Motion Estimation:

Digital Image Correlation & Optical Flow

Lecture 06

Computer Vision for Geosciences

2021-04-09

Universidad Nacional Autónoma de México

GOAL:

⇒ estimate the 2D motion projected on the image plane by the objects moving in the 3D scene

APPLICATIONS in geoscience:

⇒ capture motion, with imagery from ground based cameras, UAV, satellites, etc.

⇒ few examples:

- lava flows
- ash plumes
- dome growth
- glacier motion
- landslides
- analogue modeling
- etc.

GOAL:

⇒ estimate the 2D motion projected on the image plane by the objects moving in the 3D scene

APPLICATIONS in geoscience:

⇒ capture motion, with imagery from ground based cameras, UAV, satellites, etc.

⇒ few examples:

- lava flows
- ash plumes
- dome growth
- glacier motion
- landslides
- analogue modeling
- etc.

Methods used to estimate image motion:

1. **cross-correlation methods**

   ⇒ determine a displacement vector by maximizing the correlation peak from two successive images

   - **Digital Image Correlation** (DIC) [1,2]

     → commonly used for measuring surface deformation

   - **Particle Image Velocimetry** (PIV) [3]

     → commonly used for flow visualization, typically fluid seeded with tracer particles (experimental fluid mechanics)

     NB: PIV is very similar to DIC in principle and implementation algorithm

2. **optical flow methods** (OF)

   ⇒ originally developed by comp. vision scientists to track objects motion (e.g., people and cars) in videos [4]

   - **Sparse Optical Flow**, e.g. Lucas-Kanade algorithm [5]
   - **Dense Optical Flow**, e.g. Farnebäck algorithm [6]

[1] Peters et al. (1983) Application of digital correlation methods to rigid body mechanics Opt. Eng. 22 738–42

[2] Pan et al. (2009) Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review

[3] Adrian (1991) Particle-imaging techniques for experimental fluid mechanics. Ann Rev Fluid Mech 23:261–304

[4] Horn and Schunck (1981) Determining optical flow. Artif Intell 17:185–204

[5] Lucas and Kanade (1981) An iterative image registration technique with an application to stereo vision. Proc. of Imaging Understanding

[6] Farnebäck (2003) Two-frame motion estimation based on polynomial expansion, Proc. Scandinavian Conf. on Image Analysis

Methods used to estimate image motion:

1. **cross-correlation methods**

   ⇒ determine a displacement vector by maximizing the correlation peak from two successive images

   - **Digital Image Correlation** (DIC) [1,2]
     → commonly used for measuring surface deformation
   - **Particle Image Velocimetry** (PIV) [3]
     → commonly used for flow visualization, typically fluid seeded with tracer particles (experimental fluid mechanics)
     NB: PIV is very similar to DIC in principle and implementation algorithm

2. **optical flow methods** (OF)

   ⇒ originally developed by comp. vision scientists to track objects motion (e.g., people and cars) in videos [4]

   - **Sparse Optical Flow**, e.g. Lucas-Kanade algorithm [5]
   - **Dense Optical Flow**, e.g. Farnebäck algorithm [6]

[1] Peters et al. (1983) Application of digital correlation methods to rigid body mechanics Opt. Eng. 22 738–42

[2] Pan et al. (2009) Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review

[3] Adrian (1991) Particle-imaging techniques for experimental fluid mechanics. Ann Rev Fluid Mech 23:261–304

[4] Horn and Schunck (1981) Determining optical flow. Artif Intell 17:185–204

[5] Lucas and Kanade (1981) An iterative image registration technique with an application to stereo vision. Proc. of Imaging Understanding

[6] Farnebäck (2003) Two-frame motion estimation based on polynomial expansion, Proc. Scandinavian Conf. on Image Analysis

Methods used to estimate image motion:

1. **cross-correlation methods**

   ⇒ determine a displacement vector by maximizing the correlation peak from two successive images

   - **Digital Image Correlation** (DIC) [1,2]
     → commonly used for measuring surface deformation
   - **Particle Image Velocimetry** (PIV) [3]
     → commonly used for flow visualization, typically fluid seeded with tracer particles (experimental fluid mechanics)
     NB: PIV is very similar to DIC in principle and implementation algorithm

2. **optical flow methods** (OF)

   ⇒ originally developed by comp. vision scientists to track objects motion (e.g., people and cars) in videos [4]

   - **Sparse Optical Flow**, e.g. Lucas-Kanade algorithm [5]
   - **Dense Optical Flow**, e.g. Farnebäck algorithm [6]

---

[1] Peters et al. (1983) Application of digital correlation methods to rigid body mechanics Opt. Eng. 22 738–42

[2] Pan et al. (2009) Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review

[3] Adrian (1991) Particle-imaging techniques for experimental fluid mechanics. Ann Rev Fluid Mech 23:261–304

[4] Horn and Schunck (1981) Determining optical flow. Artif Intell 17:185–204

[5] Lucas and Kanade (1981) An iterative image registration technique with an application to stereo vision. Proc. of Imaging Understanding

[6] Farnebäck (2003) Two-frame motion estimation based on polynomial expansion, Proc. Scandinavian Conf. on Image Analysis

Methods used to estimate image motion:

1. **cross-correlation methods**

   ⇒ determine a displacement vector by maximizing the correlation peak from two successive images

   - **Digital Image Correlation** (DIC) [1,2]
     → commonly used for measuring surface deformation
   - **Particle Image Velocimetry** (PIV) [3]
     → commonly used for flow visualization, typically fluid seeded with tracer particles (experimental fluid mechanics)
     NB: PIV is very similar to DIC in principle and implementation algorithm

2. **optical flow methods** (OF)

   ⇒ originally developed by comp. vision scientists to track objects motion (e.g., people and cars) in videos [4]

   - **Sparse Optical Flow**, e.g. Lucas-Kanade algorithm [5]
   - **Dense Optical Flow**, e.g. Farnebäck algorithm [6]

---

[1] Peters et al. (1983) Application of digital correlation methods to rigid body mechanics Opt. Eng. 22 738–42

[2] Pan et al. (2009) Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review

[3] Adrian (1991) Particle-imaging techniques for experimental fluid mechanics. Ann Rev Fluid Mech 23:261–304

[4] Horn and Schunck (1981) Determining optical flow. Artif Intell 17:185–204

[5] Lucas and Kanade (1981) An iterative image registration technique with an application to stereo vision. Proc. of Imaging Understanding

[6] Farnebäck (2003) Two-frame motion estimation based on polynomial expansion, Proc. Scandinavian Conf. on Image Analysis

Cross-correlation method to estimate motion:

$\Rightarrow$ analyze the displacement within 2 images acquired at different time
$\Rightarrow$ analyze within discretized subsets (windows) of both images
$\Rightarrow$ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion
$\Rightarrow$ the maximum correlation in each window corresponds to the displacement

NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically

Cross-correlation method to estimate motion:

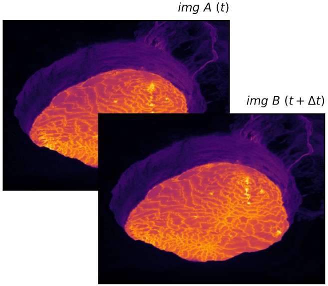
$\Rightarrow$ analyze the displacement within 2 images acquired at different time

$\Rightarrow$ analyze within discretized subsets (windows) of both images

$\Rightarrow$ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion

$\Rightarrow$ the maximum correlation in each window corresponds to the displacement

NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically



Cross-correlation method to estimate motion:

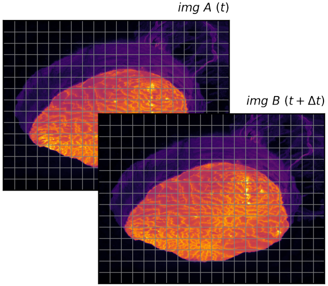
$\Rightarrow$ analyze the displacement within 2 images acquired at different time

$\Rightarrow$ analyze within discretized subsets (windows) of both images

$\Rightarrow$ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion

$\Rightarrow$ the maximum correlation in each window corresponds to the displacement

NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically



Cross-correlation method to estimate motion:

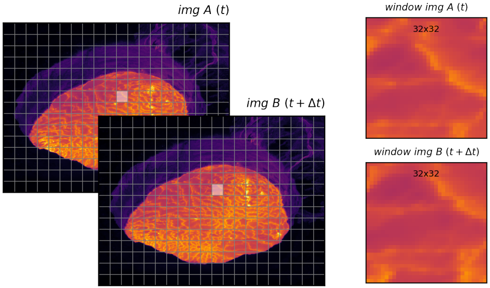
$\Rightarrow$ analyze the displacement within 2 images acquired at different time

$\Rightarrow$ analyze within discretized subsets (windows) of both images

$\Rightarrow$ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion

$\Rightarrow$ the maximum correlation in each window corresponds to the displacement

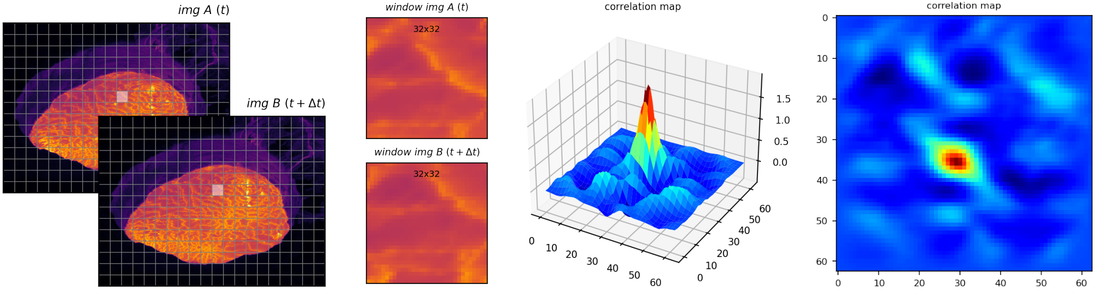
NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically



Cross-correlation method to estimate motion:

⇒ analyze the displacement within 2 images acquired at different time
⇒ analyze within discretized subsets (windows) of both images
⇒ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion
⇒ the maximum correlation in each window corresponds to the displacement

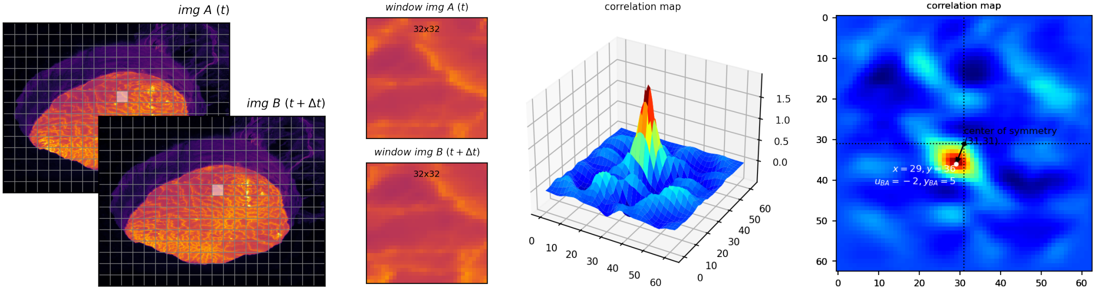
NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically



Cross-correlation method to estimate motion:

⇒ analyze the displacement within 2 images acquired at different time
⇒ analyze within discretized subsets (windows) of both images
⇒ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion
⇒ the maximum correlation in each window corresponds to the displacement

NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically



Cross-correlation method to estimate motion:

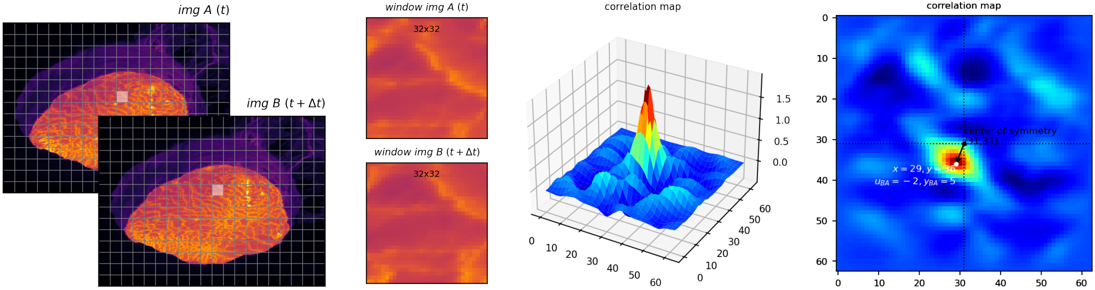
$\Rightarrow$ analyze the displacement within 2 images acquired at different time

$\Rightarrow$ analyze within discretized subsets (windows) of both images

$\Rightarrow$ evaluate similarity degree between both subsets using a cross-correlation (CC) criterion

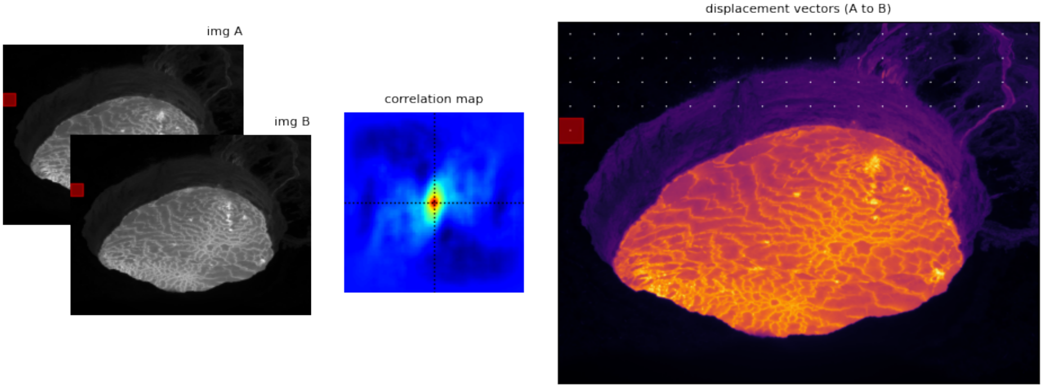
$\Rightarrow$ the maximum correlation in each window corresponds to the displacement

NB: the correlation-map is twice as big as the window sizes because windows can shift by their maximum size both horizontally and vertically



Cross-correlation method to estimate motion:

$\Rightarrow$ loop over the entire image to recover the displacements



<u>NB</u>: the above animation will only run with PDF readers having built-in JavaScript engine (ex: Adobe Reader, recent versions of Okular, etc.)

NB 1: several correlation criterion can be used to evaluate the similarity degree

NB 2: post-processing of displacement vectors allow to recover e.g. strain maps (local derivative calculation)

Table 1. Commonly used cross-correlation criterion.

CC correlation criterion	Definition
Cross-correlation (CC)	$C_{CC} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} [f(x_i, y_j) g(x'_i, y'_j)]$
Normalized cross-correlation (NCC)	$C_{NCC} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} \left[\frac{f(x_i, y_j) g(x'_i, y'_j)}{\bar{f}\,\bar{g}} \right]$
Zero-normalized cross-correlation (ZNCC)	$C_{ZNCC} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} \left\{ \frac{[f(x_i, y_j) - f_m] \times [g(x'_i, y'_j) - g_m]}{\Delta f \Delta g} \right\}$

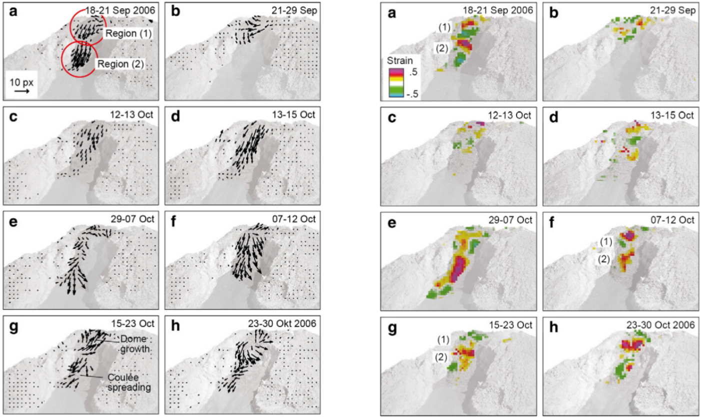
Table 2. Commonly used SSD correlation criterion.

SSD correlation criterion	Definition
Sum of squared differences (SSD)	$C_{SSD} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} [f(x_i, y_j) - g(x'_i, y'_j)]^2$
Normalized sum of squared differences (NSSD)	$C_{NSSD} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} \left[\frac{f(x_i, y_j)}{\bar{f}} - \frac{g(x'_i, y'_j)}{\bar{g}} \right]^2$
Zero-normalized sum of squared differences (ZNSSD)	$C_{ZNSSD} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} \left[\frac{f(x_i, y_j) - f_m}{\Delta f} - \frac{g(x'_i, y'_j) - g_m}{\Delta g} \right]^2$

from *Pan et al. 2009*

NB 1: several correlation criterion can be used to evaluate the similarity degree

NB 2: post-processing of displacement vectors allow to recover e.g. strain maps (local derivative calculation)



Colima volcano dome growth and coulée spreading (*Walter et al. 2013*)

(compression=green / extension=red)

Optical-flow method to estimate motion:

⇒ the most general version of motion estimation is to compute an independent estimate of motion at each pixel → generally known as **optical flow** *(Szeliski 2010)*[1]

⇒ in contrast to the correlation method that is essentially an integral approach, the optical flow method is a differential approach (hence better suited for to images with continuous patterns) *(Liu et al. 2015)*[2]

⇒ *Horn and Schunck (1981)* gave the first optical flow equation (a.k.a. the brightness constraint equation)

⇒ the most famous algorithms developped to solve the optical flow equation are:

- *Lucas and Kanade (1981)*: **sparse** optical flow (Lucas-Kanade, 1981)
  ⇒ displacement vectors computed for "best-suited" image regions: corners & edges (good features!)
- *Farneback, 2003*: **dense** optical flow
  ⇒ displacement vectors computed for every pixel in the image

[1] Szeliski (2010) Computer Vision: Algorithms and Applications, Springer editions

[2] Liu et al. (2015) Comparison between optical flow and cross-correlation methods for extraction of velocity fields from particle images, Exp. Fluids

Optical-flow method to estimate motion:

⇒ the most general version of motion estimation is to compute an independent estimate of motion at each pixel → generally known as **optical flow** *(Szeliski 2010)*[1]

⇒ in contrast to the correlation method that is essentially an integral approach, the optical flow method is a differential approach (hence better suited for to images with continuous patterns) *(Liu et al. 2015)*[2]

⇒ *Horn and Schunck (1981)* gave the first optical flow equation (a.k.a. the brightness constraint equation)

⇒ the most famous algorithms developped to solve the optical flow equation are:

- *Lucas and Kanade (1981)*: **sparse** optical flow (Lucas-Kanade, 1981)
  ⇒ displacement vectors computed for "best-suited" image regions: corners & edges (good features!)
- *Farneback, 2003*: **dense** optical flow
  ⇒ displacement vectors computed for every pixel in the image

[1] Szeliski (2010) Computer Vision: Algorithms and Applications, Springer editions

[2] Liu et al. (2015) Comparison between optical flow and cross-correlation methods for extraction of velocity fields from particle images, Exp. Fluids

Optical-flow method to estimate motion:

⇒ the most general version of motion estimation is to compute an independent estimate of motion at each pixel → generally known as **optical flow** *(Szeliski 2010)*[1]

⇒ in contrast to the correlation method that is essentially an integral approach, the optical flow method is a differential approach (hence better suited for to images with continuous patterns) *(Liu et al. 2015)*[2]

⇒ *Horn and Schunck (1981)* gave the first optical flow equation (a.k.a. the brightness constraint equation)

⇒ the most famous algorithms developped to solve the optical flow equation are:

- *Lucas and Kanade (1981)*: **sparse** optical flow (Lucas-Kanade, 1981)
  ⇒ displacement vectors computed for "best-suited" image regions: corners & edges (good features!)
- *Farneback, 2003*: **dense** optical flow
  ⇒ displacement vectors computed for every pixel in the image

[1] Szeliski (2010) Computer Vision: Algorithms and Applications, Springer editions

[2] Liu et al. (2015) Comparison between optical flow and cross-correlation methods for extraction of velocity fields from particle images, Exp. Fluids

Optical-flow method to estimate motion:

$\Rightarrow$ the most general version of motion estimation is to compute an independent estimate of motion at each pixel $\rightarrow$ generally known as **optical flow** *(Szeliski 2010)*[1]

$\Rightarrow$ in contrast to the correlation method that is essentially an integral approach, the optical flow method is a differential approach (hence better suited for to images with continuous patterns) *(Liu et al. 2015)*[2]

$\Rightarrow$ *Horn and Schunck (1981)* gave the first optical flow equation (a.k.a. the brightness constraint equation)

$\Rightarrow$ the most famous algorithms developped to solve the optical flow equation are:

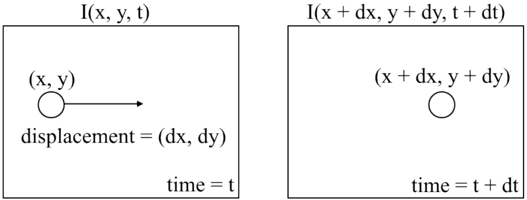
- *Lucas and Kanade (1981)*: **sparse** optical flow (Lucas-Kanade, 1981)
  $\Rightarrow$ displacement vectors computed for "best-suited" image regions: corners & edges (good features!)
- *Farneback, 2003*: **dense** optical flow
  $\Rightarrow$ displacement vectors computed for every pixel in the image

[1] Szeliski (2010) Computer Vision: Algorithms and Applications, Springer editions

[2] Liu et al. (2015) Comparison between optical flow and cross-correlation methods for extraction of velocity fields from particle images, Exp. Fluids

How is the optical flow equation *obtained* ? *(Horn & Schunck, 1981)*

1. **Define the optical flow problem**
   $\Rightarrow$ optical flow = motion of objects between consecutive frames
   $\Rightarrow$ how can we recover displacements dx and dy?



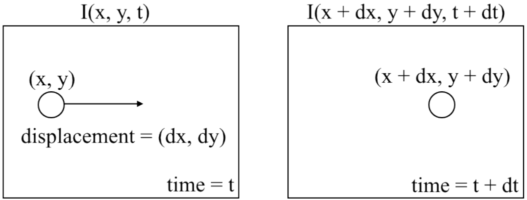
2. **Brightness constancy assumption**
   $\Rightarrow$ assume that pixel intensities are constant between consecutive frames

NB: this assumption is valid for small time difference between frames (dt), and for pixels in a small region (small dx, dt)

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{1}$$

How is the optical flow equation *obtained* ? *(Horn & Schunck, 1981)*

1. **Define the optical flow problem**
   ⇒ optical flow = motion of objects between consecutive frames
   ⇒ how can we recover displacements dx and dy?



2. **Brightness constancy assumption**
   ⇒ assume that pixel intensities are constant between consecutive frames
   NB: this assumption is valid for small time difference between frames (dt), and for pixels in a small region (small dx, dt)

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{1}$$

2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials
$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$

2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials

$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$

2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials
$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

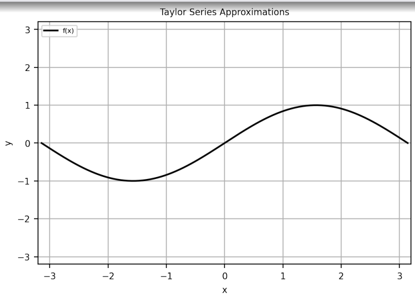
$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$

2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials
$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$



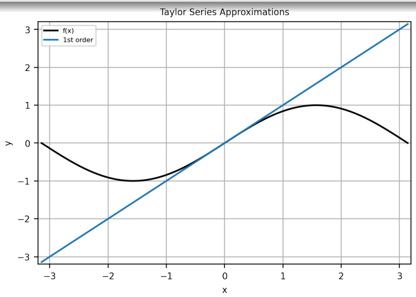
2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials

$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$



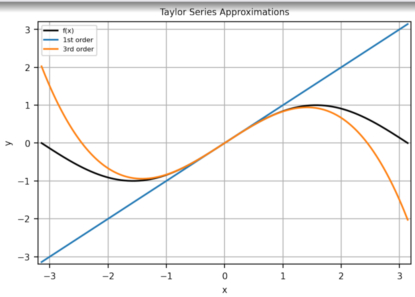
2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials

$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$



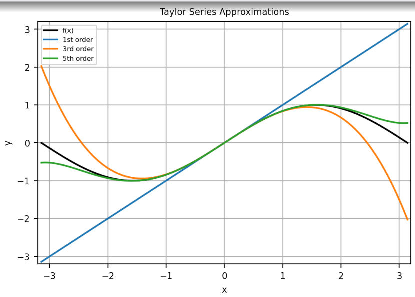
2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

Reminder

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials

$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$



2. **Taylor Series Approximation of the right-hand side**
   $\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

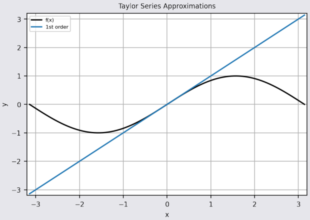
Reminder (continued)

$\Rightarrow$ the Taylor series is an extremely powerful tool for approximating functions as polynomials
$\Rightarrow$ the Taylor series of a function $f(x)$ is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point (wikipedia)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n$$

$\Rightarrow$ EX: 1st order Taylor approximation of an image profile $I(x)$, centered around x=0 (a=0):

$$\begin{aligned} I(x) &\approx I(a) + I'(a)(x-a) \\ &\approx I(a) + \frac{d}{dx}I(a)(x-a) \\ &\approx I(0) + \frac{d}{dx}I(0)x \\ &\approx b + ax \end{aligned}$$



2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{1}$$

Recall 1st order Taylor general approximation:

$$f(x) \approx f(a) + f'(a)(x - a)$$

The right-hand side can therefore be approximated as:

$$\begin{aligned} I(x + dx, y + dy, t + dt) &\approx I(x, y, t) + \frac{\partial I}{\partial x}(x + dx - x) + \frac{\partial I}{\partial y}(y + dy - y) + \frac{\partial I}{\partial t}(t + dt - t) \\ &\approx I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \end{aligned}$$

Replacing the approximation inside equation (1), and canceling out the $I(x, y, t)$ term on both sides gives:

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0 \tag{2}$$

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

$$I(x, y, t) = \boxed{I(x + dx, y + dy, t + dt)} \tag{1}$$

Recall 1st order Taylor general approximation:

$$f(x) \approx f(a) + f'(a)(x - a)$$

The right-hand side can therefore be approximated as:

$$\begin{aligned} I(x + dx, y + dy, t + dt) &\approx I(x, y, t) + \frac{\partial I}{\partial x}(x + dx - x) + \frac{\partial I}{\partial y}(y + dy - y) + \frac{\partial I}{\partial t}(t + dt - t) \\ &\approx I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \end{aligned}$$

Replacing the approximation inside equation (1), and canceling out the $I(x, y, t)$ term on both sides gives:

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0 \tag{2}$$

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

$$I(x, y, t) = \boxed{I(x + dx, y + dy, t + dt)} \tag{1}$$

Recall 1st order Taylor general approximation:

$$f(x) \approx f(a) + f'(a)(x - a)$$

The right-hand side can therefore be approximated as:

$$\begin{aligned} \boxed{I(x + dx, y + dy, t + dt)} &\approx I(x, y, t) + \frac{\partial I}{\partial x}(x + dx - x) + \frac{\partial I}{\partial y}(y + dy - y) + \frac{\partial I}{\partial t}(t + dt - t) \\ &\approx I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \end{aligned}$$

Replacing the approximation inside equation (1), and canceling out the $I(x, y, t)$ term on both sides gives:

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0 \tag{2}$$

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ approximate the right-hand side of equation (1) with the 1st order Taylor series

$$I(x, y, t) = \boxed{I(x + dx, y + dy, t + dt)} \quad (1)$$

Recall 1st order Taylor general approximation:

$$f(x) \approx f(a) + f'(a)(x - a)$$

The right-hand side can therefore be approximated as:

$$\begin{aligned} \boxed{I(x + dx, y + dy, t + dt)} &\approx I(x, y, t) + \frac{\partial I}{\partial x}(x + dx - x) + \frac{\partial I}{\partial y}(y + dy - y) + \frac{\partial I}{\partial t}(t + dt - t) \\ &\approx I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \end{aligned}$$

Replacing the approximation inside equation (1), and canceling out the $I(x, y, t)$ term on both sides gives:

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0 \quad (2)$$

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ dividing equation (2) by dt gives:

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

where:

- $\frac{dx}{dt} = u$ and $\frac{dy}{dt} = v$ are the **displacement vectors**
- $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the **image gradients** along the horizontal axis, the vertical axis, and time

$\Rightarrow$ the **optical flow equation** is therefore defined as :

$$\boxed{\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0} \tag{3}$$

1 equation, 2 unknowns! $\Rightarrow$ underdetermined

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ dividing equation (2) by dt gives:

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

where:

- $\frac{dx}{dt} = u$ and $\frac{dy}{dt} = v$ are the **displacement vectors**
- $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the **image gradients** along the horizontal axis, the vertical axis, and time

$\Rightarrow$ the **optical flow equation** is therefore defined as :

$$\boxed{\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0} \tag{3}$$

1 equation, 2 unknowns! $\Rightarrow$ underdetermined

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ dividing equation (2) by dt gives:

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

where:

- $\frac{dx}{dt} = u$ and $\frac{dy}{dt} = v$ are the **displacement vectors**
- $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the **image gradients** along the horizontal axis, the vertical axis, and time

$\Rightarrow$ the **optical flow equation** is therefore defined as :

$$\boxed{\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0} \qquad (3)$$

1 equation, 2 unknowns! $\Rightarrow$ underdetermined

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ dividing equation (2) by dt gives:

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

where:

- $\frac{dx}{dt} = u$ and $\frac{dy}{dt} = v$ are the **displacement vectors**
- $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the **image gradients** along the horizontal axis, the vertical axis, and time

$\Rightarrow$ the **optical flow equation** is therefore defined as :

$$\boxed{\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0} \tag{3}$$

1 equation, 2 unknowns! $\Rightarrow$ underdetermined

2. **Taylor Series Approximation of the right-hand side**

$\Rightarrow$ dividing equation (2) by dt gives:

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

where:

- $\frac{dx}{dt} = u$ and $\frac{dy}{dt} = v$ are the **displacement vectors**
- $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the **image gradients** along the horizontal axis, the vertical axis, and time

$\Rightarrow$ the **optical flow equation** is therefore defined as :

$$\boxed{\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0} \qquad (3)$$

1 equation, 2 unknowns! $\Rightarrow$ underdetermined

How is the optical flow equation *solved* ?

⇒ most famous approach is the **Lucas & Kanade, 1981** method

→ the method assumes that pixels in a small neighbood have similar motion, hence a 3x3 window around the central pixel gives 9 optical flow equations

To simplify the reading, let's rename the variables in the optical flow equation:

$$\frac{\partial I}{\partial x} = dI_x (=\text{image horizontal gradient, compute with convolution kernel!})$$
$$\frac{\partial I}{\partial y} = dI_y (=\text{image vertical gradient, compute with convolution kernel!})$$
$$\frac{\partial I}{\partial t} = dI_t = I_t[x,y] - I_{t+dt}[x,y]$$

→ the 9 optical flow equations can therefore be expressed as a system of equations:

$$\begin{cases} dI_{x_1} u + dI_{y_1} v & = -dI_{t_1} \\ \vdots \quad\quad \vdots & = \quad \vdots \\ dI_{x_9} u + dI_{y_9} v & = -dI_{t_9} \end{cases}$$

How is the optical flow equation *solved* ?

$\Rightarrow$ most famous approach is the **Lucas & Kanade, 1981** method

$\rightarrow$ the method assumes that pixels in a small neighbood have similar motion, hence a 3x3 window around the central pixel gives 9 optical flow equations

To simplify the reading, let's rename the variables in the optical flow equation:

$$\frac{\partial I}{\partial x} = dI_x (\text{=image horizontal gradient, compute with convolution kernel!})$$

$$\frac{\partial I}{\partial y} = dI_y (\text{=image vertical gradient, compute with convolution kernel!})$$

$$\frac{\partial I}{\partial t} = dI_t = I_t[x, y] - I_{t+dt}[x, y]$$

$\rightarrow$ the 9 optical flow equations can therefore be expressed as a system of equations:

$$\begin{cases} dI_{x_1} u + dI_{y_1} v & = -dI_{t_1} \\ \vdots \qquad \vdots & = \vdots \\ dI_{x_9} u + dI_{y_9} v & = -dI_{t_9} \end{cases}$$

How is the optical flow equation *solved* ?

$\Rightarrow$ most famous approach is the **Lucas & Kanade, 1981** method

$\rightarrow$ the method assumes that pixels in a small neighbood have similar motion, hence a 3x3 window around the central pixel gives 9 optical flow equations

To simplify the reading, let's rename the variables in the optical flow equation:

$$\frac{\partial I}{\partial x} = dI_x (=\text{image horizontal gradient, compute with convolution kernel!})$$
$$\frac{\partial I}{\partial y} = dI_y (=\text{image vertical gradient, compute with convolution kernel!})$$
$$\frac{\partial I}{\partial t} = dI_t = I_t[x, y] - I_{t+dt}[x, y]$$

$\rightarrow$ the 9 optical flow equations can therefore be expressed as a system of equations:

$$\begin{cases} dI_{x_1} u + dI_{y_1} v & = -dI_{t_1} \\ \vdots \quad\quad \vdots & = \vdots \\ dI_{x_9} u + dI_{y_9} v & = -dI_{t_9} \end{cases}$$

How is the optical flow equation *solved* ?

$\Rightarrow$ most famous approach is the **Lucas & Kanade, 1981** method

$\rightarrow$ the method assumes that pixels in a small neighbood have similar motion, hence a 3x3 window around the central pixel gives 9 optical flow equations

To simplify the reading, let's rename the variables in the optical flow equation:

$$\frac{\partial I}{\partial x} = dI_x \text{(=image horizontal gradient, compute with convolution kernel!)}$$
$$\frac{\partial I}{\partial y} = dI_y \text{(=image vertical gradient, compute with convolution kernel!)}$$
$$\frac{\partial I}{\partial t} = dI_t = I_t[x, y] - I_{t+dt}[x, y]$$

$\rightarrow$ the 9 optical flow equations can therefore be expressed as a system of equations:

$$\begin{cases} dI_{x_1} u + dI_{y_1} v & = -dI_{t_1} \\ \vdots \quad\quad \vdots & = \quad \vdots \\ dI_{x_9} u + dI_{y_9} v & = -dI_{t_9} \end{cases}$$

How is the optical flow equation *solved* ?

$\Rightarrow$ most famous approach is the **Lucas & Kanade, 1981** method

$\rightarrow$ the method assumes that pixels in a small neighbood have similar motion, hence a 3x3 window around the central pixel gives 9 optical flow equations

To simplify the reading, let's rename the variables in the optical flow equation:

$$\frac{\partial I}{\partial x} = dI_x \text{(=image horizontal gradient, compute with convolution kernel!)}$$

$$\frac{\partial I}{\partial y} = dI_y \text{(=image vertical gradient, compute with convolution kernel!)}$$

$$\frac{\partial I}{\partial t} = dI_t = I_t[x, y] - I_{t+dt}[x, y]$$

$\rightarrow$ the 9 optical flow equations can therefore be expressed as a system of equations:

$$\begin{cases} dI_{x_1} u + dI_{y_1} v & = -dI_{t_1} \\ \quad \vdots \qquad\quad \vdots & = \quad \vdots \\ dI_{x_9} u + dI_{y_9} v & = -dI_{t_9} \end{cases}$$

Lucas & Kanade method (continued)

→ the system of equations can be written in matrix form:

$$\boxed{A\nu = b} \tag{4}$$

with: $A = \begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ f_{x9} & f_{y9} \end{bmatrix}$, $\nu = \begin{bmatrix} u \\ v \end{bmatrix}$, and $b = \begin{bmatrix} -f_{t1} \\ \vdots \\ -f_{t9} \end{bmatrix}$

⇒ the Lucas-Kanade algorithm solves for $\nu = [u, v]$ by minimizing the sum-squared error of the optical flow equations for each pixel in the chosen window (least square fit)

NB: A is not square hence not inversable ⇒ trick is to multiply by its transform to make it square (⇒ inversable):

$$\begin{aligned} A\nu &= b \\ A^T A\nu &= A^T b \\ (A^T A)^{-1}(A^T A)\nu &= (A^T A)^{-1} A^T b \\ \nu &= (A^T A)^{-1} A^T b \end{aligned}$$

Lucas & Kanade method (continued)

→ the system of equations can be written in matrix form:

$$\boxed{A\nu = b} \tag{4}$$

with: $A = \begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ f_{x9} & f_{y9} \end{bmatrix}$, $\nu = \begin{bmatrix} u \\ v \end{bmatrix}$, and $b = \begin{bmatrix} -f_{t1} \\ \vdots \\ -f_{t9} \end{bmatrix}$

⇒ the Lucas-Kanade algorithm solves for $\nu = [u, v]$ by minimizing the sum-squared error of the optical flow equations for each pixel in the chosen window (least square fit)

NB: A is not square hence not inversable ⇒ trick is to multiply by its transform to make it square (⇒ inversable):

$$A\nu = b$$
$$A^T A\nu = A^T b$$
$$(A^T A)^{-1}(A^T A)\nu = (A^T A)^{-1} A^T b$$
$$\nu = (A^T A)^{-1} A^T b$$

Lucas & Kanade method (continued)

$\rightarrow$ the system of equations can be written in matrix form:

$$\boxed{A\nu = b} \tag{4}$$

with: $A = \begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ f_{x9} & f_{y9} \end{bmatrix}$, $\nu = \begin{bmatrix} u \\ v \end{bmatrix}$, and $b = \begin{bmatrix} -f_{t1} \\ \vdots \\ -f_{t9} \end{bmatrix}$

$\Rightarrow$ the Lucas-Kanade algorithm solves for $\nu = [u, v]$ by minimizing the sum-squared error of the optical flow equations for each pixel in the chosen window (least square fit)

NB: A is not square hence not inversable $\Rightarrow$ trick is to multiply by its transform to make it square ($\Rightarrow$ inversable):

$$\begin{aligned} A\nu &= b \\ A^T A\nu &= A^T b \\ (A^T A)^{-1}(A^T A)\nu &= (A^T A)^{-1} A^T b \\ \nu &= (A^T A)^{-1} A^T b \end{aligned}$$

Lucas & Kanade method (continued)

$\rightarrow$ the system of equations can be written in matrix form:

$$\boxed{A\nu = b} \tag{4}$$

with: $A = \begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ f_{x9} & f_{y9} \end{bmatrix}$, $\nu = \begin{bmatrix} u \\ v \end{bmatrix}$, and $b = \begin{bmatrix} -f_{t1} \\ \vdots \\ -f_{t9} \end{bmatrix}$

$\Rightarrow$ the Lucas-Kanade algorithm solves for $\nu = [u, v]$ by minimizing the sum-squared error of the optical flow equations for each pixel in the chosen window (least square fit)

NB: A is not square hence not inversable $\Rightarrow$ trick is to multiply by its transform to make it square ($\Rightarrow$ inversable):

$$\begin{aligned} A\nu &= b \\ A^T A\nu &= A^T b \\ (A^T A)^{-1}(A^T A)\nu &= (A^T A)^{-1} A^T b \\ \boxed{\nu = (A^T A)^{-1} A^T b} \end{aligned}$$

Lucas & Kanade method (continued)

Beware, $A^T A$ only invertable where eigen values λ_1 and $\lambda_2 > 0$:

$\Rightarrow$ inversable where image has "texture"
$\Rightarrow$ compute only for good **features points**, e.g. edges and corners ! (e.g. Harris corners, Shi-Tomasi corners, ...)

Lucas & Kanade method (continued)

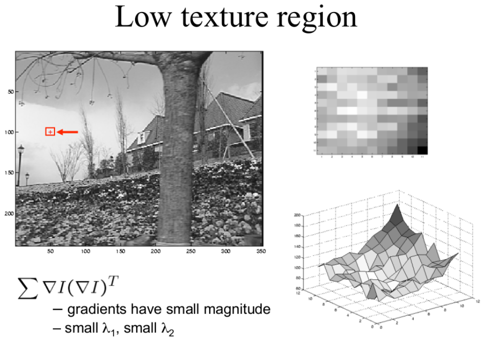
Beware, $A^T A$ only invertable where eigen values λ_1 and $\lambda_2 > 0$:

$\Rightarrow$ inversable where image has "texture"
$\Rightarrow$ compute only for good **features points**, e.g. edges and corners ! (e.g. Harris corners, Shi-Tomasi corners, ...)

Lucas & Kanade method (continued)

Beware, $A^T A$ only invertable where eigen values λ_1 and $\lambda_2 > 0$:

$\Rightarrow$ inversable where image has "texture"
$\Rightarrow$ compute only for good **features points**, e.g. edges and corners ! (e.g. Harris corners, Shi-Tomasi corners, ...)

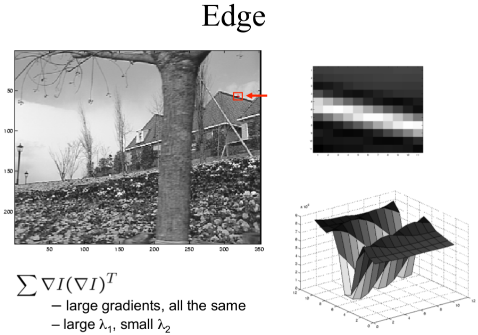


bad!

Lucas & Kanade method (continued)

Beware, $A^T A$ only invertable where eigen values λ_1 and $\lambda_2 > 0$:

$\Rightarrow$ inversable where image has "texture"
$\Rightarrow$ compute only for good **features points**, e.g. edges and corners ! (e.g. Harris corners, Shi-Tomasi corners, ...)

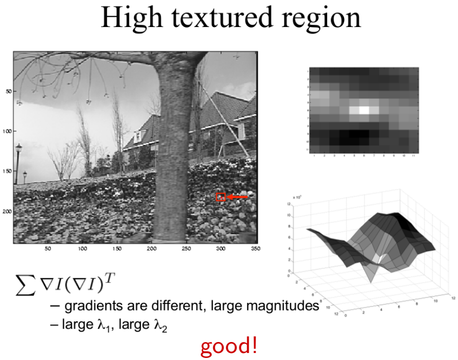


pretty good

Lucas & Kanade method (continued)

Beware, $A^T A$ only invertable where eigen values λ_1 and $\lambda_2 > 0$:

$\Rightarrow$ inversable where image has "texture"
$\Rightarrow$ compute only for good **features points**, e.g. edges and corners ! (e.g. Harris corners, Shi-Tomasi corners, ...)



Demonstration:

1. **Sparse Optical Flow** (Lucas-Kanade algorithm)

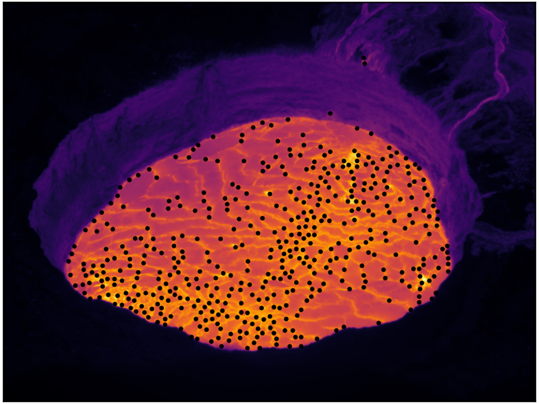
   $\Rightarrow$ computes flow only for specific features (ex: Shi-Tomasi corners), i.e. sparse

Demonstration:

1. **Sparse Optical Flow** (Lucas-Kanade algorithm)

   $\Rightarrow$ computes flow only for specific features (ex: Shi-Tomasi corners), i.e. sparse

shitomasi corners

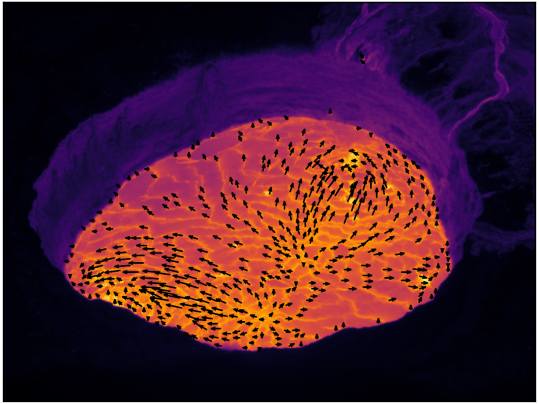


Demonstration:

1. **Sparse Optical Flow** (Lucas-Kanade algorithm)

   $\Rightarrow$ computes flow only for specific features (ex: Shi-Tomasi corners), i.e. sparse

displacement vectors



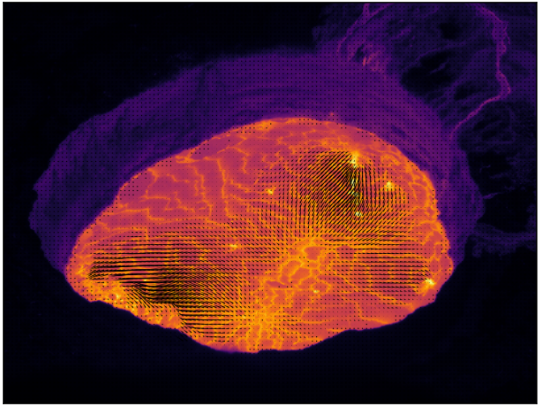
Demonstration:

2. **Dense Optical Flow** (Farnebäck algorithm)

⇒ computes flow for all pixels, i.e. dense

⇒ approximation uses a second-order Taylor Expansion

sampling step 5



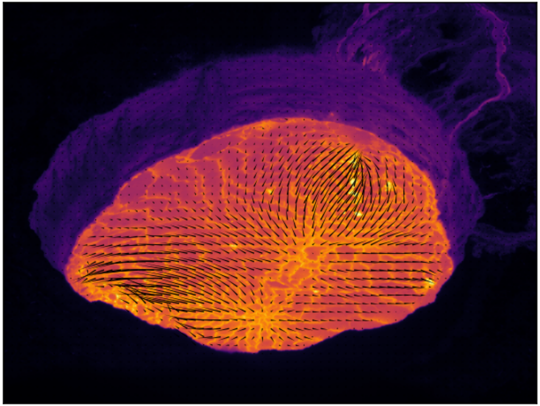
Demonstration:

2. **Dense Optical Flow** (Farnebäck algorithm)

   ⇒ computes flow for all pixels, i.e. dense

   ⇒ approximation uses a second-order Taylor Expansion

sampling step 10



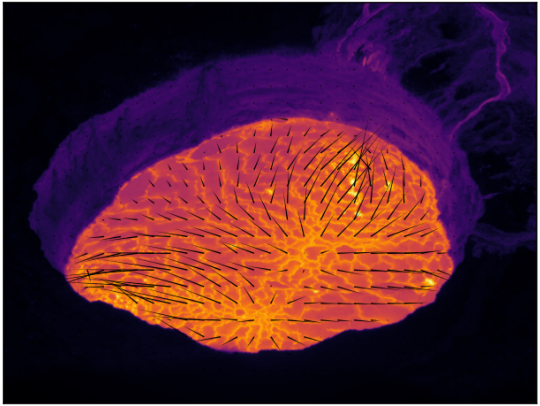
Demonstration:

2. **Dense Optical Flow** (Farnebäck algorithm)

   $\Rightarrow$ computes flow for all pixels, i.e. dense

   $\Rightarrow$ approximation uses a second-order Taylor Expansion

sampling step 20



OpenCV (Open Source Computer Vision Library):

⇒ library of programming functions mainly aimed at real-time computer vision

⇒ written in C++ (primary interface), APIs in Python, Java, and Matlab

Installing OpenCV with Anaconda (conda-forge packages):

```
$ conda install -c conda-forge opencv
```

Nota Bene

If the above command hangs or fails with error message "`Solving environment: failed with initial frozen solve. Retrying with flexible solve`", it is likely that there is dependency clash in the default conda environment.

⇒ *Solution 1 (quick & dirty): update all packages and retry*

```
$ conda update --all
$ conda install -c conda-forge opencv
```

⇒ *Solution 2 (clean): create a separate environment where OpenCV is to be installed*

```
$ conda create --name <name>
$ activate <name>
$ conda install -c conda-forge opencv
```

OpenCV (Open Source Computer Vision Library):

⇒ library of programming functions mainly aimed at real-time computer vision

⇒ written in C++ (primary interface), APIs in Python, Java, and Matlab

Installing OpenCV with Anaconda (conda-forge packages):

```
$ conda install -c conda-forge opencv
```

Nota Bene

If the above command hangs or fails with error message "`Solving environment: failed with initial frozen solve. Retrying with flexible solve`", it is likely that there is dependency clash in the default conda environment.

⇒ *Solution 1 (quick & dirty): update all packages and retry*

```
$ conda update --all
$ conda install -c conda-forge opencv
```

⇒ *Solution 2 (clean): create a separate environment where OpenCV is to be installed*

```
$ conda create --name <name>
$ activate <name>
$ conda install -c conda-forge opencv
```

OpenCV (Open Source Computer Vision Library):

⇒ library of programming functions mainly aimed at real-time computer vision

⇒ written in C++ (primary interface), APIs in Python, Java, and Matlab

Installing OpenCV with Anaconda (conda-forge packages):

```
$ conda install -c conda-forge opencv
```

Nota Bene

If the above command hangs or fails with error message "`Solving environment: failed with initial frozen solve. Retrying with flexible solve`", it is likely that there is dependency clash in the default conda environment.

⇒ *Solution 1 (quick & dirty): update all packages and retry*

```
$ conda update --all
$ conda install -c conda-forge opencv
```

⇒ *Solution 2 (clean): create a separate environment where OpenCV is to be installed*

```
$ conda create --name <name>
$ activate <name>
$ conda install -c conda-forge opencv
```