From the last lecture, the optical flow energy is:

$$\mathcal{E}(X) = \int_D \left[\| I(\Phi^X_{0,t}(x), t) - I_0(x) \| + \alpha^2 \|\nabla u\|^2 + \alpha^2 \|\nabla v\|^2 \right] d\vec{x}$$

where $X = \begin{Bmatrix} u \\ v \end{Bmatrix}$

don't forget functional dependence on spatial element $x \in D \subset \mathbb{R}^2$

$$X(x) = \begin{Bmatrix} u(x) \\ v(x) \end{Bmatrix}$$

A linearized version was given to be:

$$\mathcal{E}(X) \approx \int_D \left[\|\nabla I \cdot X + I_t\|^2 + \alpha^2 \|\nabla u\|^2 + \alpha^2 \|\nabla v\|^2 \right] d\vec{x}$$

whose variation was found to be:

$$\frac{d}{d\epsilon}\Big|_{\epsilon=0} \mathcal{E}(X + \epsilon \delta X) = 2 \int_D \left[(\nabla I \cdot X + I_t) \nabla I - \alpha^2 \begin{Bmatrix} \Delta u \\ \Delta v \end{Bmatrix} \right] \cdot \delta X \, d\vec{x}$$

Option 1] gradient descent: pick δX so that energy decreases.

This would be

$$\delta X = - \left[(\nabla I \cdot X + I_t) \nabla I - \alpha^2 \begin{Bmatrix} \Delta u \\ \Delta v \end{Bmatrix} \right]$$

⇒ break up into components.

$$\delta u = -(\nabla I \cdot X + I_t) I_x + \alpha^2 \Delta u$$

$$\delta v = -(\nabla I \cdot X + I_t) I_y + \alpha^2 \Delta v$$

⇒ apply as an update

$$u = u + d\tau \left[\alpha^2 \Delta u - (\nabla I \cdot X + I_t) I_x \right]$$

$$v = v + d\tau \left[\alpha^2 \Delta v - (\nabla I \cdot X + I_t) I_y \right]$$

- it is not clear what $d\tau$ should be. best to start out tiny to see what happens & how many iterations are needed, then see if can make bigger.
- to make sure it is OK, the max and min values of u & v should make sense. Huge values means $d\tau$ probably too big.
- method is sort of slow & can get stuck in local minima.

if it is written more explicitly:

$$u = u + d\tau \left[\alpha^2 \Delta u - I_x^2 u - I_x I_y v - I_x I_t \right]$$

$$v = v + d\tau \left[\alpha^2 \Delta v - I_x I_y u - I_y^2 v - I_x I_t \right]$$

Option 2] Solve for X directly to make left part of integrand vanish. It is a linear problem, so an exact solution exists.

Essentially this is; find $X = \begin{Bmatrix} u \\ v \end{Bmatrix}$ so that

$$(\nabla I \cdot X + I_t)\nabla I - \alpha^2 \begin{Bmatrix} \Delta u \\ \Delta v \end{Bmatrix} = 0$$

$\Rightarrow$

$$(\nabla I \cdot X + I_t) I_x - \alpha^2 \Delta u = 0$$

$$(\nabla I \cdot X + I_t) I_y - \alpha^2 \Delta v = 0$$

$\Rightarrow$

$$I_x^2 u + I_x I_y v + I_x I_t - \alpha^2 \Delta u = 0$$

$$I_x I_y u + I_y^2 v + I_y I_t - \alpha^2 \Delta v = 0$$

$\Rightarrow$

$$(I_x^2 - \alpha^2 \Delta) u + I_x I_y v = -I_x I_t$$

$$I_x I_y u + (I_y^2 - \alpha^2 \Delta) v = -I_y I_t$$

- the Laplacian operator Δ is a linear operator, so as a discrete system, this actually can be posed as a large system of linear equations in the unknowns u and v, and solved exactly. the setup is a little complicated but has nice structure that makes it simpler than it seems.

- we will not solve the linear system. Instead, we'll look at an iterative alternative, the one proposed by Horn & Schunck.

Horn & Schunck rearranged the problem as follows:

$$I_x^2 u + I_x I_y v = -I_x I_t + \alpha^2 \Delta u$$

$$I_x I_y u + I_y^2 v = -I_y I_t + \alpha^2 \Delta v$$

furthermore, as a discrete operator, the Laplace operator is equal to:

$$\Delta u = K(\bar{u} - u)$$

↑ to be defined soon.

for example, we showed in class that one possibility was

$$\Delta u = u(x+1,y) + u(x-1,y) + u(x,y+1) + u(x,y-1) - 4u(x,y)$$

$$\Rightarrow$$

$$\Delta u = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} * u - 4u = 4\left(\begin{bmatrix} 0 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 0 \end{bmatrix} * u - u \right)$$

↑ convolution

$$= 4(\bar{u} - u)$$

in their paper, they use

$$K = 3 \quad \text{and} \quad \bar{u} = \begin{bmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{bmatrix} * u$$

the same holds for V,

$$\Delta V = 4\left(\begin{bmatrix} 0 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 0 \end{bmatrix} * V - V\right)$$

Well, Horn & Schunck consider a more generic case,

$$\Delta u = K\left[L * u - u\right] \quad , \quad \Delta V = K\left[L * V - V\right]$$

where L is a convolution stencil for the Laplacian.

as an example, they give

$$L = \begin{bmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{bmatrix} \quad \& \quad K = 3$$

What about the gradients I_x, I_y, I_t?

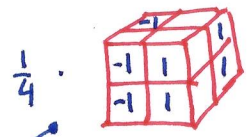
Well, the do not use the forward Euler stencil $[-1\ 1]$, but an averaged version of it.

$$I_x = \frac{1}{4}\Big[I(i,j+1,k) - I(i,j,k) + I(i+1,j+1,k) - I(i+1,j,k) + I(i,j+1,k+1) - I(i,j,k+1) + I(i+1,j+1,k+1) - I(i+1,j,k+1) \Big]$$

- $I(\cdot,\cdot,k)$ is "first" image, $I(\cdot,\cdot,k+1)$ is "second" image.

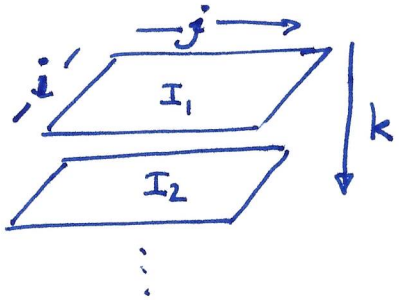
- note that it is the average of four forward Euler stencils for the x-derivative.

- as a stencil across the two stacked images,



← done more simply it is: $\frac{1}{4}\left(\begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix} * I_1 + \begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix} * I_2\right)$

- just a reminder, for images



Same idea for I_y & I_t but with altered stencil

$$I_y = \frac{1}{4}\Big[\, I(i+1,j,k) - I(i,j,k) + I(i+1,j+1,k) - I(i,j+1,k) + I(i+1,j,k+1) - I(i,j,k+1) + I(i+1,j+1,k+1) - I(i,j+1,k+1)\Big]$$

$$I_t = \frac{1}{4}\Big[\, I(i,j,k+1) - I(i,j,k) + I(i+1,j,k+1) - I(i+1,j,k) + I(i,j+1,k+1) - I(i,j+1,k) + I(i+1,j+1,k+1) - I(i+1,j+1,k)\Big]$$

- can you figure out the simple stencils for these?

$\Rightarrow$

$$I_x^2 u + I_x I_y v = -I_x I_t + \alpha^2 K \bar{u} - \alpha^2 K u$$

$$I_x I_y u + I_y^2 v = -I_y I_t + \alpha^2 K \bar{v} - \alpha^2 K v$$

$\Rightarrow$

$$(I_x^2 + \alpha^2 K) u + I_x I_y v = -I_x I_t + \alpha^2 K \bar{u}$$

$$I_x I_y u + (I_y^2 + \alpha^2 K) v = -I_y I_t + \alpha^2 K \bar{v}$$

$\Rightarrow$

$$\begin{bmatrix} I_x^2 + \alpha^2 K & I_x I_y \\ I_x I_y & I_y^2 + \alpha^2 K \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} \alpha^2 K \bar{u} - I_x I_t \\ \alpha^2 K \bar{v} - I_y I_t \end{Bmatrix}$$

$\Rightarrow$

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} I_x^2 + \alpha^2 K & I_x I_y \\ I_x I_y & I_y^2 + \alpha^2 K \end{bmatrix}^{-1} \begin{Bmatrix} \alpha^2 K \bar{u} - I_x I_t \\ \alpha^2 K \bar{v} - I_y I_t \end{Bmatrix}$$

↓ to invert, need determinant

$$\det: (I_x^2 + \alpha^2 K)(I_y^2 + \alpha^2 K) - I_x^2 I_y^2$$

$$= \cancel{I_x^2 I_y^2} + \alpha^2 K I_x^2 + \alpha^2 K I_y^2 + \alpha^4 K^2 - \cancel{I_x^2 I_y^2}$$

$$= \alpha^2 K \left(I_x^2 + I_y^2 + \alpha^2 K \right)$$

$$\Rightarrow$$

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \frac{\begin{bmatrix} I_y^2 + \alpha^2 K & -I_x I_y \\ -I_x I_y & I_x^2 + \alpha^2 K \end{bmatrix} \begin{Bmatrix} \alpha^2 K \bar{u} - I_x I_t \\ \alpha^2 K \bar{v} - I_y I_t \end{Bmatrix}}{\alpha^2 K (I_x^2 + I_y^2 + \alpha^2 K)}$$

$$= \frac{\begin{Bmatrix} \alpha^2 K I_y^2 \bar{u} + \alpha^4 K^2 \bar{u} - I_x I_y^2 I_t - \alpha^2 K I_x I_t - \alpha^2 K I_x I_y \bar{v} + I_x I_y^2 I_t \\ -\alpha^2 K I_x I_y \bar{u} + I_x^2 I_y I_t + \alpha^2 K \bar{v} I_x^2 + \alpha^4 K^2 \bar{v} - \alpha^2 K I_y I_t - I_x^2 I_y I_t \end{Bmatrix}}{\alpha^2 K (I_x^2 + I_y^2 + \alpha^2 K)}$$

$$= \frac{\begin{Bmatrix} I_y^2 \bar{u} + \alpha^2 K \bar{u} + I_x^2 \bar{u} - I_x^2 \bar{u} - I_x I_t - I_x I_y \bar{v} \\ -I_x I_y \bar{u} + I_x^2 \bar{v} + \alpha^2 K \bar{v} + I_y^2 \bar{v} - I_y^2 \bar{v} - I_y I_t \end{Bmatrix}}{I_x^2 + I_y^2 + \alpha^2 K}$$

$$= \begin{Bmatrix} \bar{u} \\ \bar{v} \end{Bmatrix} - (I_x^2 + I_y^2 + \alpha^2 K)^{-1} \begin{Bmatrix} I_x (I_x \bar{u} + I_y \bar{v} - I_t) \\ I_y \underbrace{(I_x \bar{u} + I_y \bar{v} - I_t)} \end{Bmatrix}$$

this part is the same.

$$\Rightarrow$$

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} \bar{u} \\ \bar{v} \end{Bmatrix} - \frac{(I_x \bar{u} + I_y \bar{v} - I_t)}{(I_x^2 + I_y^2 + \alpha^2 K)} \begin{Bmatrix} I_x \\ I_y \end{Bmatrix}$$

- because, after each update of u and v, the quantities $\bar{u}$ and $\bar{v}$ change, this needs to be run a bunch of times. Thus, it is necessary to specify some number of iterations.

Things to note:

these parts do change w/each iteration.

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} \bar{u} \\ \bar{v} \end{Bmatrix} - \frac{(I_x \bar{u} + I_y \bar{v} - I_t)}{(I_x^2 + I_y^2 + \alpha^2 K)} \begin{Bmatrix} I_x \\ I_y \end{Bmatrix}$$

these parts don't change w/each iteration.

- in the for loop over the # of iterations, only the parts that change need to be updated. The rest can be precomputed & reused w/every iteration.

[Proving Correctness of Horn & Schunck Optical Flow Procedure]

Here, I would like to spend a little time showing you why the approach of Horn & Schunck works, e.g., how starting from an initial u & v, one can approach the solution u^* & v^* of the linear system.

The solution strategy is somewhat common & may be helpful in other areas.

In the optical flow case, we can "stack" the linear equations for u & v for all of the pixels.

Recall, that for each pixel, we are trying to solve:

$$\begin{bmatrix} (I_x^2 - \alpha^2\Delta) & I_x I_y \\ I_x I_y & (I_y^2 - \alpha^2\Delta) \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} -I_x I_t \\ -I_y I_t \end{Bmatrix}$$

when discretized, the Laplacian operator will depend on u & v as evaluated at the current pixel plus its neighboring pixels.

It takes a little trickery, but if the entire "image" of u & v ~~nodes~~ values were vectorized, then the ~~resulting~~ equivalent linear system would be

$$Az = b \qquad \text{where} \quad z = \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix}$$

combined vectorizations of u & v.

[An Iterative Method for Solving Some Linear Systems of Equations]

Now, the matrix A actually has some very nice structure. It can be broken up into a diagonal part plus a well-behaved non-diagonal part; $A = D + N$. Using this information,

$$Az = b$$

$$\Rightarrow$$

$$(D+N)z = b$$

↳ suppose that z^* is actually the true solution. then

$$(D+N)z^* = b$$

$$\Rightarrow$$

$$Dz^* + Nz^* = b$$

$$\Rightarrow$$

$$Dz^* = b - Nz^*$$

$$\Rightarrow$$

$$z^* = D^{-1}(b - Nz^*) \qquad (\dagger)$$

↳ remember that this is diagonal (with non-zero entries), so it is invertible. This inverse is easy to find (like we did already).

if z^* is the solution, then it satisfies equation $(\dagger)$. if not, then it won't.

If, instead of the true solution z^*, we had a guess z_0, then what? Well using the following approach,

$$z_{n+1} = D^{-1}(b - Nz_n)$$

leads one closer to the true solution, interestingly enough. This must be performed for some amount of iterations. Usually one stops when the difference $\|z_{n+1} - z_n\|$ is small enough.

Here is the proof that ~~eventually~~ z_n converges to z^* as n goes to ∞ (although, normally one doesn't have to go that far!)

Consider the error associated to the next guess:

$$e_{k+1} = z_{k+1} - z^*$$

$$= D^{-1}(b - Nz_k) - D^{-1}(b - Nz^*)$$

$$= D^{-1}b - D^{-1}Nz_k - D^{-1}b + D^{-1}Nz^*$$

$$= -D^{-1}N(z_k - z^*)$$

$$= -D^{-1}N(e_k)$$

$$e_{k+1} = -D^{-1}Ne_k$$

↑ it's related to the error of the previous gu

OK, so $$e_{k+1} = -D^{-1} N e_k$$

turns out that the norm/length of the error decreases if the eigenvalues of $D^{-1}N$ lie inside of the range $(-1, 1)$.

Basically $$\|e_{k+1}\| = \|D^{-1}N e_k\|$$

$$\leq \|D^{-1}N\| \cdot \|e_k\|$$

$$\leq |\lambda| \, \|e_k\|$$

↑ for some eigenvalue λ of the matrix $D^{-1}N$.

if $|\lambda| < 1$ for all possible λ, then

$$\|e_{k+1}\| < \|e_k\|$$ ← error gets smaller!

and with each iteration $\|e_k\|$ gets smaller.

Thus $\|e_k\|$ goes to zero as k goes to infinity.

We have

$$\lim_{k \to 0} \|e_k\| = 0$$

$$\Rightarrow$$

$$\lim_{k \to \infty} \|Z_k - Z^*\| = 0 .$$

$$\Rightarrow$$

$$Z_k \to Z^* \quad \text{as} \quad k \to \infty .$$

↑ thus H&S gave a sane technique for computing optical flow.

[Implementation of Horn & Schunck Optical Flow Algorithm]

Given that only parts of the iterative solution really update w/each iteration, it is possible to "optimize" on the calculations performed per iteration. Here is the overall idea:

```
begin w/initial u & v      (some people start w/zero)
compute Ix, Iy, It
compute denominator

for # iterations
    compute ū, v̄
    compute numerator
    update u & v
end
```

- COMPUTE $\bar{u}, \bar{v}$, & the gradients (I_x, I_y, I_t)

in class, I've used

$$\Delta u = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 0 \\ 0 & 1 & 0 \end{bmatrix} * u = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} * u - 4u$$

$$= 4\left(\begin{bmatrix} 0 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 0 \end{bmatrix} * u - u \right)$$

↳ see how it really ends up being an average?

↳ of the 4 neighbors