xx} = \frac{\partial^2 E}{\partial x^2} = \frac{1}{\epsilon^2}(E(x-1, y) - 2(x, y) + E(x+1, y))$
5. $E_{yy} = \frac{\partial^2 z}{\partial y^2} = \frac{1}{\epsilon^2}(E(x, y-1) - 2(x, y) + E(x, y+1))$
6. **Robert's Cross**: This approximates derivatives in a coordinate system rotated 45 degrees (x', y'). The derivatives can be approximated using the $K_{x'}$ and $K_{y'}$ kernels:

$$\frac{\partial E}{\partial x'} \rightarrow K_{x'} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$
$$\frac{\partial E}{\partial y'} \rightarrow K_{y'} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

7. **Sobel Operator**: This computational molecule requires more computation and it is not as high-resolution. It is also more robust to noise than the computational molecules used above:

$$\frac{\partial E}{\partial x} \rightarrow K_x = \begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial E}{\partial y} \rightarrow K_y = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

8. **Silver Operators**: This computational molecule is designed for a hexagonal grid. Though these filters have some advantages, unfortunately, they are not compatible with most cameras as very few cameras have a hexagonal pixel structure.

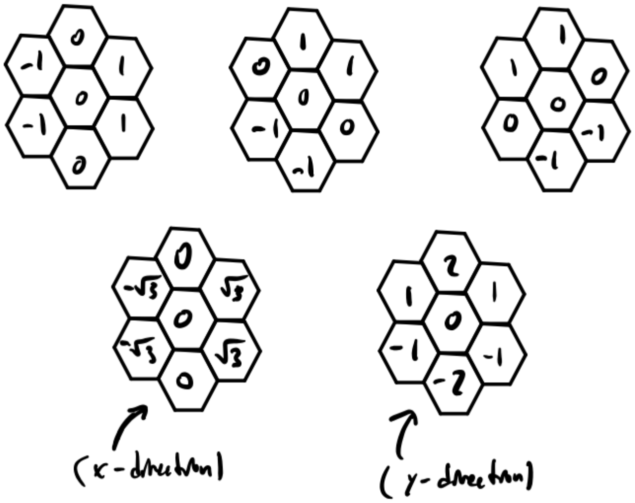


Figure 91: Silver Operators with a hexagonal grid.

9. **"Direct Edge" Laplacian**: $\frac{1}{\epsilon^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

10. **"Indirect Edge" Laplacian**: $\frac{1}{2\epsilon^2}\begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix}$

11. **Rotationally-symmetric Laplacian**:

$$4\Big(\frac{1}{\epsilon^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}\Big) + 1\Big(\frac{1}{2\epsilon^2}\begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix}\Big) = \frac{1}{6\epsilon^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

Some methods we used for analyzing how well these work:

1. **Taylor Series**: From previous lectures we saw that we could use averaging to reduce the error terms from 2nd order derivatives to third order derivatives. This is useful for analytically determining the error.

2. **Test functions**: We will touch more on these later, but these are helpful for testing your derivative estimates using analytical expressions, such as polynomial functions.

3. **Fourier domain**: This type of analysis is helpful for understanding how these "stencils"/molecules affect higher (spatial) frequency image content.

23.9 Lenses

Lenses are also important, because they determine our ability to sense light and perform important machine vision applications. Some types of lenses:

- **Thin lenses** are the first type of lens we consider. These are often made from glass spheres, and obey the following three rules:
  - Central rays (rays that pass through the center of the lens) are undeflected - this allows us to preserve perspective projection as we had for pinhole cameras.
  - The ray from the focal center emerges parallel to the optical axis.
  - Any parallel rays go through the focal center.
- **Thick lenses** (cascaded thin lenses)
- **Telecentric lenses** - These "move" the the Center of Projection to infinity to achieve approximately orthographic projection.
- Potential distortions caused by lenses:
  - **Radial distortion**: In order to bring the entire angle into an image (e.g. for wide-angle lenses), we have the "squash" the edges of the solid angle, thus leading to distortion that is radially-dependent. Typically, other lens defects are mitigated at the cost of increased radial distortion. Some specific kinds of radial distortion [5]:
    - Barrel distortion
    - Mustache distortion
    - Pincushion distortion
  - **Lens Defects**: These occur frequently when manufacturing lenses, and can originate from a multitude of different issues.

23.10 Patent Review

The two patents we touched on were for:

- **Fast and Accurate Edge detection**: At a high level, this framework leveraged the following steps for fast edge detection:
  - CORDIC algorithm for estimating (and subsequently quantizing) the brightness gradient direction
  - Thresholding gradients via Non-Maximum Suppression (NMS)
  - Interpolation at the sub-pixel level along with peak finding for accurate edge detection at sub-pixel level
  - Bias compensation
  - Plane position

  We also discussed some issues/considerations with the following aspects of this framework's design:
  - Quantization of gradient direction
  - Interpolation method for sub-pixel accuracy (quadratic, piecewise linear, cubic, etc.)
- **Object detection and pose estimation**: At a high level, this framework leveraged the following steps for fast object detection/pose estimation:
  - Finding specific **probing points** in a template image that we use for comparing/estimating gradients.
  - Using different **scoring functions** to compare template image to different configurations of the runtime image to find the object along with its pose (position with orientation).

Here you will find a high-level review of some of the topics covered since Quiz 1. These are as follows in the section notes:

1. **Relevant Mathematics Review** - Rayleigh Quotients, Groups, Levenberg-Marquadt, Bezout's Theorem
2. **Systems/Patents** - PatQuick, PatMAx, Fast Convolutions, Hough Transforms
3. **Signals** - Sampling, Spline Interpolation, Integral Images, Repeated Block Averaging
4. **Photogrammetry** - Photogrammetry Problems, Absolute orientation, Relative Orientation, Interior Orientation, Exterior Orientation, Critical Surfaces, Radial and Tangential Distortion
5. **Rotations/Orientation** - Gibb's vector, Rodrigues Formula, Quaternions, Orthonormal Matrices
6. **3D Recognition** - Extended Gaussian Image, Solids of Revolution, Polyhedral Objects, Desirable Properties

24 Quiz 2 Review

24.1 Relevant Mathematics Review

We'll start with a review of some of the relevant mathematical tools we have relied on in the second part of the course.

24.1.1 Rayleigh Quotients

We saw from our lectures with absolute orientation that this is a simpler way (relative to Lagrange Multipliers) to take constraints into account:

The intuitive idea behind them: How do I prevent my parameters from becoming too large) positive or negative) or too small (zero)? We can accomplish this by dividing our objective by our parameters, in this case our constraint. In this case, with the Rayleigh Quotient taken into account, our objective becomes:

$$\min_{\overset{o}{q}, \overset{o}{q}\cdot\overset{o}{q}=1} \overset{o}{q}^T N \overset{o}{q} \longrightarrow \min_{\overset{o}{q}} \frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}}$$

24.1.2 (Optional) Groups

Though we don't cover them specifically in this course, **groups** can be helpful for better understanding rotations. "In mathematics, a group is a set equipped with a binary operation that combines any two elements to form a third element in such a way that four conditions called group axioms are satisfied, namely closure, associativity, identity and invertibility." [1]. We won't focus too much on the mathematical details, but being aware of the following groups may be helpful when reading machine vision, robotics, control, or vision papers:

- Orthonormal Rotation Matrices → $\mathbf{SO}(3) \in \mathbb{R}^{3\times 3}$, **Special Orthogonal Group**. Note that the "3" refers to the number of dimensions, which may vary depending on the domain.
- Poses (Translation + Rotation Matrices) → $\mathbf{SE}(3) \in \mathbb{R}^{4\times 4}$, **Special Euclidean Group**.

Since these groups do not span standard Euclidean spaces (it turns out these groups are both manifolds - though this is not generally true with groups), we cannot apply standard calculus-based optimization techniques to solve for them, as we have seen to be the case in our photogrammetry lectures.

24.1.3 (Optional) Levenberg-Marquadt and Gauss-Newton Nonlinear Optimization

Recall this was a nonlinear optimization technique we saw to solve photogrammetry problems such as Bundle Adjustment (BA).

Levenberg-Marquadt (LM) and Gauss-Newton (GN) are two nonlinear optimization procedures used for deriving solutions to nonlinear least squares problems. These two approaches are largely the same, except that LM uses an additional **regularization term** to ensure that a solution exists by making the closed-form matrix to invert in the normal equations positive semidefinite. The normal equations, which derive the closed-form solutions for GN and LM, are given by:

1. **GN**: $(J(\theta)^T J(\theta))^{-1}\theta = J(\theta)^T e(\theta) \implies \theta = (J(\theta)^T J(\theta))^{-1} J(\theta)^T e(\theta)$
2. **LM**: $(J(\theta)^T J(\theta) + \lambda I)^{-1}\theta = J(\theta)^T e(\theta) \implies \theta = (J(\theta)^T J(\theta) + \lambda I)^{-1} J(\theta)^T e(\theta)$

Where:

- θ is the vector of parameters and our solution point to this nonlinear optimization problem.
- $J(\theta)$ is the Jacobian of the nonlinear objective we seek to optimize.
- $e(\theta)$ is the residual function of the objective evaluated with the current set of parameters.

Note the λI, or regularization term, in Levenberg-Marquadt. If you're familiar with ridge regression, LM is effectively ridge regression/regression with L2 regularization for nonlinear optimization problems. Often, these approaches are solved iteratively using gradient descent:

1. **GN**: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha(J(\theta^{(\mathbf{t})})^T J(\theta^{(\mathbf{t})}))^{-1} J(\theta^{(\mathbf{t})})^T e(\theta^{(\mathbf{t})})$
2. **LM**: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha(J(\theta^{(\mathbf{t})})^T J(\theta^{(\mathbf{t})}) + \lambda I)^{-1} J(\theta^{(\mathbf{t})})^T e(\theta^{(\mathbf{t})})$

Where α is the step size, which dictates how quickly the estimates of our approaches update.

24.1.4 Bezout's Theorem

Though you're probably well-versed with this theorem by now, its importance is paramount for understanding the number of solutions we are faced with when we solve our systems:

Theorem: *The maximum number of solutions is the product of the polynomial order of each equation in the system of equations*:

$$\text{number of solutions} = \prod_{e=1}^{E} o_e$$

24.2 Systems

In this section, we'll review some of the systems we covered in this course through patents, namely PatQuick, PatMAx, and Fast Convolutions. A block diagram showing how we can cascade the edge detection systems we studied in this class can be found below:

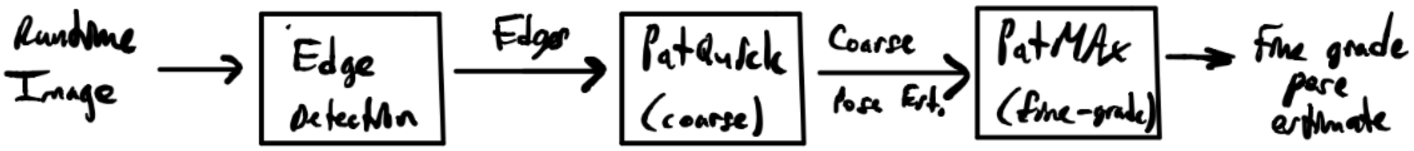


Figure 92: An overview of how the patents we have looked at for object inspection fit together.

24.2.1 PatQuick

There were three main "objects" in this model:

- **Training/template image**. This produces a model consisting of probe points.
- **Model**, containing probe points.
- **Probe points**, which encode evidence for where to make gradient comparisons, i.e. to determine how good matches between the template image and the runtime image under the current pose configuration.

Once we have the model from the training step, we can summarize the process for generating matches as:

1. Loop over/sample from configurations of the pose space (which is determined and parameterized by our degrees of freedom), and modify the runtime image according to the current pose configuration.
2. Using the probe points of the model, compare the gradient direction (or magnitude, depending on the scoring function) to the gradient direction (magnitude) of the runtime image under the current configuration, and score using one of the scoring functions below.
3. Running this for all/all sampled pose configurations from the pose space produces a multidimensional scoring surface. We can find matches by looking for peak values in this surface.

24.2.2 PatMAx

- This framework builds off of the previous PatQuick patent.
- This framework, unlike PatQuick, does not perform quantization of the pose space, which is one key factor in enabling sub-pixel accuracy.
- PatMAx assumes we already have an approximate initial estimate of the pose.
- PatMAx relies on an iterative process for optimizing energy, and each **attraction step** improves the fit of the configuration.
- Another motivation for the name of this patent is based off of electrostatic components, namely dipoles, from Maxwell. As it turns out, however, this analogy works better with mechanical springs than with electrostatic dipoles.
- PatMAx performs an **iterative attraction process** to obtain an estimate of the pose.

- An iterative approach (e.g. gradient descent, Gauss-Newton, Levenberg-Marquadt) is taken because we likely will not have a closed-form solution in the real world. Rather than solving for a closed-form solution, we will run this iterative optimization procedure until we reach convergence.

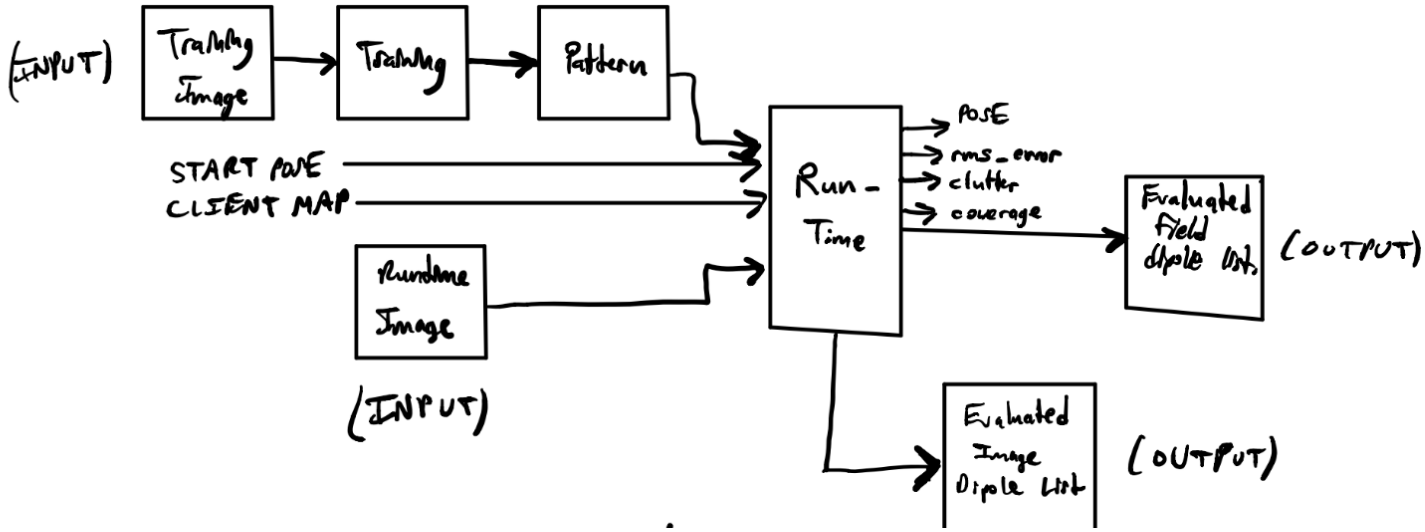


Figure 93: High-level diagram of the PatMAx system.

24.2.3 Fast Convolutions

The patent we explore is **"Efficient Flexible Digital Filtering, US 6.457,032**.

The goal of this system is to efficiently compute filters for multiscale. For this, we assume the form of an N$^{\text{th}}$-order piecewise polynomial, i.e. a N$^{\text{th}}$-order spline.

24.2.4 System Overview

The block diagram of this system can be found below:

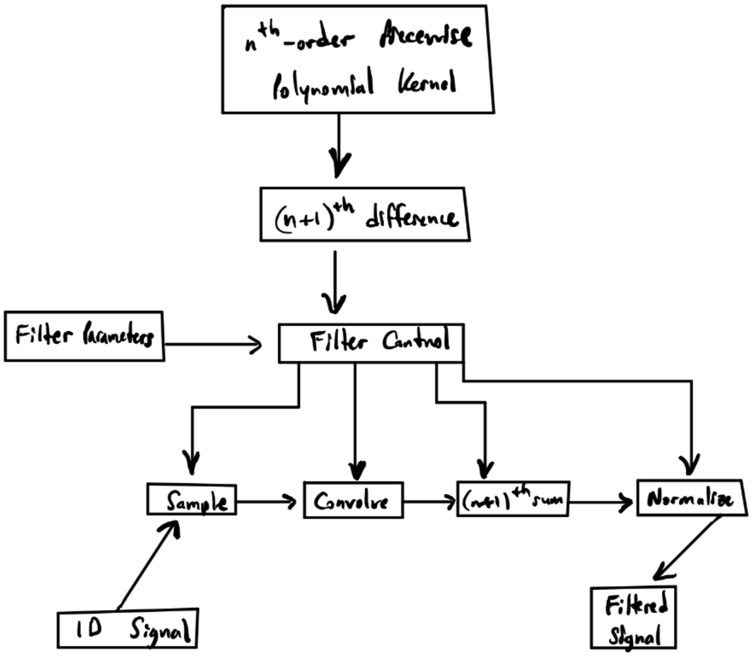


Figure 94: Block diagram of this sparse/fast convolution framework for digital filtering. Note that this can be viewed as a compression problem, in which differencing compresses the signal, and summing decompresses the signal.

A few notes on this system:

- Why is it of interest, if we have N^{th}-order splines as our functions, to take N^{th}-order differences? The reason for this is that the differences create **sparsity**, which is critical for fast and efficient convolution. Sparsity is ensured because:

  $$\frac{d^{N+1}}{dx^{N+1}}f(x) = 0 \; \forall \; x \text{ if } f(x) = \sum_{i=0}^{N} a_i x^i, a_i \in \mathbb{R} \;\; \forall \; i \in \{1, \cdots, N\}$$

  (I.e, if $f(x)$ is a order-N polynomial, then the order-(N+1) difference will be 0 for all x.

  This sparse structure makes convolutions much easier and more efficient to compute by reducing the size/cardinality of the support.

- Why do we apply an order-(N+1) summing operator? We apply this because we need to "invert" the effects of the order-(N+1) difference:

  $$\begin{aligned}
  &\textbf{First Order}: \; \mathcal{D}\mathcal{S} = I \\
  &\textbf{Second Order}: \; \mathcal{D}\mathcal{D}\mathcal{S}\mathcal{S} = \mathcal{D}\mathcal{S}\mathcal{D}\mathcal{S} = (\mathcal{D}\mathcal{S})(\mathcal{D}\mathcal{S}) = II = I \\
  &\vdots \\
  &\textbf{Order K}: \; (\mathcal{D})^K(\mathcal{S})^K = (\mathcal{D}\mathcal{S})^K = I^K = I
  \end{aligned}$$

24.2.5 Hough Transforms

Motivation: Edge and line detection for industrial machine vision; we are looking for lines in images, but our gradient-based methods may not necessarily work, e.g. due to non-contiguous lines that have "bubbles" or other discontinuities. These discontinuities can show up especially for smaller resolution levels.

Idea: The main idea of the **Hough Transform** is to intelligently map from image/surface space to parameter space for that surface.

Some notes on Hough Transforms:

- Often used as a subroutine in many other machine vision algorithms.
- Actually generalize beyond edge and line detection, and extend more generally into any domain in which we map a parameterized surface in image space into parameter space in order to estimate parameters.

Example: Hough Transforms with Lines A line/edge in image space can be expressed (in two-dimensions for now, just for building intuition, but this framework is amenable for broader generalizations into higher-dimensional lines/planes): $y = mx + c$. Note that because $y = mx + c, m = \frac{y-c}{x}$ and $c = y - mx$. Therefore, this necessitates that:

- A line in image space maps to a singular point in Hough parameter space.
- A singular point in line space corresponds to a line in Hough parameter space.

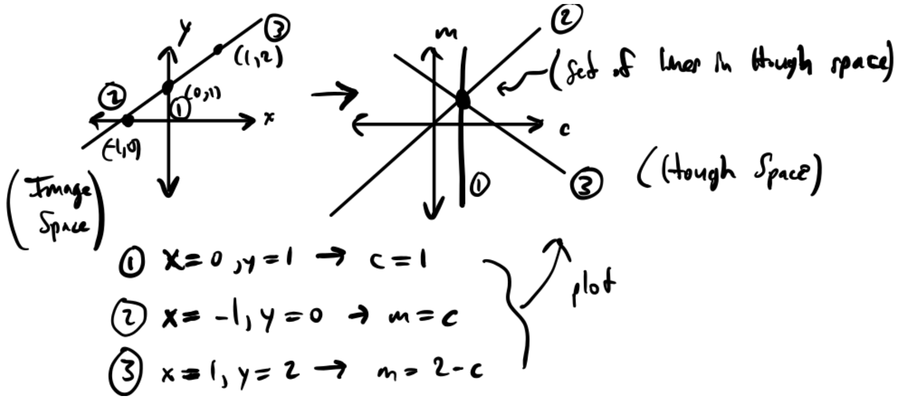


Figure 95: Example of finding parameters in Hough Space via the Hough Transform.

24.3 Photogrammetry

Given the length of the mathematical derivations in these sections, we invite you to revisit notes in lectures 17-21 for a more formal treatment of these topics. In this review, we hope to provide you with strong intuition about different classes of photogrammetric problems and their solutions.

Four important problems in photogrammetry that we covered in this course:

- **Absolute Orientation** $3D \longleftrightarrow 3D$, Range Data
- **Relative Orientation** $2D \longleftrightarrow 2D$, Binocular Stereo
- **Exterior Orientation** $2D \longleftrightarrow 3D$, Passive Navigation
- **Intrinsic Orientation** $3D \longleftrightarrow 2D$, Camera Calibration

Below we discuss each of these problems at a high level. We will be discussing these problems in greater depth later in this and following lectures.

24.3.1 Absolute Orientation

We will start with covering **absolute orientation**. This problem asks about the relationship between two or more objects (cameras, points, other sensors) in 3D. Some examples of this problem include:

1. Given two 3D sensors, such as lidar (light detection and ranging) sensors, our goal is to find the **transformation**, or **pose**, between these two sensors.
2. Given one 3D sensor, such as a lidar sensor, and two objects (note that this could be two distinct objects at a single point in time, or a single object at two distinct points in time), our goal is to find the **transformation**, or **pose**, between the two objects.

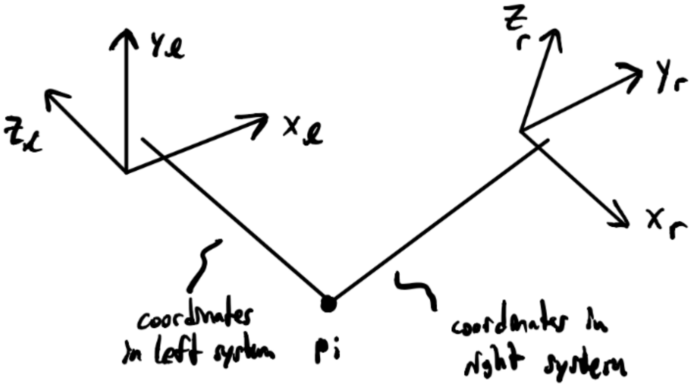


Figure 96: General case of absolute orientation: Given the coordinate systems $(\mathbf{x}_l, \mathbf{y}_l, \mathbf{z}_l) \in \mathbb{R}^{3\times3}$ and $(\mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r) \in \mathbb{R}^{3\times3}$, our goal is to find the transformation, or pose, between them using points measured in each frame of reference $\mathbf{p}_i$.

24.3.2 Relative Orientation

This problem asks how we can find the 2D ⟷ 2D relationship between two objects, such as cameras, points, or other sensors. This type of problem comes up frequently in machine vision, for instance, **binocular stereo**. Two high-level applications include:

1. Given two cameras/images that these cameras take, our goal is to extract 3D information by finding the relationship between two 2D images.
2. Given two cameras, our goal is to find the (relative) **transformation**, or **pose**, between the two cameras.

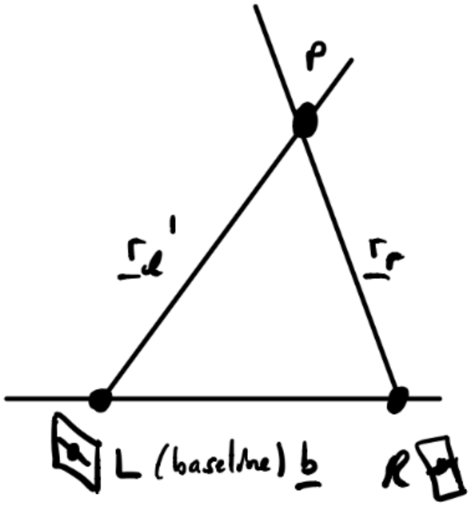


Figure 97: Binocular stereo system set up. For this problem, recall that one of our objectives is to measure the translation, or baseline, between the two cameras.

24.3.3 Exterior Orientation

This photogrammetry problem aims from going 2D ⟶ 3D. One common example in robotics (and other field related to machine vision) is **localization**, in which a robotic agent must find their location/orientation on a map given 2D information from a camera (as well as, possibly, 3D laser scan measurements).

More generally, with **localization**, our goal is to find where we are and how we are oriented in space given a 2D image and a 3D model of the world.

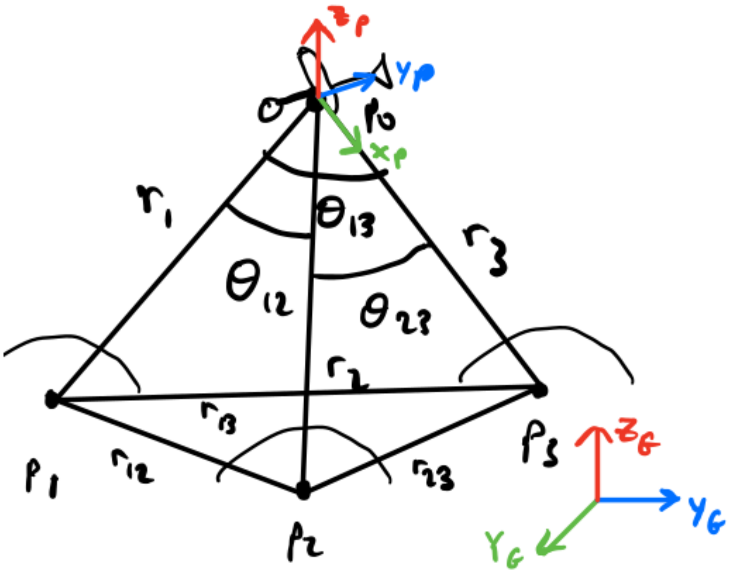


Figure 98: Exterior orientation example: Determining position and orientation from a plane using a camera and landmark observations on the ground.

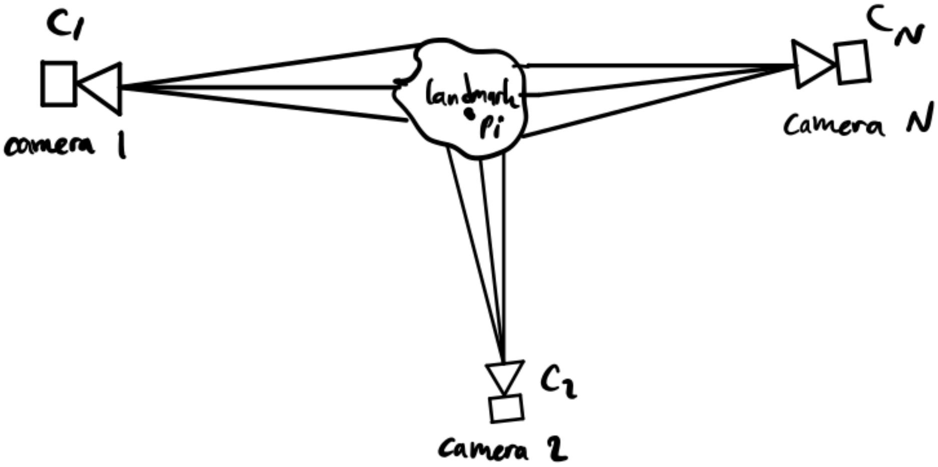


Figure 99: Bundle Adjustment (BA) is another problem class that relies on exterior orientation: we seek to find the orientation of cameras using image location of landmarks. In the general case, we can have any number of K landmark points ("interesting" points in the image) and N cameras that observe the landmarks.

24.3.4 Interior Orientation

This photogrammetry problem aims from going 3D ⟶ 2D. The most common application of this problem is **camera calibration**. Camera calibration is crucial for high-precision imaging, as well as solving machine and computer vision problems such as Bundle Adjustment [1]. Finding a principal point is another example of the interior orientation problem.

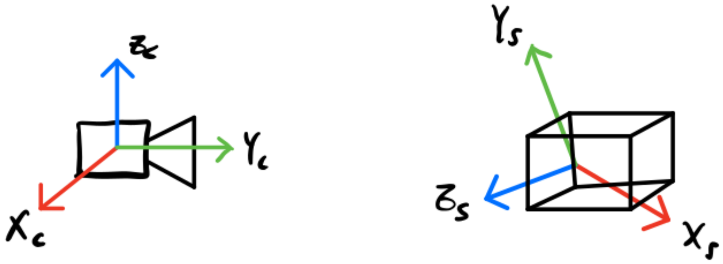


Figure 100: Interior orientation seeks to find the transformation between a camera and a calibration object - a task often known as camera calibration. This can be used, for instance, with Tsai's calibration method (note that this method also relies on exterior orientation).

24.4 Rotation

There are a myriad of representations for rotations - some of these representations include:

1. Axis and angle
2. Euler Angles
3. Orthonormal Matrices
4. Exponential cross product
5. Stereography plus bilinear complex map
6. Pauli Spin Matrices
7. Euler Parameters
8. Unit Quaternions

We would also like our representations to have the following properties:

- The ability to rotate vectors - or coordinate systems
- The ability to compose rotations
- Intuitive, non-redundant representation - e.g. rotation matrices have 9 entries but only 3 degrees of freedom
- Computational efficiency
- Interpolate orientations
- Averages over range of rotations
- Derivative with respect to rotations - e.g. for optimization and least squares
- Sampling of rotations - uniform and random (e.g. if we do not have access to closed-form solutions)
- Notion of a space of rotations

Let us delve into some of these in a little more depth.

24.4.1 Axis and Angle

This representation is composed of a vector $\hat{\boldsymbol{\omega}}$ and an angle θ, along with the **Gibb's vector** that combines these given by $\hat{\boldsymbol{\omega}} \tan\left(\frac{\theta}{2}\right)$, which has magnitude $\tan\left(\frac{\theta}{2}\right)$, providing the system with an additional degree of freedom that is not afforded by unit vectors. Therefore, we have our full 3 rotational DOF.

24.4.2 Orthonormal Matrices

We have studied these previously, but these are the matrices that have the following properties:

1. $\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = I, \mathbf{R}^T = \mathbf{R}^{-1}$ (skew-symmetric)
2. $\det|R| = +1$
3. $\mathbf{R} \in \mathbf{SO}(3)$ (see notes on groups above) - being a member of this Special Orthogonal group is contingent on satisfying the properties above.

24.4.3 Quaternions

In this section, we will discuss another way to represent rotations: quaternions.

24.4.4 Hamilton and Division Algebras

Hamilton's insight with quaternions is that these quaternions require a "4th dimension for the sole purpose of calculating triples". Hamilton noted the following insights in order to formalize his quaternion. Therefore, the complex components i, j, k defined have the following properties:

1. $i^2 = j^2 = k^2 = ijk = -1$
2. From which follows:
   (a) $ij = k$
   (b) $jk = i$
   (c) $ki = j$
   (d) $ji = -k$
   (e) $kj = -i$
   (f) $ik = -j$

   Note: As you can see from these properties, multiplication of the components of these quaternions is not commutative.

We largely use the 4-vector quaternion, denoted $\overset{o}{q} = (q, \mathbf{q}) \in \mathbb{R}^4$.

24.4.5 Properties of 4-Vector Quaternions

These properties will be useful for representing vectors and operators such as rotation later:

1. **Not commutative**: $\overset{o}{p}\overset{o}{q} \neq \overset{o}{q}\overset{o}{p}$
2. **Associative**: $(\overset{o}{p}\overset{o}{q})\overset{o}{r} = \overset{o}{p}(\overset{o}{q}\overset{o}{r})$
3. **Conjugate**: $(p, \mathbf{p})^* = (p, -\mathbf{p}) \implies (\overset{o}{p}\overset{o}{q}) = \overset{o}{q}^* \overset{o}{p}$
4. **Dot Product**: $(p, \mathbf{p}) \cdot (q, \mathbf{q}) = pq + \mathbf{p} + \mathbf{q}$
5. **Norm**: $||\overset{o}{q}||_2^2 = \overset{o}{q} \cdot \overset{o}{q}$
6. **Conjugate Multiplication**: $\overset{o}{q}\overset{o}{q}^*$:

$$\begin{aligned}\overset{o}{q}\overset{o}{q}^* &= (q, \mathbf{q})(q, -\mathbf{q}) \\ &= (q^2 + \mathbf{q}\cdot\mathbf{q}, 0) \\ &= (\overset{o}{q} \cdot \overset{o}{q})\overset{o}{e}\end{aligned}$$

   Where $\overset{o}{e} \triangleq (1, 0)$, i.e. it is a quaternion with no vector component. Conversely, then, we have: $\overset{o}{q}^*\overset{o}{q} = (\overset{o}{q}\overset{o}{q})\overset{o}{e}$.

7. **Multiplicative Inverse**: $\overset{o}{q}^{-1} = \frac{\overset{o}{q}^*}{(\overset{o}{q} \cdot \overset{o}{q}}$ (Except for $\overset{o}{q} = (0, \mathbf{0})$, which is problematic with other representations anyway.)

We can also look at properties with dot products:

1. $(\overset{o}{p}\overset{o}{q}) \cdot (\overset{o}{p}\overset{o}{q}) = (\overset{o}{p} \cdot \overset{o}{p})(\overset{o}{q} \cdot \overset{o}{q})$
2. $(\overset{o}{p}\overset{o}{q}) \cdot (\overset{o}{p}\overset{o}{r}) = (\overset{o}{p} \cdot \overset{o}{p})(\overset{o}{q} \cdot \overset{o}{r})$
3. $(\overset{o}{p}\overset{o}{q}) \cdot \overset{o}{r} = \overset{o}{p} \cdot (\overset{o}{r}\overset{o}{q}^*)$

We can also represent these quaternions as vectors. Note that these quaternions all have zero scalar component.

1. $\overset{o}{r} = (0, \mathbf{r})$
2. $\overset{o}{r}^* = (0, -\mathbf{r})$
3. $\overset{o}{r} \cdot \overset{o}{s} = \mathbf{r} \cdot \mathbf{s}$
4. $\overset{o}{r}\overset{o}{s} = (-\mathbf{r} \cdot \mathbf{s}, \mathbf{r} \times \mathbf{s})$
5. $(\overset{o}{r}\overset{o}{s}) \cdot \overset{o}{t} = \overset{o}{r} \cdot (\overset{o}{s}\overset{o}{t}) = [\mathbf{r}\ \mathbf{s}\ \mathbf{t}]$ (Triple Products)
6. $\overset{o}{r}\overset{o}{r} = -(\mathbf{r} \cdot \mathbf{r})\overset{o}{e}$

Another **note**: For representing rotations, we will use unit quaternions. We can represent scalars and vectors with:

- **Representing scalars**: $(s, \mathbf{0})$
- **Representing vectors**: $(0, \mathbf{v})$

24.4.6 Quaternion Rotation Operator

To represent a rotation operator using quaternions, we need a quaternion operation that maps from vectors to vectors. More specifically, we need an operation that maps from 4D, the operation in which quaternions reside, to 3D in order to ensure that we are in the correct for rotation in 3-space. Therefore, our rotation operator is given:

$$\begin{aligned}\overset{o}{r}' = R(\overset{o}{r}) &= \overset{o}{q}\overset{o}{r}\overset{o}{q}^* \\ &= (Q\overset{o}{r})\overset{o}{q}^* \\ &= (\bar{Q}^T Q)\overset{o}{r}\end{aligned}$$

Where the matrix $\bar{Q}^T Q$ is given by:

$$\bar{Q}^T Q = \begin{bmatrix} \overset{o}{q}\cdot\overset{o}{q} & 0 & 0 & 0 \\ 0 & q_0^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & q_0^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_z) & q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

A few notes about this matrix:

- Since the scalar component of $\overset{o}{q}$ is zero, the first row and matrix of this column are sparse, as we can see above.
- If $\overset{o}{q}$ is a unit quaternion, the lower right 3×3 matrix of $\bar{Q}^T Q$ will be orthonormal (it is an orthonormal rotation matrix).

Let us look at more properties of this mapping $\overset{o}{r}' = \overset{o}{q}\overset{o}{r}\overset{o}{q}^*$:

1. **Scalar Component**: $r' = r(\overset{o}{q}\cdot\overset{o}{q})$
2. **Vector Component**: $\mathbf{r}' = (q^2 - \mathbf{q}\cdot\mathbf{q})\mathbf{r} + 2(\mathbf{q}\cdot\mathbf{r})\mathbf{q} + 2q(\mathbf{q}\times\mathbf{r})$
3. **Operator Preserves Dot Products**: $\overset{o}{r}' \cdot \overset{o}{s}' = \overset{o}{r}\cdot\overset{o}{s} \implies \mathbf{r}'\cdot\mathbf{s}' = \mathbf{r}\cdot\mathbf{s}$
4. **Operator Preserves Triple Products**: $(\overset{o}{r}' \cdot \overset{o}{s}')\cdot\overset{o}{t}' = (\overset{o}{r}\cdot\overset{o}{s})\cdot\overset{o}{t} \implies (\mathbf{r}'\cdot\mathbf{s}')\mathbf{t}' = (\mathbf{r}\cdot\mathbf{s})\cdot\mathbf{t} \implies [\mathbf{r}'\ \mathbf{s}'\ \mathbf{t}'] = [\mathbf{r}\ \mathbf{s}\ \mathbf{t}]$
5. **Composition (of rotations!)**: Recall before that we could not easily compose rotations with our other rotation representations. Because of associativity, however, we can compose rotations simply through quaternion multiplication:

   $$\overset{o}{p}(\overset{o}{q}\overset{o}{r}\overset{o}{q}^*)\overset{o}{p}^* = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{q}^*\overset{o}{p}^*) = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{p}\overset{o}{q})^*$$

   I.e. if we denote the product of quaternions $\overset{o}{z} \triangleq \overset{o}{p}\overset{o}{q}$, then we can write this rotation operator as a single rotation:

   $$\overset{o}{p}(\overset{o}{q}\overset{o}{r}\overset{o}{q}^*)\overset{o}{p}^* = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{q}^*\overset{o}{p}^*) = (\overset{o}{p}\overset{o}{q})\overset{o}{r}(\overset{o}{p}\overset{o}{q})^* = \overset{o}{z}\overset{o}{r}\overset{o}{z}^*$$

   This ability to compose rotations is quite advantageous relative to many of the other representations of rotations we have seen before (orthonormal rotation matrices can achieve this as well).

24.5 3D Recognition

24.5.1 Extended Gaussian Image

The idea of the extended Gaussian Image: what do points on an object and points on a sphere have in common? They have the same surface normals.

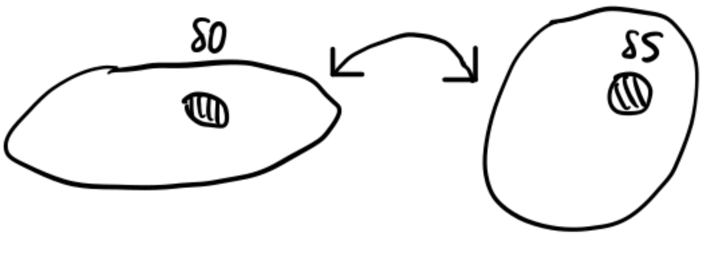


Figure 101: Mapping from object space the Gaussian sphere using correspondences between surface normal vectors. Recall that this method can be used for tasks such as image recognition and image alignment.

- **Curvature**: $\kappa = \frac{\delta S}{\delta O} = \lim_{\delta\to 0}\frac{\delta S}{\delta O} = \frac{dS}{dO}$
- **Density**: $G(\eta) = \frac{\delta O}{\delta S} = \lim_{\delta\to 0}\frac{\delta O}{\delta S} = \frac{dO}{dS}$

24.5.2 EGI with Solids of Revolution

Are there geometric shapes that lend themselves well for an "intermediate representation" with EGI (not too simple, nor too complex)? It turns out there are, and these are the **solids of revolution**. These include:

- Cylinders
- Spheres
- Cones
- Hyperboloids of one and two sheets

How do we compute the EGI of solids of revolution? We can use **generators** that produce these objects to help.

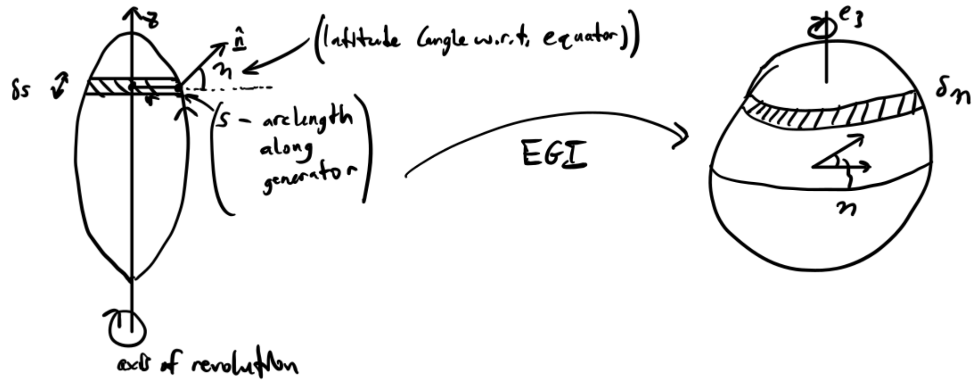


Figure 102: EGI representation of a generalized solid of revolution. Note that bands in the object domain correspond to bands in the sphere domain.

As we can see from the figure, the bands "map" into each other! These solids of revolution are symmetric in both the object and transform space. Let's look at constructing infinitesimal areas so we can then compute Gaussian curvature κ and density G:

- **Area of object band**: $\delta O = 2\pi r \delta s$
- **Area of sphere band**: $\delta S = 2\pi\cos(\eta)\delta n$

Then we can compute the curvature as:

$$\kappa = \frac{\delta S}{\delta O} = \frac{2\pi\cos(\eta)\delta\eta}{2\pi r\delta s} = \frac{\cos(\eta)\delta\eta}{r\delta s}$$
$$G = \frac{1}{\kappa} = \frac{\delta O}{\delta S} = r\sec(\eta)\frac{\delta s}{\delta\eta}$$

Then in the limit of $\delta \to 0$, our curvature and density become:

$$\kappa = \lim_{\delta\to O}\frac{\delta S}{\delta O} = \frac{\cos\eta}{r}\frac{d\eta}{ds} \text{ (Where } \frac{d\eta}{ds} \text{ is the rate of change of surface normal direction along the arc, i.e. curvature)}$$
$$G = \lim_{\delta\to 0}\frac{\delta O}{\delta S} = r\sec\eta\frac{ds}{d\eta} \text{ (Where } \frac{ds}{d\eta} \text{ is the rate of change of the arc length w.r.t. angle)}$$
$$\kappa = \frac{\cos\eta}{r}\kappa$$

Recall that we covered this for the following

24.5.3 Sampling From Spheres Using Regular and Semi-Regular Polyhedra

More efficient to sample rotations from shapes that form a "tighter fit" around the sphere - for instance: **polyhedra**! Some polyhedra we can use:

- **Tetrahedra** (4 faces)
- **Hexahedra** (6 faces)

- **Octahedra** (8 faces)
- **Dodecahedra** (12 faces)
- **Icosahedra** (20 faces)

These polyhedra are also known as the **regular solids**.

As we did for the cube, we can do the same for polyhedra: to sample from the sphere, we can sample from the polyhedra, and then **project** onto the point on the sphere that intersects the line from the origin to the sampled point on the polyhedra. From this, we get **great circles** from the edges of these polyhedra on the sphere when we project.

Fun fact: Soccer balls have 32 faces! More related to geometry: soccer balls are part of a group of **semi-regular** solids, specifically an **icosadodecahedron**.

24.5.4 Desired Properties of Dividing Up the Sphere/Tessellations

To build an optimal representation, we need to understand what our desired optimal properties of our tessellations will be:

1. **Equal areas of cells/facets**
2. **Equal shapes of cells/facets**
3. **"Rounded" shapes of cells/facets**
4. **Regular pattern**
5. **Allows for easy "binning"**
6. **Alignment of rotation**

Platonic and **Archimedean** solids are the representations we will use for these direction histograms.

24.6 References

1. Groups, https://en.wikipedia.org/wiki/Group_(mathematics)