Video Processing

Lesson 2 – Optical Flow & Lucas Kanade

General stuff

- Student pairs (for submission) can be from different groups. **Use the forum. Notify me if you haven't found.**
- **Questions about HW should be asked in the forum on the course site.**
  - **Try to answer each other.**
  - **Read previous answers.**
- Although Lessons aren't mandatory, it can answer many of the questions in the forum. Looping over the PDF isn't like watching the lesson.
- Try to debug your own code (if you have a general question, feel free to ask).
  - No need to upload your code as you should be able to solve your own problems in your code (it will help you in the future).
  - Same goes to Python issues.
  - **Google it** ☺

HW1 repeating questions

- I showed a ballpark estimate of the result. Try to understand if your results make sense (are you receiving more then one corner on each tile?).
- It's not a programming course, you are expected to read/research and then implement the algorithm. The lessons are supposed to help you. You are supposed to set the thresholds such that your code will produce the best results. Your grade won't be much effected if your algorithm is correct.
- Regarding Sobel filter, you are expected to read about it:
  - From Wikipedia: The operator uses **two** 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical.
  - Python code from previous lesson contained an implementation of the Sobel filter ☺ You can see how to use it….
  - So asking if you should use the X or Y derivatives…I have no objection that you will use opencv functions, but you are required to understand them…

More About HW

- This exercise due date will be by the end of April.
- I will send a message about the next lesson (after Passover).
- Final project will be posted after the third HW assignment.

Optical Flow- Motivation

- What is Optical Flow?
  - Computing the movement (flow) of each pixel (or a set of pixels) between two given frames.
  - For each pixel, we'll find the displacement (horizontal and vertical, often refer as u,v) between the input frame and the target frame.
  - Often visualized as a vector field. Given the (u,v) pair for each pixel, we can also represent it using vectors (direction + magnitude).
  - Can be dense/sparse.

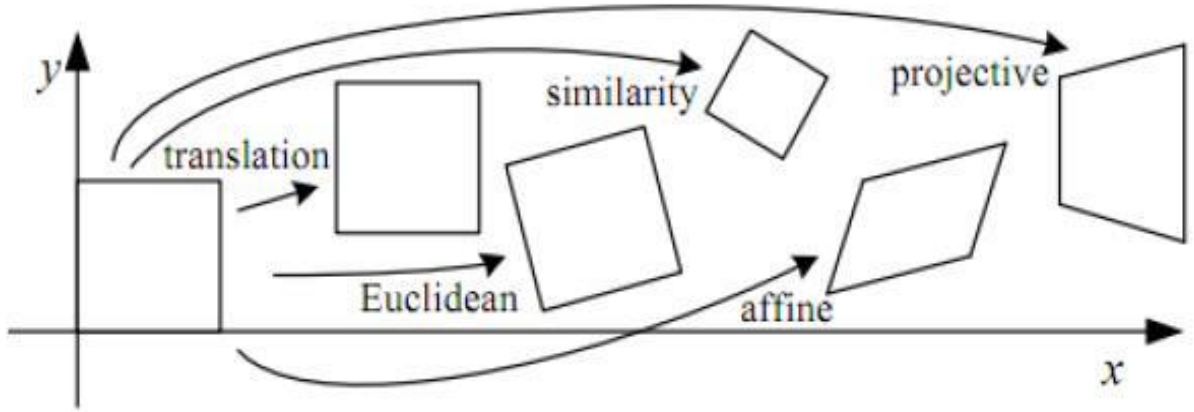
https://gfycat.com/bewitchedkindheartedclownanemonefish

Optical Flow- Applications

- Video stabilization (HW)
- Scene understanding (moving/static objects)
- Tracking (for various tasks)
- Motion segmentation
- Human pose estimation
- Action classification

Image Transformation

- Why not use the most complex transformation (projective)?

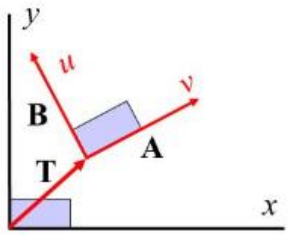


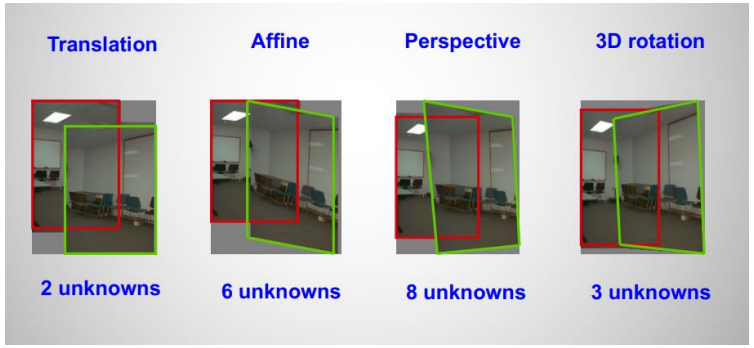
Similarity transformation (rotation + translation)

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} r_{xx} & r_{xy} & tr_x \\ r_{yx} & r_{yy} & tr_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

only rotation

translation





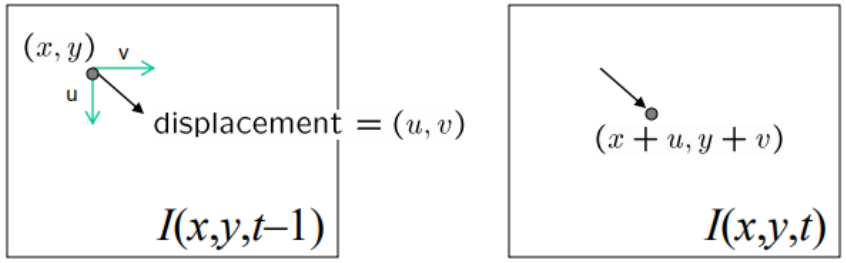
Optical Flow – Harris recap

$$E(u,v)_{\min u,v} = \sum_{(x,y)}[I(x+u,y+v) - I(x,y)]^2 \quad \approx \sum [I(x,y) + uI_x + vI_y - I(x,y)]^2 \quad \text{First order approx}$$

$$E(u,v) \approx [u \;\; v] \sum_{x,y} \begin{bmatrix} I_x^2 & I_x * I_y \\ I_x * I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

- Note that Taylor series is correct only for small movements.

Optical Flow - Formula



Input: two images; output: (u,v) for each pixel, such that:

$$I(x,y,t-1) = I(x+u(x,y), y+v(x,y), t)$$

$$I(x+u,y+u,t) \approx I(x,y,t-1) + I_x \cdot u(x,y) + I_y \cdot v(x,y) + I_t$$

$$I(x+u,y+u,t) - I(x,y,t-1) = I_x \cdot u(x,y) + I_y \cdot v(x,y) + I_t$$

Hence, $I_x \cdot u + I_y \cdot v + I_t \approx 0$ $\rightarrow \nabla I \cdot [u \;\; v]^T + I_t = 0$

$$I_t = I_2 - I_1$$

For each pixel we get a different equation!

Two variables (u,v); one equation (scalar)!

I(x,y,t-1)

I(x,y,t)

Taken from: http://www.cs.ucf.edu/~bagci/teaching/computervision17/Lec7.pdf
And
https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf

The Brightness Constancy Constraint

What's the problem with the equation we're solving: $I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t)$

What happens when we reflective objects in the image?

I(x,y,t-1)

I(x,y,t)

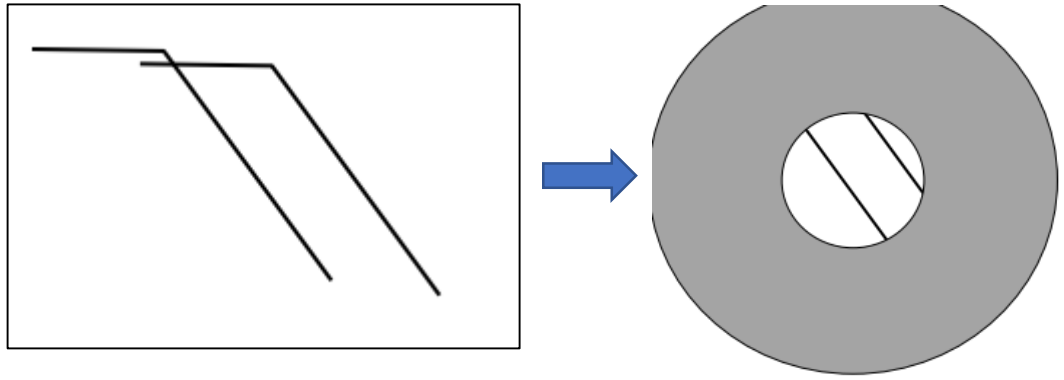
Is RGB the best option? There are other color spaces (YUV, HSV,..) which can help us.

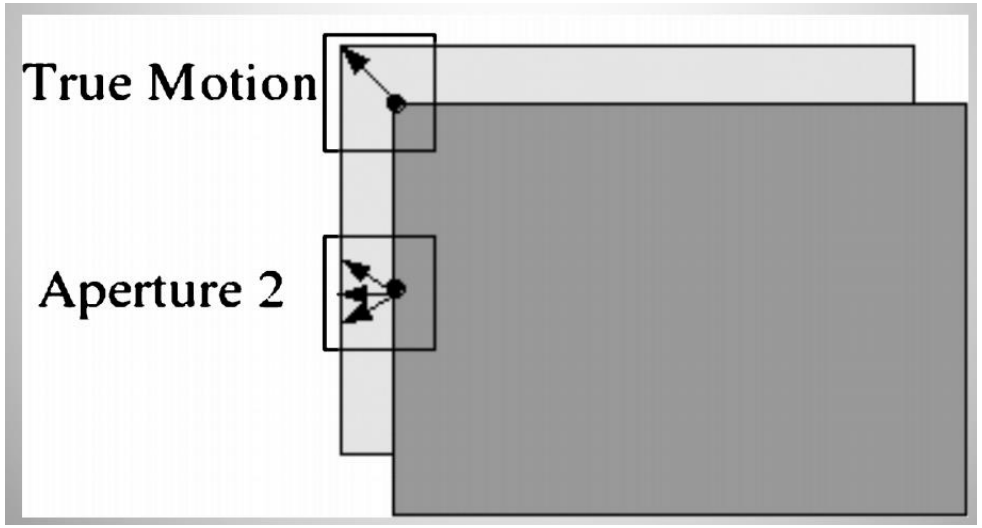
Aperture Problem

What can we do about it?

$$M = \begin{bmatrix} I_x^2 & I_x * I_y \\ I_x * I_y & I_y^2 \end{bmatrix}$$ Edge→ rank(M) = 1

Window size?





Taken from: http://www.cs.umd.edu/~djacobs/CMSC426/OpticalFlow.pdf
http://www.cs.ucf.edu/~bagci/teaching/computervision17/Lec7.pdf

Least Squares Solution

Least squares solution of linear system of equations

$$Ax = b$$

$$A^T Ax = A^T b$$

Where $A^T A$ is square and symmetric

The least squares solution x = $(A^T A)^{-1} A^T b$
Makes $||Ax - b||^2$ minimal

Lucas-Kanade (LK) Optical Flow - Concept

$$I_x \cdot u + I_y \cdot v + I_t \approx 0$$

$$I_t = I_2 - I_1$$

- We saw that we can't solve the general optical flow, as the problem is ill-posed (one equation, two variables).

- LK assume the neighborhood (of each pixel) of each pixel, moves the same.
  - Why is this important?
  - Usually assumes a windows of 5X5.

Overconstrained linear system

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix} \qquad \underset{25\text{x}2}{A} \quad \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

Least squares solution for *d* given by $(A^T A)\ d = A^T b$

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

$A^T A$ $\qquad$ $A^T b$

M matrix (Harris)! what does it mean?
(Solving it for flat/edge/corner regions)

$$M = A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

Taken from: https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf

Low-texture region

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

– gradients have small magnitude

– small λ_1, small λ_2

Taken from: https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf

Edge

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

– gradients very large or very small

– large λ_1, small λ_2

Taken from: https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf

High-texture region

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

– gradients are different, large magnitudes

– large λ_1, large λ_2

Taken from: https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf

Lucas-Kanade (LK) Optical Flow - Review

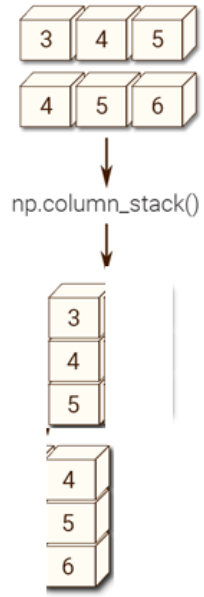
- Assumptions:
  - Small motion (points do not move very far).
  - Brightness constancy – projection of the same point looks the same in every frame.
  - Spatiail coherence: points move like their neighbors.
- We saw two types of 2D transformations:
  - Translation – $\mathrm{W}(\mathrm{x};\mathrm{p}) = \begin{pmatrix} x+P_1 \\ y+P_2 \end{pmatrix}$
  - Affine - $W(x;p) = \begin{pmatrix} P_1x+P_2y+P_3 \\ P_1x+P_2y+P_4 \end{pmatrix}$
- **The aim of the LK algorithm is to minimize:**
  $\boldsymbol{p^* = argmin_p \sum_x [I_2(w(x;p)) - I_1(x)]^2}$ **;** $\boldsymbol{x - the\ observed\ area}$**, p – warp parameters**

Lucas-Kanade (LK) Optical Flow - Review

- Incremental algorithm:
  - $\Delta p^* = argmin_{\Delta p} \sum_x [I_2(w(x; p + \Delta p) - I_1(x)]^2$
  - $p_{new} = p + \Delta p$
  - $p_0 - \ initializtion \ vector$
- Approximation:

$$I_2\big(W(x; p + \Delta p)\big) \cong I_2\big(w(x; p)\big)_{NX1} + \nabla I_{2\,NX2} \cdot \frac{\partial w}{\partial p} \cdot \Delta p$$

- Note that $\frac{\partial w}{\partial p}$ is the jacobian of the transformation.
  - In the case of translation, $\frac{\partial w}{\partial p} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
- Δp is the delta warp parameters (In the case of translation $\in R^{2X1}$)
- ∇I_2 is the image derivatives ($\nabla I_2 = \begin{bmatrix} I_x & I_y \end{bmatrix} \in R^{nX2}$)



Lucas-Kanade (LK) Optical Flow - Review

$$\Delta p^* = argmin_{\Delta p} \sum_x [I_2(w(x; p + \Delta p) - I_1(x)]^2 = argmin_{\Delta p} \sum_x [I_2\big(w(x; p)\big) + \nabla I_2 \cdot \Delta p - I_1(x)]^2 = argmin_{\Delta p} \sum_x [I_t + \nabla I_2 \cdot \Delta p]^2$$

$$I_t = I_2\big(w(x; p)\big) - I_1(x)$$

Where the sum is over all pixels in the area (window)…can be the entire image.

- In the least squares case we wish to minimize, we obtain the equation:

$$\rightarrow \nabla I_2 \Delta p = -I_t$$

- If we assume that in each area x, all pixels $c_i \in x$ transforme from I_1 to I_2 with the same parameters (Δp):

$$\nabla I_2(c_1) \cdot \Delta p = -I_t(c_1)$$
$$\dots$$
$$\nabla I_2(c_i) \cdot \Delta p = -I_t(c_i)$$
$$\dots$$
$$\nabla I_2(c_n) \cdot \Delta p = -I_t(c_n)$$

$$\underline{\underline{B}} = \nabla I = \{[I_x\ I_y]\}_{i=1}^{n} = \begin{bmatrix} I_x(c_1) & I_y(c_1) \\ \dots & \dots \\ I_x(c_n) & I_y(c_n) \end{bmatrix} \in R^{nX2}$$

$$\Delta p = \begin{pmatrix} dP_1 \\ dP_2 \end{pmatrix} \in R^{2X1}; \qquad \underline{I_t} = -\begin{bmatrix} I_t(c_1) \\ \dots \\ I_t(c_n) \end{bmatrix} \in R^{nX1}$$

$$\rightarrow \underline{\underline{B}} \cdot \Delta p = -\underline{I_t}$$

$$\underline{\underline{B}}^T \cdot \underline{\underline{B}} \cdot \Delta p = -\underline{\underline{B}}^T \cdot \underline{I_t}$$

$$\boxed{\Delta p = -\left(\underline{\underline{B}}^T \cdot \underline{\underline{B}}\right)^{-1} \cdot \underline{\underline{B}}^T \cdot \underline{I_t}}$$

Lucas-Kanade (LK) Optical Flow - Review

For translation we will get:

$$\Delta p = -\left(\underline{\underline{B}}^T \cdot \underline{\underline{B}}\right)^{-1} \cdot \underline{\underline{B}}^T \cdot \underline{I_t} = -\begin{pmatrix} I_x^T I_x & I_x^T I_y \\ I_y^T I_x & I_y^T I_y \end{pmatrix}^{-1} \begin{pmatrix} I_x^T I_t \\ I_y^T I_t \end{pmatrix}$$

$$\underline{\underline{B}} = \nabla I = \{[I_x\ I_y]\}_{i=1}^{n} = \begin{bmatrix} I_x(c_1) & I_y(c_1) \\ \dots & \dots \\ I_x(c_n) & I_y(c_n) \end{bmatrix} \in R^{nX2}$$

$$\Delta p = \begin{pmatrix} dP_1 \\ dP_2 \end{pmatrix} \in R^{2X1}; \qquad \underline{I_t} = -\begin{bmatrix} I_t(c_1) \\ \dots \\ I_t(c_n) \end{bmatrix} \in R^{nX1}$$

$$\rightarrow \underline{\underline{B}} \cdot \Delta p = -\underline{I_t}$$

$$\underline{\underline{B}}^T \cdot \underline{\underline{B}} \cdot \Delta p = -\underline{\underline{B}}^T \cdot \underline{I_t}$$

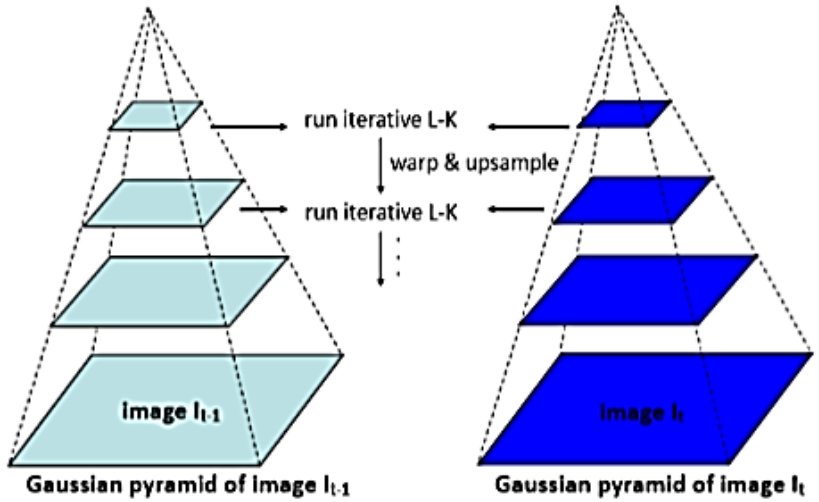
$$\boxed{\Delta p = -\left(\underline{\underline{B}}^T \cdot \underline{\underline{B}}\right)^{-1} \cdot \underline{\underline{B}}^T \cdot \underline{I_t}}$$

Revisiting the small motion assumption

Is this motion small enough?

• Probably not—it's much larger than one pixel (2nd order terms dominate).

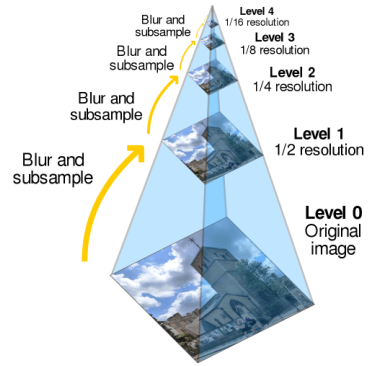
• How might we solve this problem?



What happens to (u,v) when we move from low resolution level to the higher resolution level?

When down-sampling an image, we would get aliasing if the effective sampling rate is below the Nyquist frequency. We need to prefilter the image before down-sampling. Ideally the prefilter should be a low-pass filter with a cut-of frequency half of the new sampling rate. **In practice we may use simple averaging (or a gaussian filter).**

No need to up-sample in our case as we can save the original image.



Lucas-Kanade (LK) Optical Flow - Algorithm

Can differ between implementations ☺
Decide what to do around the edges

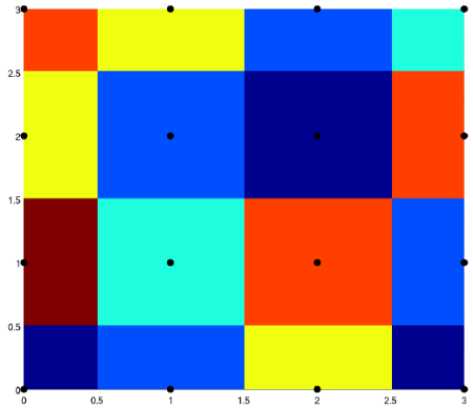
- Compute image pyramid for I1, I2
- for level:NumLevelsPyramid:
- I2_warp = WarpImage(Pyr2[level],u,v)
- for step in range(MaxIter):
  - du , dv = LucasKanadeSingleIteration(Pyr1[level],I2_warp,WindowSize) → For each pixel computes: $\Delta p = -\left(\underline{\underline{B}}^T \cdot \underline{\underline{B}}\right)^{-1} \cdot \underline{\underline{B}}^T \cdot \underline{I_t}$
  - u = u + du
  - v = v + dv
  - I2_warp = WarpImage(Pyr2[q],u,v)
- Adjust u,v according to the level (multiply by 2)

How to determine the number of levels in the pyramid and number of iterations?
What is the effect of the window size? What transformation model do we assume?

Interpolation

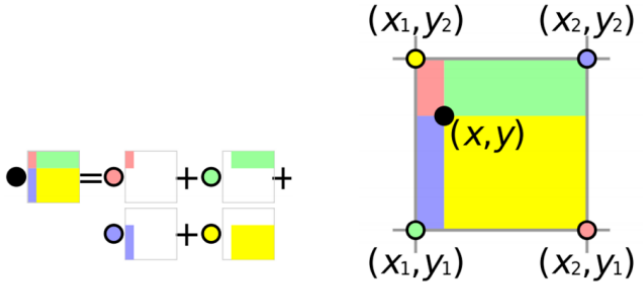
- Solves the case when dealing with non-integer pixel values.
- Uses neighbor's intensity values to calculate the desired pixels intensity.
- Nearest neighbor.
- Bilinear.
- BiCubic.

Nearest Neighbor



Bi-linear Interpolation

- Bi-linear interpolates four closest pixels.
- The weight for each pixel is proportional to its distance from the sampling point (x,y)

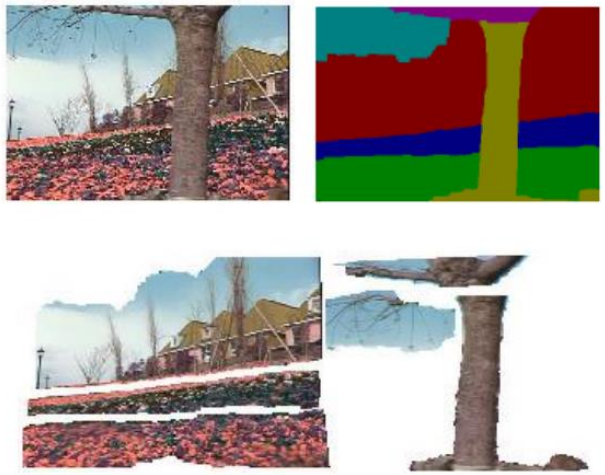


Different Approach to solve Optical Flow

LK assumes the neighborhood of each pixel moves in the same intensity, what if we could get a label image (assign each pixel to the object it belongs to).

How could we use this information to solve the optical flow problem?

Break image sequence into “layers” each of which has a coherent (affine) motion



Taken from: https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf

HW

Python code - Laplacian Pyramid Blending

- Not mandatory, won't help you much with your HW.
- But now you can implement it (or just use OpenCV implementation) ☺