CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Machine Vision

Lecture 6: Photometric Stereo & Optical Flow

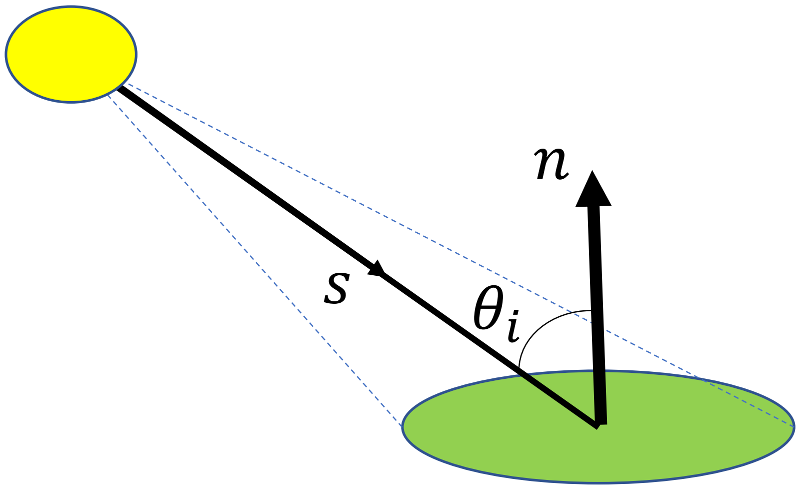
Photometric stereo

- What information about the 3D structure of an object can we derive from its shading?

Irradiance

- The **irradiance** is the amount of light falling on a surface patch per unit area (Wm^{-2})
- For a point light source with intensity I the irradiance on the surface patch depends on the angle of incident light

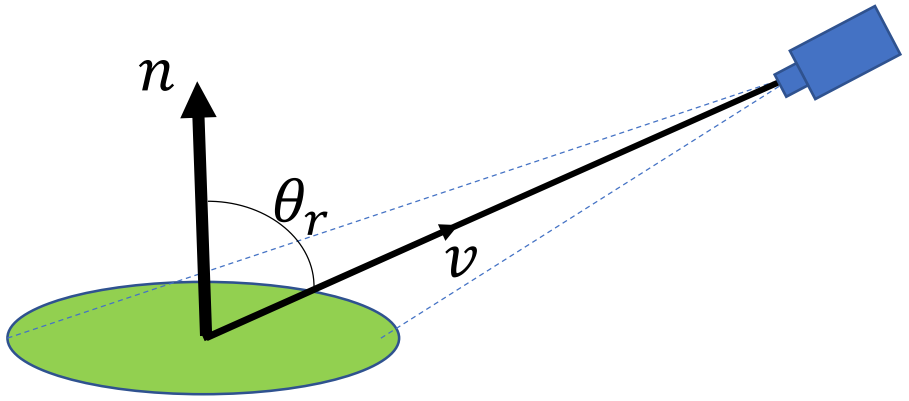
$$\delta E = I \cos \theta_i$$



Radiance

- The **radiance** is the amount of light emitted from a surface patch per unit area per unit angle ($Wm^{-2}sr^{-1}$)
- The **brightness** measured by the camera in every pixel is proportional to the surface radiance

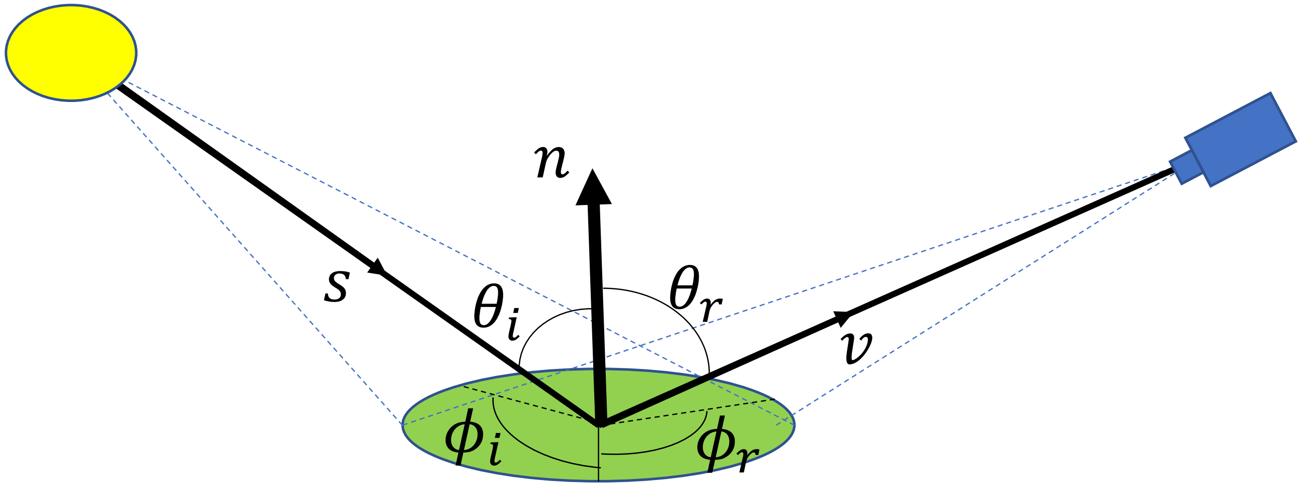
$$\delta B \propto \delta L$$



BDRF

- The ratio between these depends on the material and how it reflects/absorbs light at different angles, which is characterised by the **bi-directional reflectance distribution function** (BDRF):

$$\frac{\delta L}{\delta E} = f[\theta_i, \phi_i, \theta_r, \phi_r]$$



Diffuse reflection

- A **Lambertian surface** reflects light in all directions equally
- The BDRF in this case is constant regardless of the angles

$$f[\theta_i, \phi_i, \theta_r, \phi_r] = \frac{\rho_d}{\pi}$$

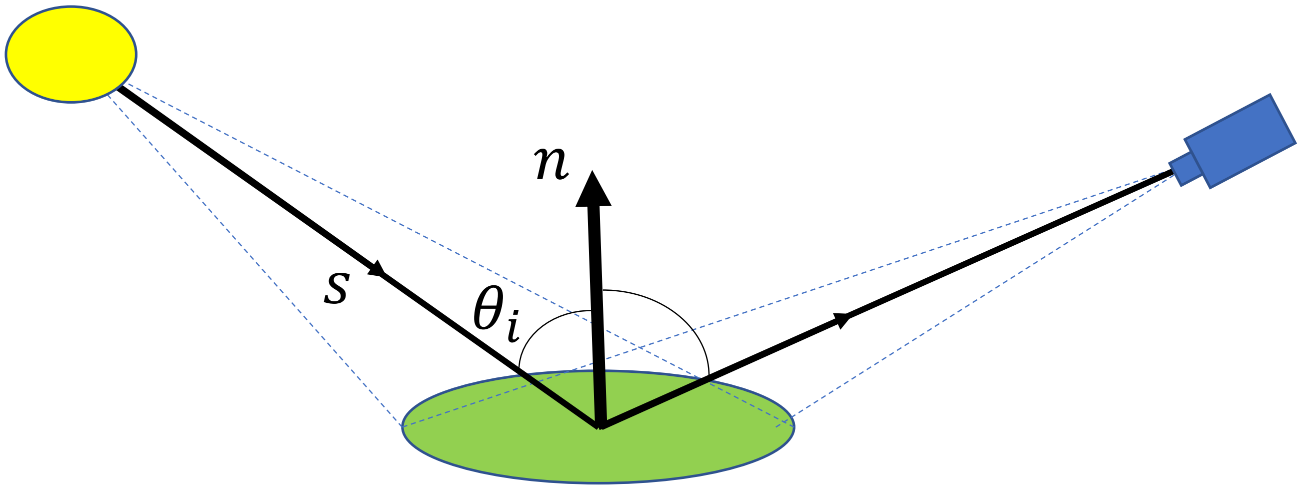
- The material constant ρ_d is called the **albedo**, which is the fraction of light not absorbed by the surface (0=black, 1=white)

Cosine law

- Putting it all together, a **Lambertian surface** illuminated by a point light source of intensity I has a surface radiance of

$$\delta L = \frac{\rho_d}{\pi} I \cos\theta_i = \frac{\rho_d}{\pi} I \, \boldsymbol{s}^T \boldsymbol{n}$$

- Pixel brightness in this case is therefore proportional to the scalar product of the surface normal and the direction of incident light



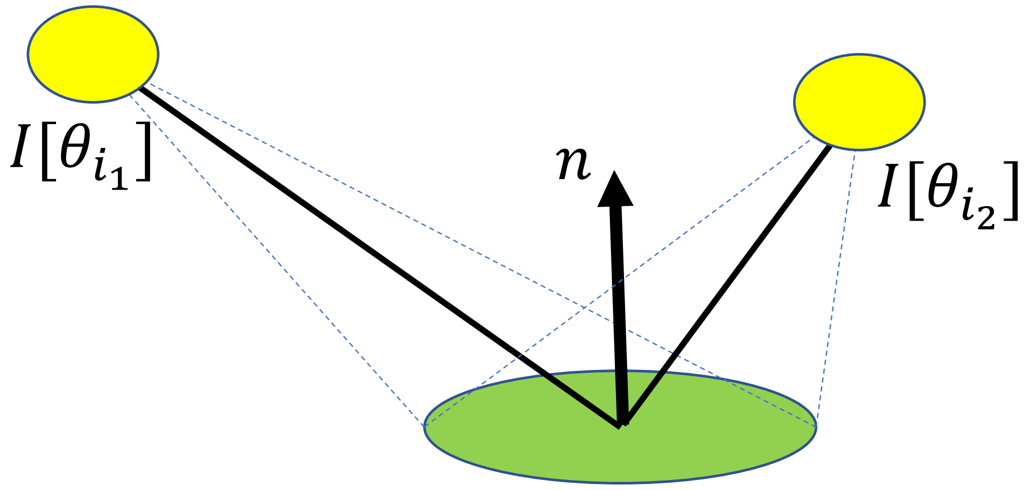
Multiple light sources

- If there are multiple light sources we can consider this by using an incident light angle dependent intensity function

$$I[\theta_i, \phi_i]$$

- The irradiance on a surface patch is in this case aggregated over all angles of the hemisphere

$$\delta E = \int_{-pi}^{pi} \int_{0}^{\frac{pi}{2}} I[\theta_i, \phi_i] \sin \phi_i \cos \theta_i \, d\theta_i d\phi_i$$



Diffuse light

- In **diffuse lighting conditions** (clouds) the intensity is independent of the angle
- If we now also assume a Lambertian surface, then the scene radiance is constant everywhere independent of surface orientation

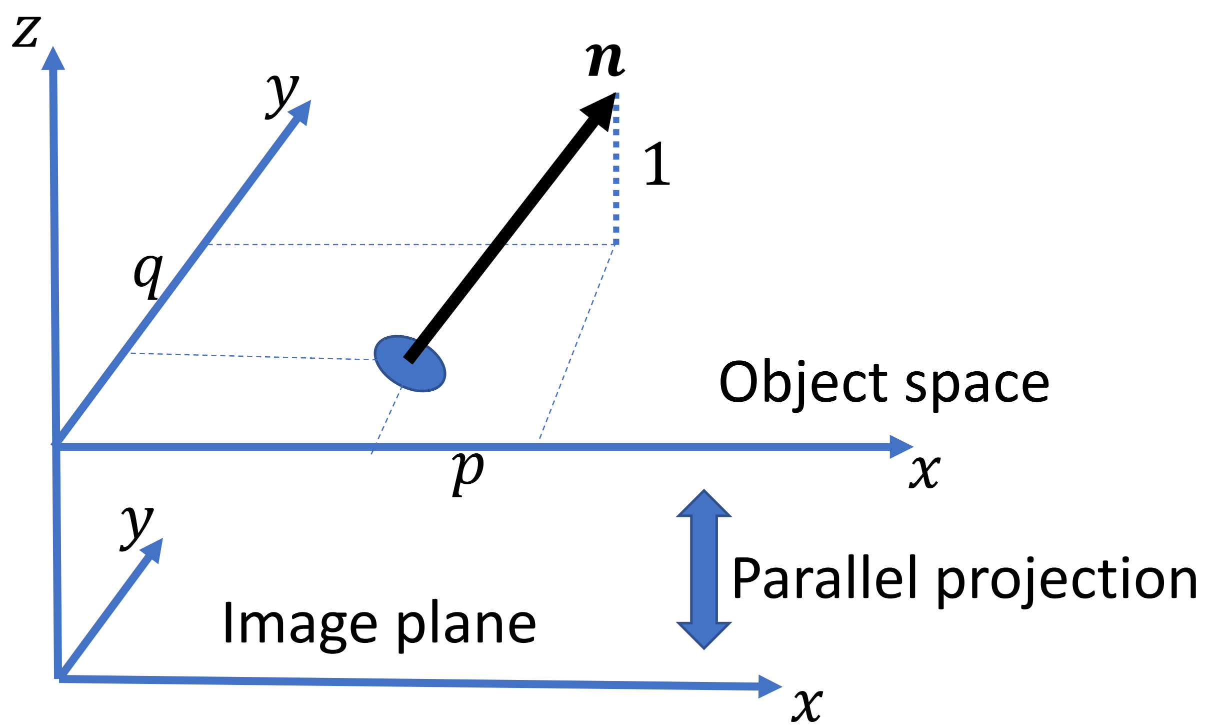
$$\delta L = \int_{-pi}^{pi} \int_{0}^{\frac{pi}{2}} \frac{I}{\pi} \sin\phi_i \cos\theta_i \, d\theta_i d\phi_i = I$$

- This conditions are causing the "white out" effect experienced when skiing in cloudy weather conditions

Reflectance map

- Instead of working with angles and polar coordinates it is useful to work with the homogeneous coordinates of the plane defined by the normal vector

$$\frac{\boldsymbol{n}}{|\boldsymbol{n}|} = \frac{1}{\sqrt{1+p^2+q^2}} \begin{pmatrix} p \\ q \\ 1 \end{pmatrix}$$



Reflectance map

- Instead of working with angles and polar coordinates it is useful to work with the homogeneous coordinates of the plane defined by the normal vector

$$\frac{\boldsymbol{n}}{|\boldsymbol{n}|} = \frac{1}{\sqrt{1+p^2+q^2}}\begin{pmatrix} p \\ q \\ 1 \end{pmatrix}$$

- We can also encode the direction of the light source in these coordinates

$$\frac{\boldsymbol{s}}{|\boldsymbol{s}|} = \frac{1}{\sqrt{1+p_s^2+q_s^2}}\begin{pmatrix} p_s \\ q_s \\ 1 \end{pmatrix}$$

- The angle of incident light is then

$$\cos\theta_i = \frac{pp_s + qq_s + 1}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}}$$

Reflectance map

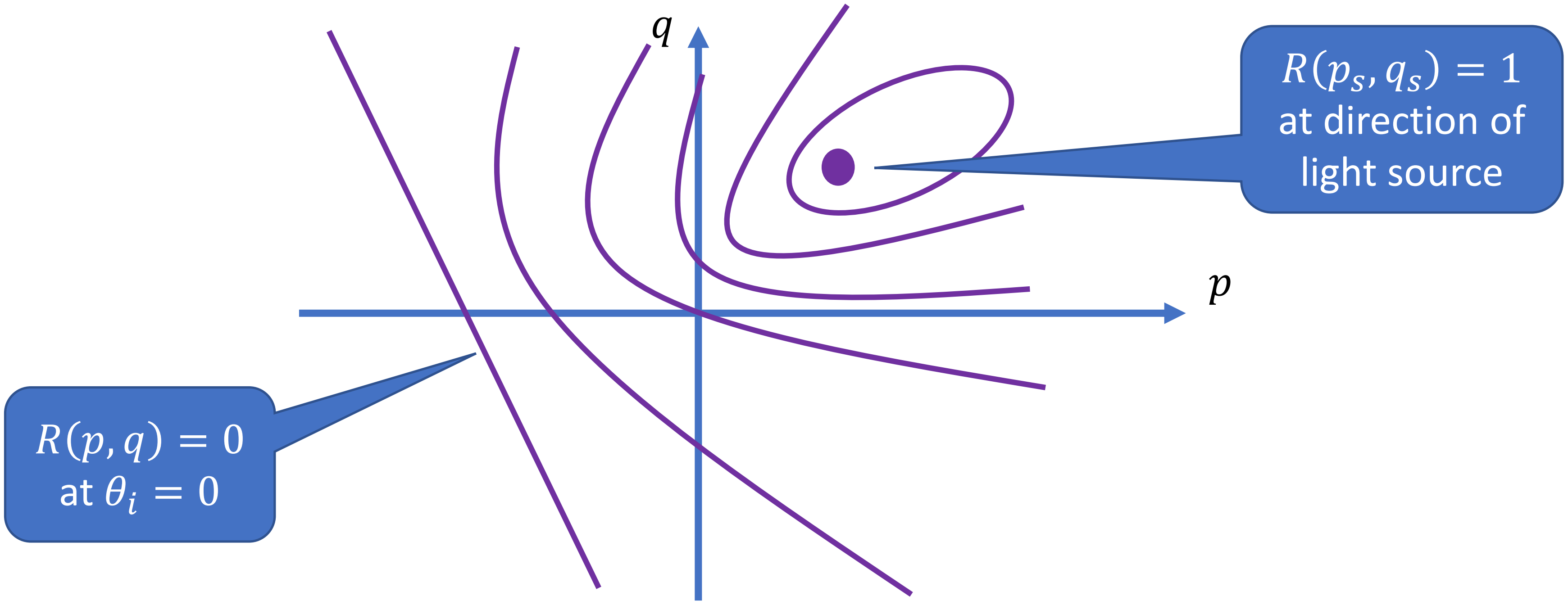
- Instead of using the BDRF and the light sources for encoding the surface properties, we instead use the **reflectance map** which for every (p, q) gives us the radiance of the surface patch

- In case of a single known light source (p_s, q_s) and a Lambertian surface we therefore get

$$R(p,q) = \cos\theta_i = \frac{pp_s + qq_s + 1}{\sqrt{1 + p^2 + q^2}\sqrt{1 + p_s^2 + q_s^2}}$$

Reflectance map

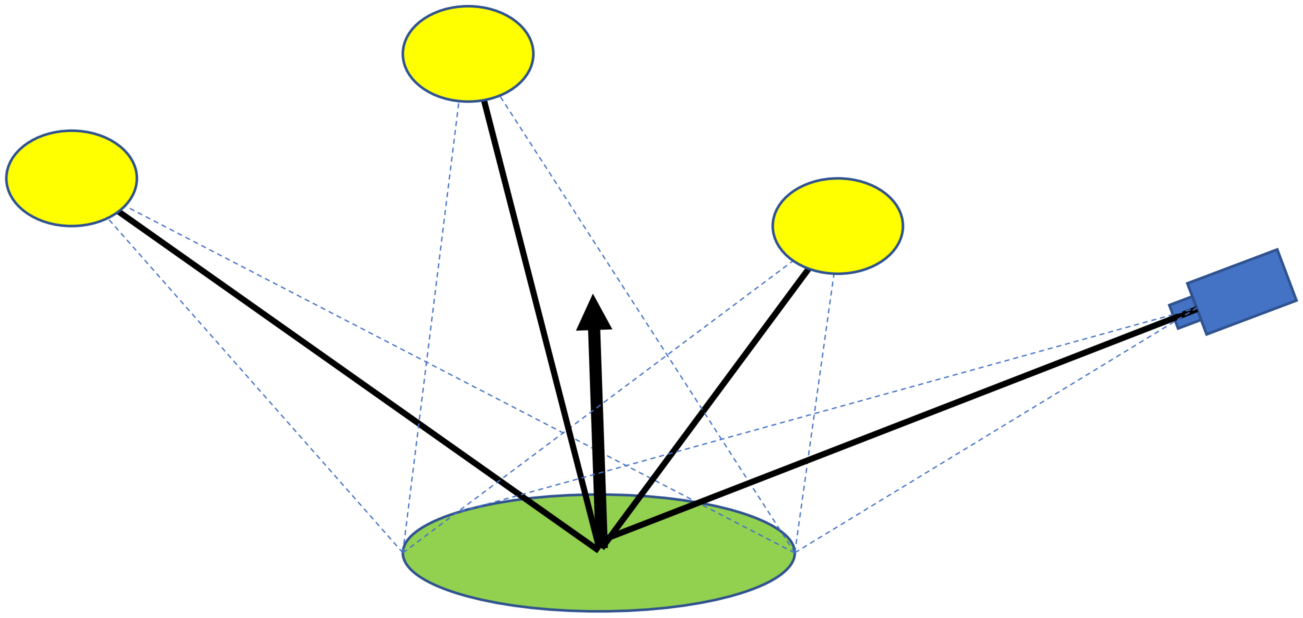
- The single source Lambertian surface reflectance map looks as follows


q
p
$R(p_s, q_s) = 1$ at direction of light source
$R(p, q) = 0$ at $\theta_i = 0$

$$R(p,q) = \cos\theta_i = \frac{pp_s + qq_s + 1}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}}$$

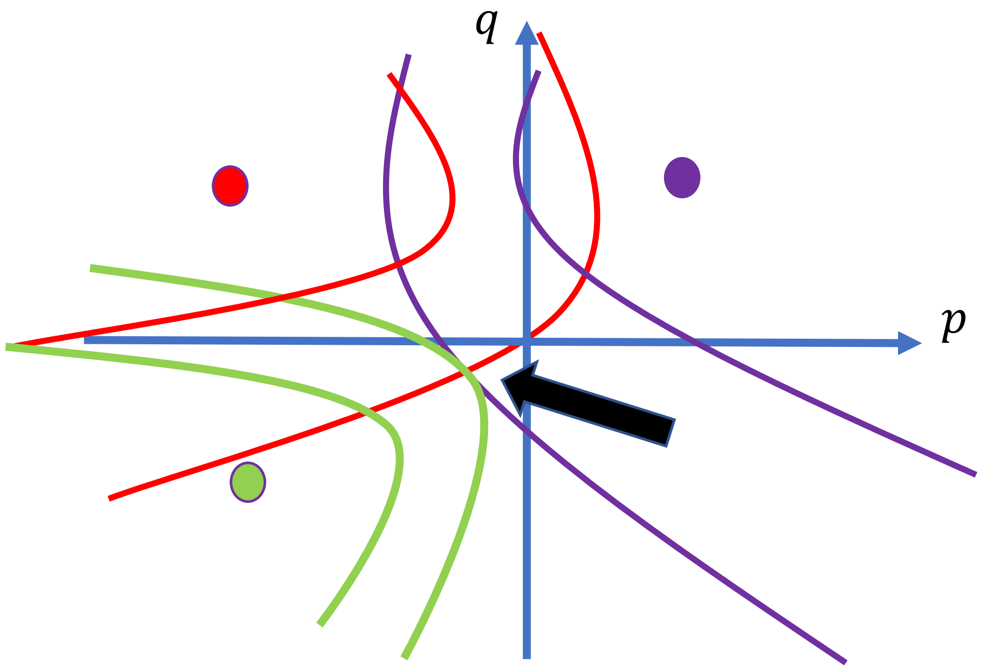
Photometric stereo

- The idea of photometric stereo is to take more than three pictures of the same scene under different known lighting conditions
- For every pixel we then get three different brightness values corresponding to different reflectance maps



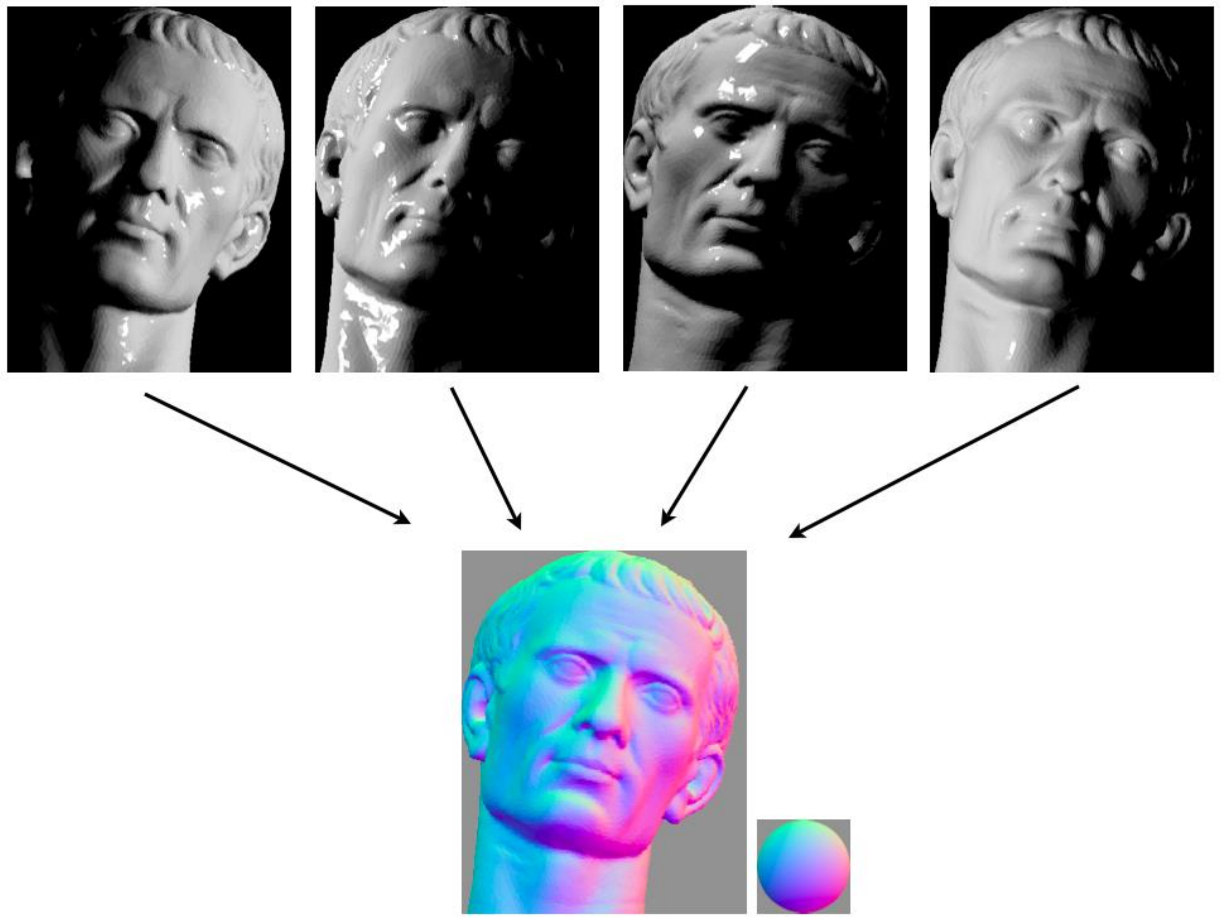
Photometric stereo

- The three reflectance maps are know, which allows to find the level where all three lines intersect in a common point
- This point is for each pixel the normal vector in the parallel projection



Photometric stereo

- This allows for each pixel to use the different brightness values and the known direction of light to calculate a normal vector



Photometric stereo

- Let the image thee brightness values for the three different light sources be

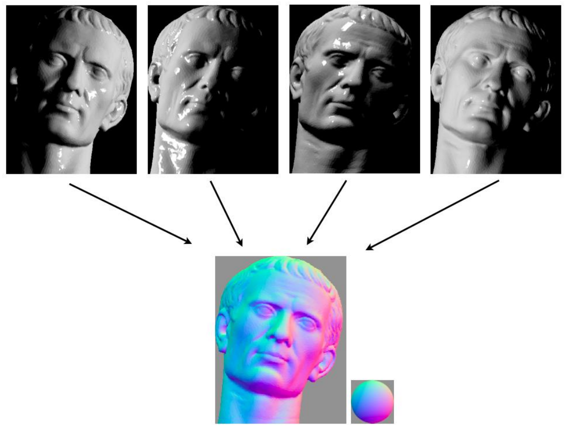
$$I_1[x,y] = \rho[x,y]\boldsymbol{n}^T\boldsymbol{s}_1$$
$$I_2[x,y] = \rho[x,y]\boldsymbol{n}^T\boldsymbol{s}_2$$
$$\mathrm{I}_3[x,y] = \rho[x,y]\boldsymbol{n}^T\boldsymbol{s}_3$$

- Then we can pixel-by-pixel write this into matrix form

$$\begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix} = \rho \begin{pmatrix} \boldsymbol{s}_1^T \\ \boldsymbol{s}_2^T \\ \boldsymbol{s}_3^T \end{pmatrix} \boldsymbol{n}$$

- And solve for the normal vector

$$\boldsymbol{n}' = \begin{pmatrix} \boldsymbol{s}_1^T \\ \boldsymbol{s}_2^T \\ \boldsymbol{s}_3^T \end{pmatrix}^{-1} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix}$$



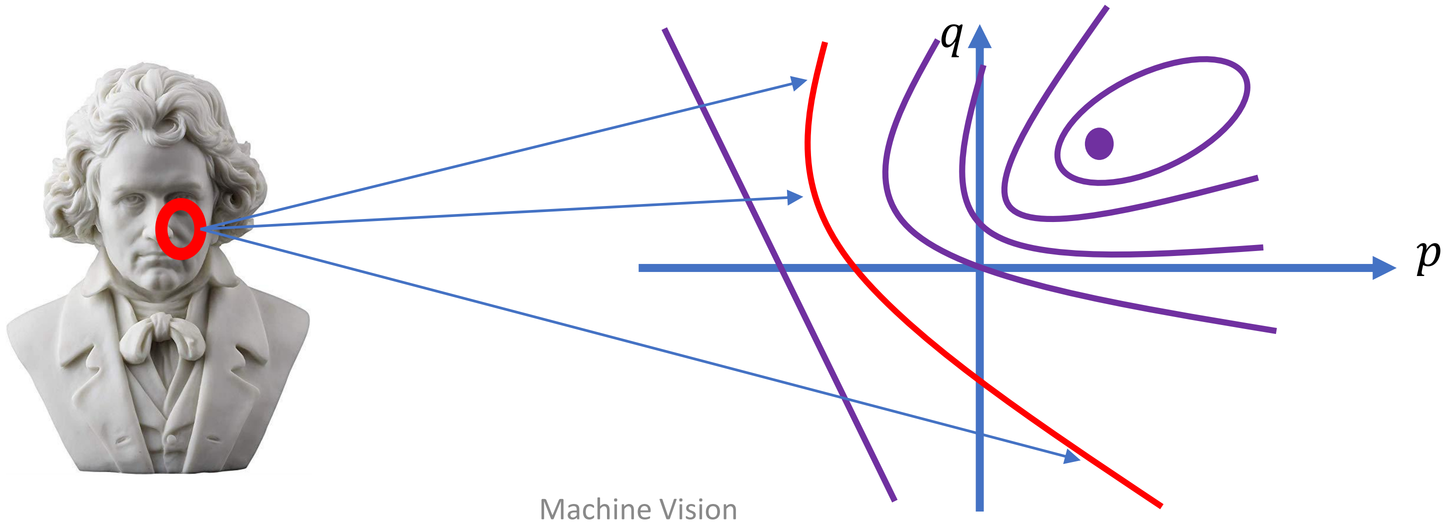
- And normalise

$$\boldsymbol{n} = \frac{\boldsymbol{n}'}{|\boldsymbol{n}'|}$$

Shape from shading

- Photometric stereo is only applicable if we consider multiple images under different lighting conditions

- Can we do the same with only one image?
- Not without extra assumptions, because for each pixel we only know a curve in (p, q)-space where the normal could be



Shape from shading

- The idea behind shape-from-shading is to require the normal vector change to be smooth
- The single image is assumed to be given by

$$I[x, y] = R[p[x, y], q[x, y]]$$

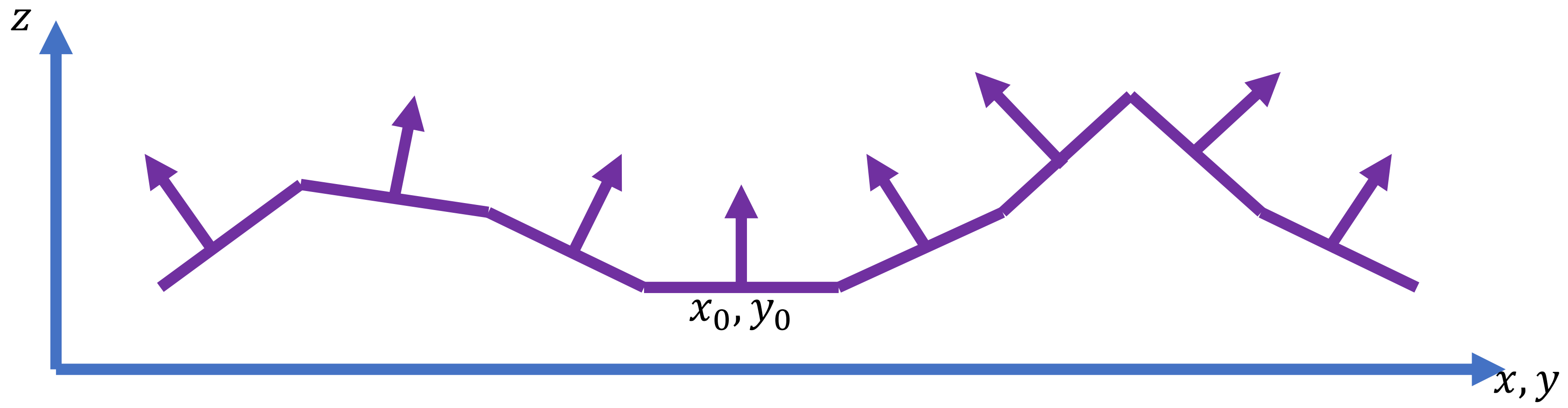
- With the reflectance map R known, but the normal vectors $p[x, y]$ and $q[x, y]$ unknown
- We now try to find the function p and q by minimising the cost function (again, trying to find a compromise between fit to the data and smoothness)

$$E[p, q] = \lambda \sum_{x,y} \left(R\left[p[x, y], q[x, y]\right] - I[x, y]\right)^2 + \sum_{x,y} (\nabla p[x, y])^2 + \sum_{x,y} (\nabla q[x, y])^2$$

Recovering the profile

- These photometric stereo methods calculate a field of normal vectors
- To recover the depth profile, the normal vectors need to be integrated to get a depth map

$$z[x,y] = z[x_0,y_0] + \int_{(x_0,y_0)}^{(x,y)} p\,dx + q\,dy$$



Optical flow

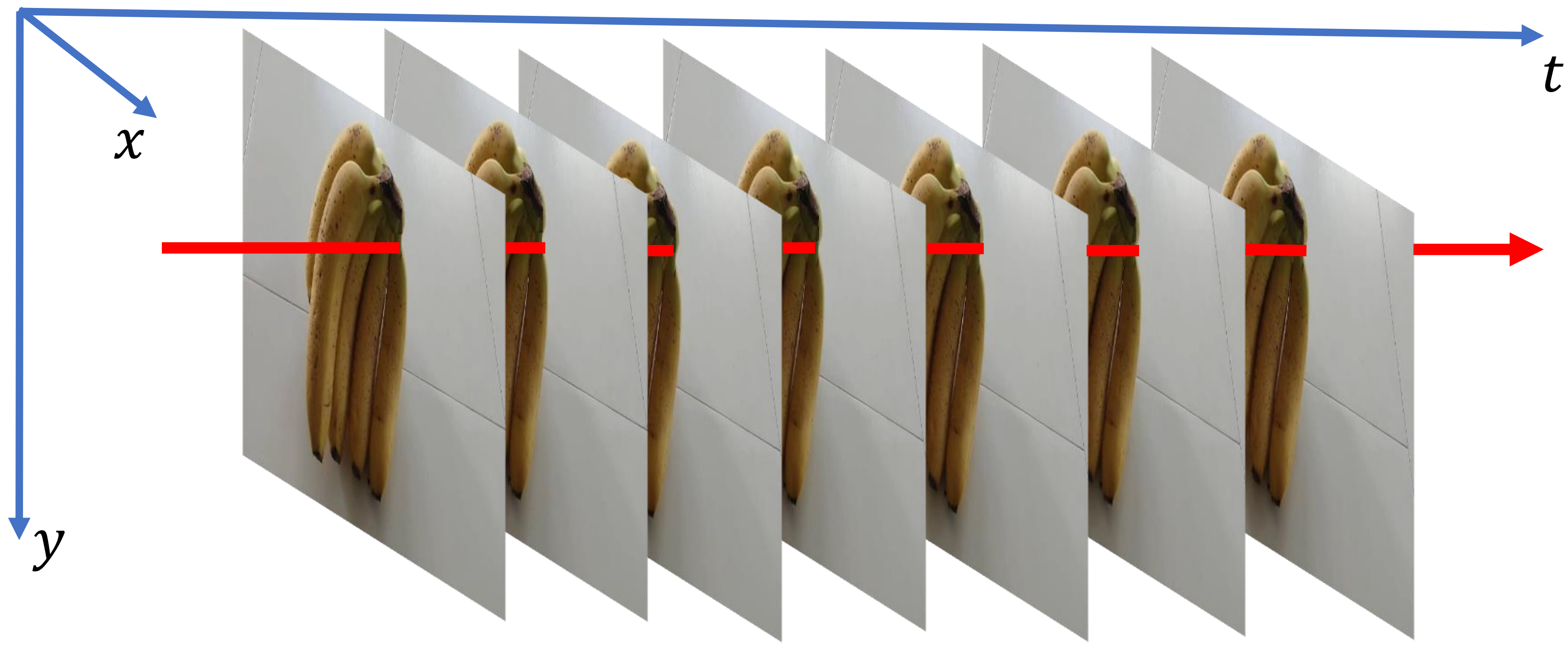
- We will now look at image sequences, i.e. videos, and what additional information we can extract from the motion of the camera relative to the objects in the scene

Optical flow

- By adding a time component we can consider the video a 3d cube of images

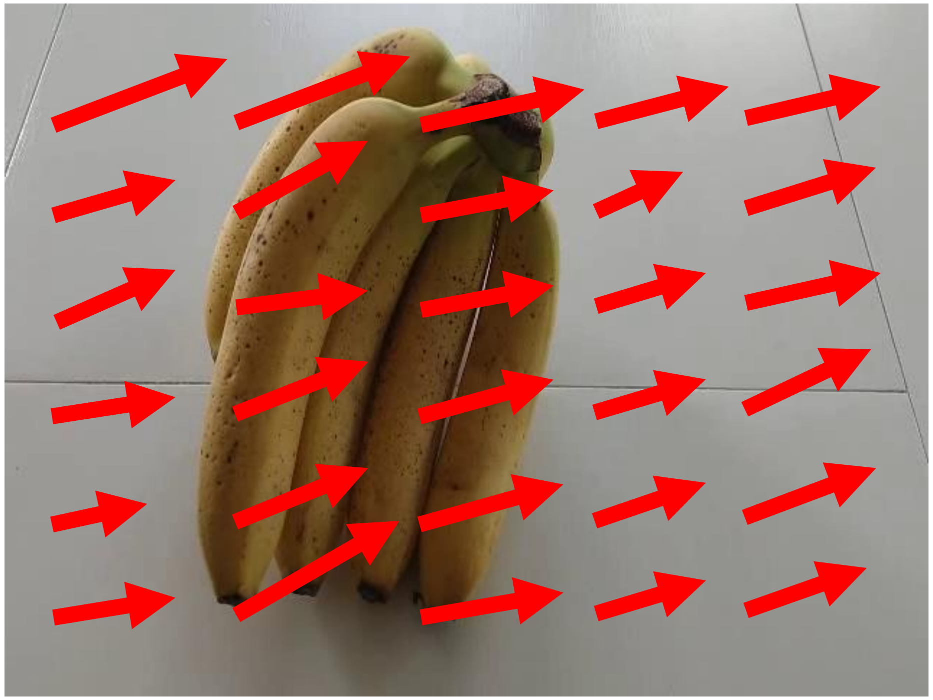
$$I[x, y, t]$$

- Every scene point can now be tracked along a (not necessarily straight) line through this cube



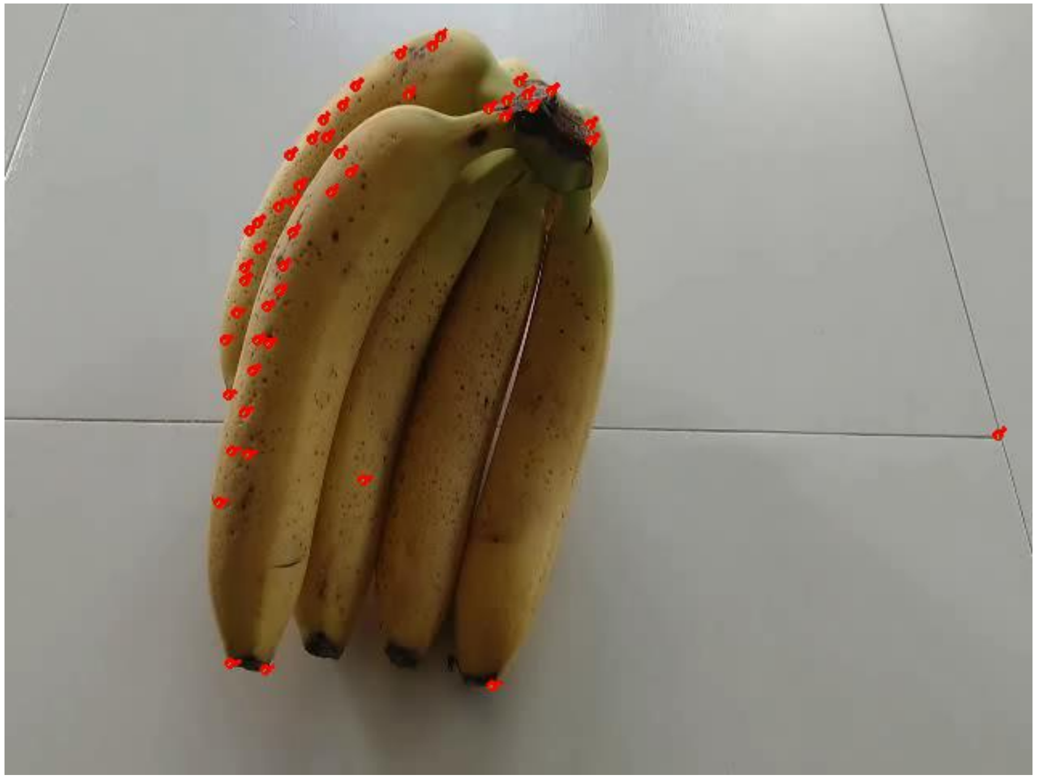
Optical flow

- Looking at the cube from the latest picture into the past, we can draw the projection of this line (either taking in all images, or just the previous two) for every pixel in the image
- This vector field represents the perceived motion in the current frame



Optical flow

- We can look at the optical flow in every image of the sequence and track the object over time through the sequence



Optical flow

- We typically assume that time is constantly progressing, so we look how every pixel is moving from one time-step to the next
- Looking at a single point (x, y) at time t we assume that it moves in one time-step Δt to a new location $(x + \Delta x, \mathrm{y} + \Delta y)$

- To understand how this movement works we look at the Taylor expansion

$$I[x + \Delta x, y + \Delta y, t + \Delta t] \approx I[x, y, t] + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

- This tells us how each intensity value in the image at time t moves to a new position $(x + \Delta x, \mathrm{y} + \Delta y)$ at time $t + \Delta t$

Optical flow

- It is reasonable to assume that intensity values do not change for the same object between time-steps, i.e. $I[x + \Delta x, y + \Delta y, t + \Delta t] = I[x, y, t]$, so we can conclude that

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y = -\frac{\partial I}{\partial t}\Delta t$$

- We already know how to calculate the gradient images $I_x = \frac{\partial I}{\partial x}$ and $I_y = \frac{\partial I}{\partial y}$
- Very similar to how we computed these, we can assume $\Delta t = 1$ and use the difference between subsequent images as approximation to

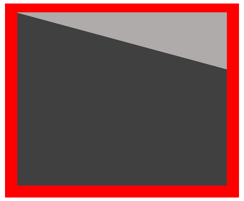
$$\frac{\partial I}{\partial t}\Delta t = I[x, y, t + 1] - I[x, y, t] = I_t$$

Aperture problem

- In conclusion, the optical flow has to follow the **optical flow equation**

$$I_x \Delta x + I_y \Delta y = -I_t$$

- This imposes one constraint in two-unknowns for each pixel, the problem is therefore under-constrained in the direction orthogonal to the gradient
- This is called the **aperture problem:** We are only able to locally observe motion orthogonal to the lines in the image



Horn–Schunck method

- In order to deal with the aperture problem, we (again) introduce an additional smoothness constraint

$$E[p,q] = \sum_{x,y} \left(I_x p + I_y q + I_t\right)^2 + \lambda \sum_{x,y} |\nabla p|^2 + |\nabla q|^2$$

- Finding the optimum to this cost function results in a smooth optical-flow field fitted to the image data
- This approach is called **dense optical flow**, because it calculates a flow vector for every pixel coordinate

Lucas-Kanade method

- Another approach is to assume that the optical flow is approximately constant in a local neighbourhood

- In this case the optical flow equation has to hold for all pixels in a patch around a feature

$$I_x[x_i, y_i]\Delta x + I_y[x_i, y_i]\Delta y = -I_t[x_i, y_i]$$

- Or in matrix notation

$$\underbrace{\begin{pmatrix} I_x[x_1, y_1] & I_y[x_1, y_1] \\ \vdots & \vdots \\ I_x[x_n, y_n] & I_y[x_n, y_n] \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}}_{v} = \underbrace{\begin{pmatrix} -I_t[x_1, y_1] \\ \vdots \\ -I_t[x_n, y_n] \end{pmatrix}}_{b}$$

Lucas-Kanade method

- Finding the least-squares solution to an overdetermined equation system

$$Av = b$$

- Means minimising the squared residual

$$\Omega = (Av - b)^2 = vA^T Av \; - 2b^T Av + b^T b$$

- The minimum has to satisfy

$$\frac{\partial \Omega}{\partial v} = 2A^T Av - 2A^T b = 0$$

- or

$$A^T Av = A^T b$$

- Which has the solution

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

Lucas-Kanade method

- Putting this all together we obtain the optical flow based on a patch around a feature to be

$$\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix} = \begin{pmatrix}\sum_i I_x[x_i,y_i]I_x[x_i,y_i] & \sum_i I_x[x_i,y_i]I_y[x_i,y_i]\\ \sum_i I_x[x_i,y_i]I_y[x_i,y_i] & \sum_i I_y[x_i,y_i]I_y[x_i,y_i]\end{pmatrix}^{-1} \begin{pmatrix}-\sum_i I_x[x_i,y_i]I_t[x_i,y_i]\\ -\sum_i I_y[x_i,y_i]I_t[x_i,y_i]\end{pmatrix}$$

- This equation (again) contains the inverse structure tensor

- Therefore, feature points where the eigenvalues of the structure tensor are large are best suited for optical flow computation, for all other points the matrix we have to invert is close to singular

KLT tracking

- Features with large eigenvalues give the best results for the Lucas-Kanade method
- We already saw that the Shi-Tomasi criterion helps us identify features for which this is the case

- The combination of these two algorithms is called Kanade-Lucas-Tomasi or KLT tracking

- As opposed to the dense optical flow calculated using the Horn-Schunck method, the KLT method is restrict to previously extracted interest points only, which is why these methods are called **sparse optical flow**

KLT tracking

- We need to initialise the feature point locations where we want to start tracking

```
p0 = cv2.goodFeaturesToTrack(new_img, 200, 0.3, 7)
```

No of points

Quality level

Minimum distance between points

KLT tracking

- We need to initialise the feature point locations where we want to start tracking

```
p0 = cv2.goodFeaturesToTrack(new_img, 200, 0.3, 7)
```

- Then we acquire the next image and calculate the optical flow

```
old_img = new_img
new_img = …
p1, st, err  = cv2.calcOpticalFlowPyrLK(old_img, new_img, p0, None)
```

New location of points in next image

Status indicator for each feature (1=valid, 0=lost)

KLT tracking

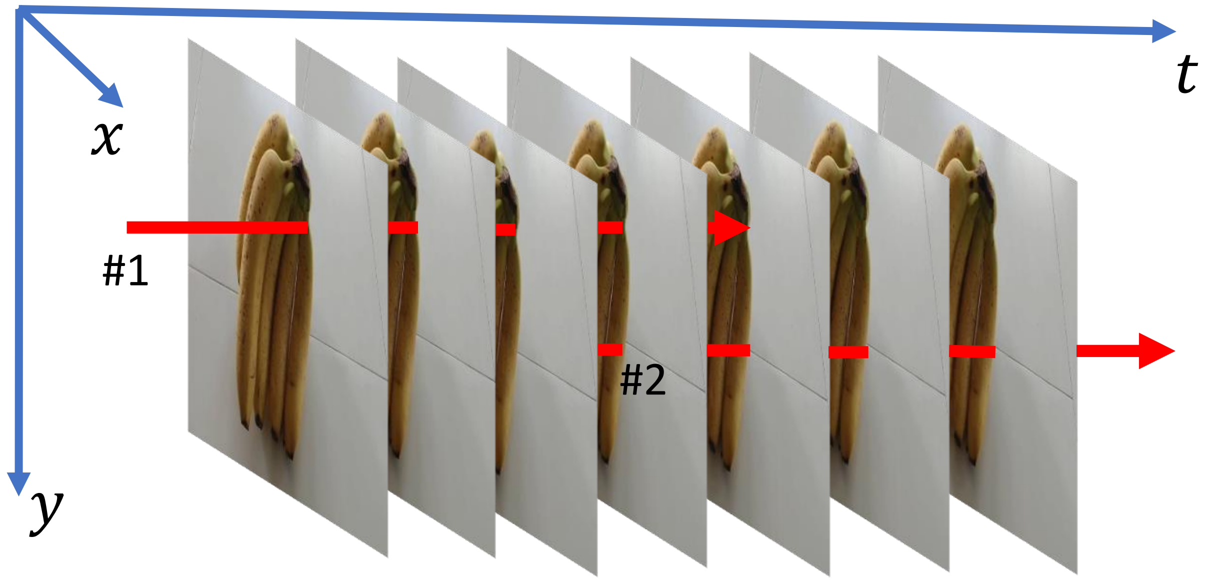
- We need to check if we still have features that are being tracked, otherwise we need to fill find new features again

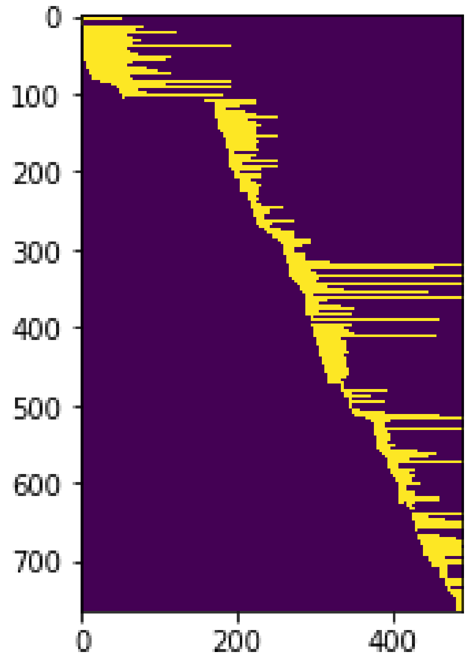
```
p0 = p1[st==1].reshape(-1,1,2)
if len(p0)<100:
   new_p0 = cv2.goodFeaturesToTrack(new_img, 200-len(p0), 0.3, 7)
```

- This does not enforce the minimum distance, so we need to ensure that we are not start tracking a feature too close to any existing one

KLT tracking

- Features are lost and new features are started all the time
- As long as there is sufficient overlap, we can analyse the motion of the full sequence





KLT live demo

Thank you for your attention!