Discussion 6

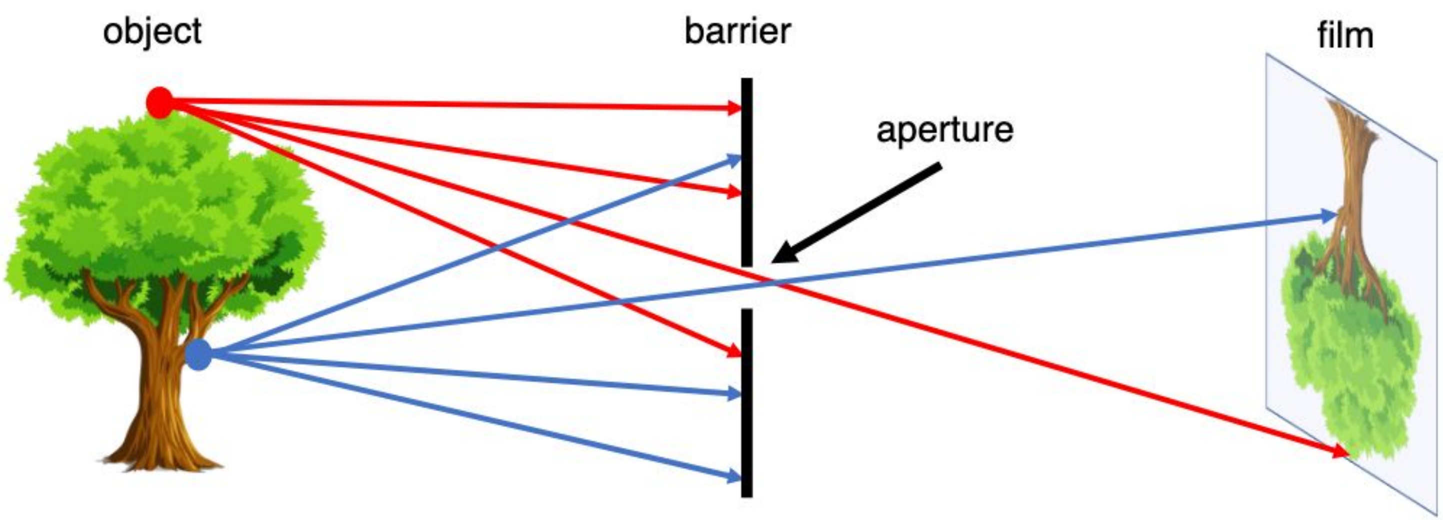
CS188 - Fall 19

Objectives

- Review of Camera Models
  - Aperture, focal length and depth of field
  - Perspective projection equations
  - Return to a small problem
  - Affine vs Homography
- Introduction to Epipolar Geometry
  - Vocabulary
  - 8 point algorithm
- RANSAC
- Homework 2
  - Goal
  - Experimental set-up
  - Template matching
  - Normalized cross-correlation

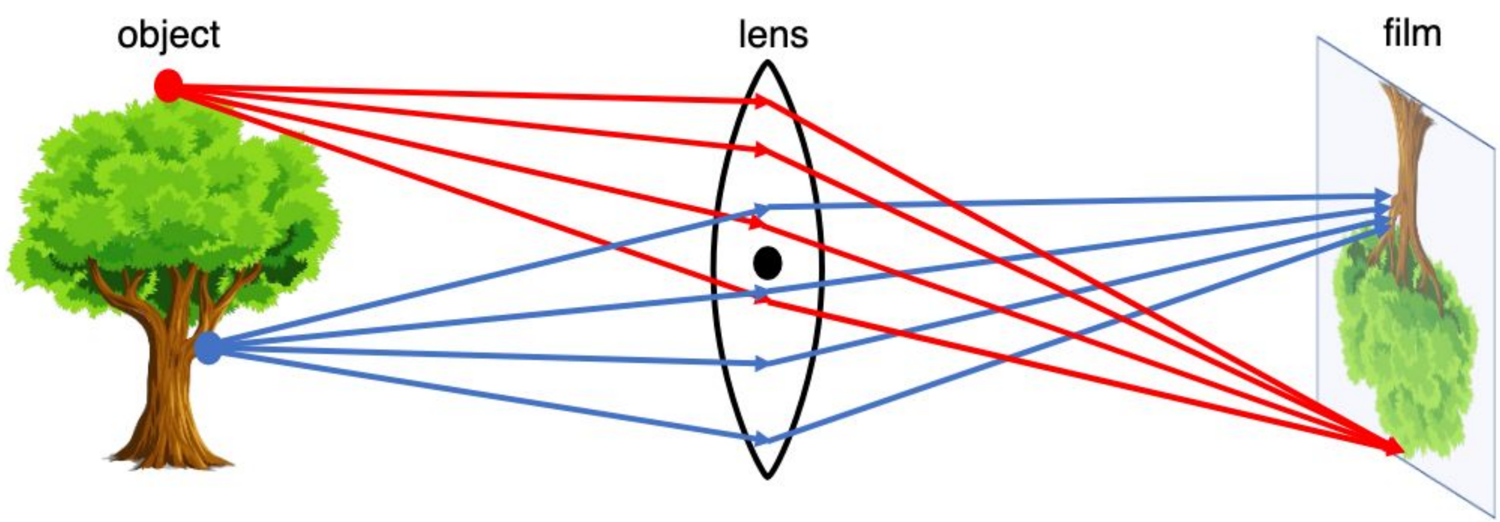
Pinhole camera

- Every point in the scene maps to a single point in the image plane
  - This only holds for an infinitesimally small aperture!
    - In that case almost no light goes through
    - Compromise between sharpness and brightness



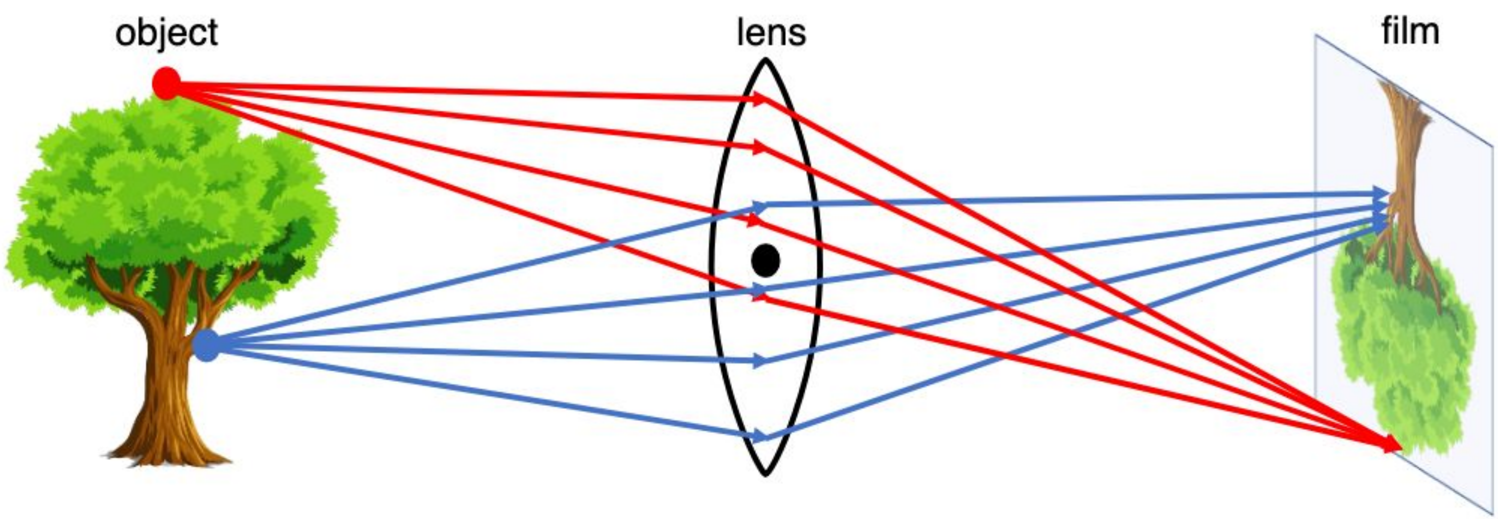
Lens Cameras

- Lenses are devices that can focus or disperse light
  - This mitigates the brightness vs sharpness problem
  - But ...



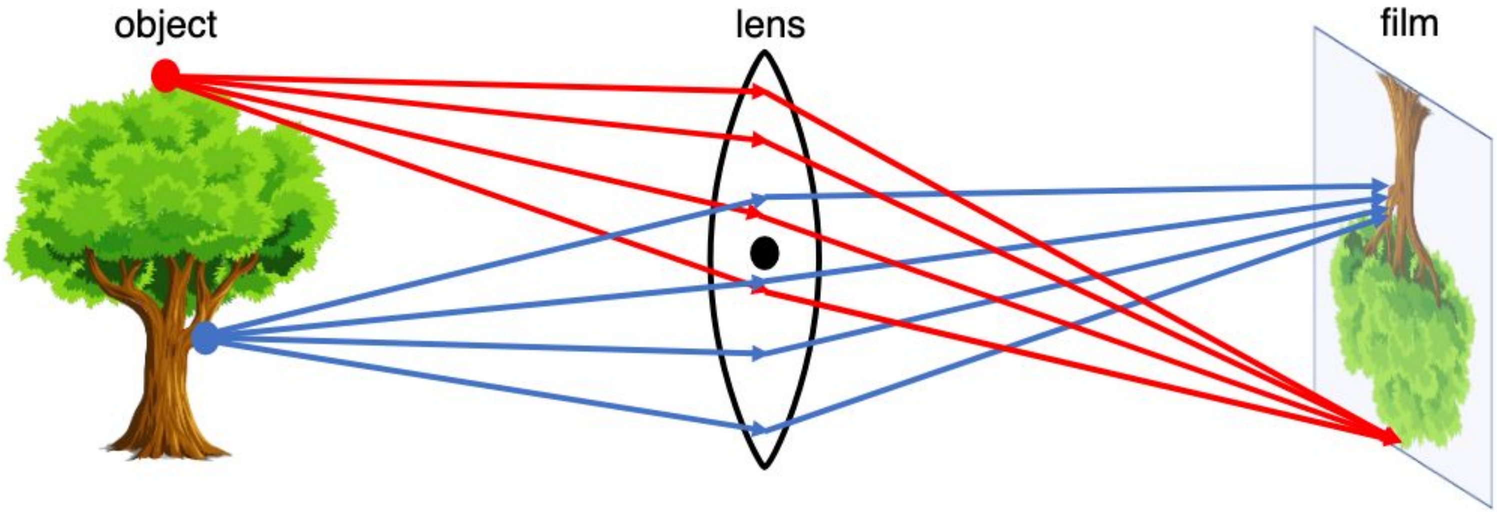
Lens Cameras

- Lenses are devices that can focus or disperse light
  - This mitigates the brightness vs sharpness problem
  - But a point doesn't necessarily always map to a point
    - This is only valid in



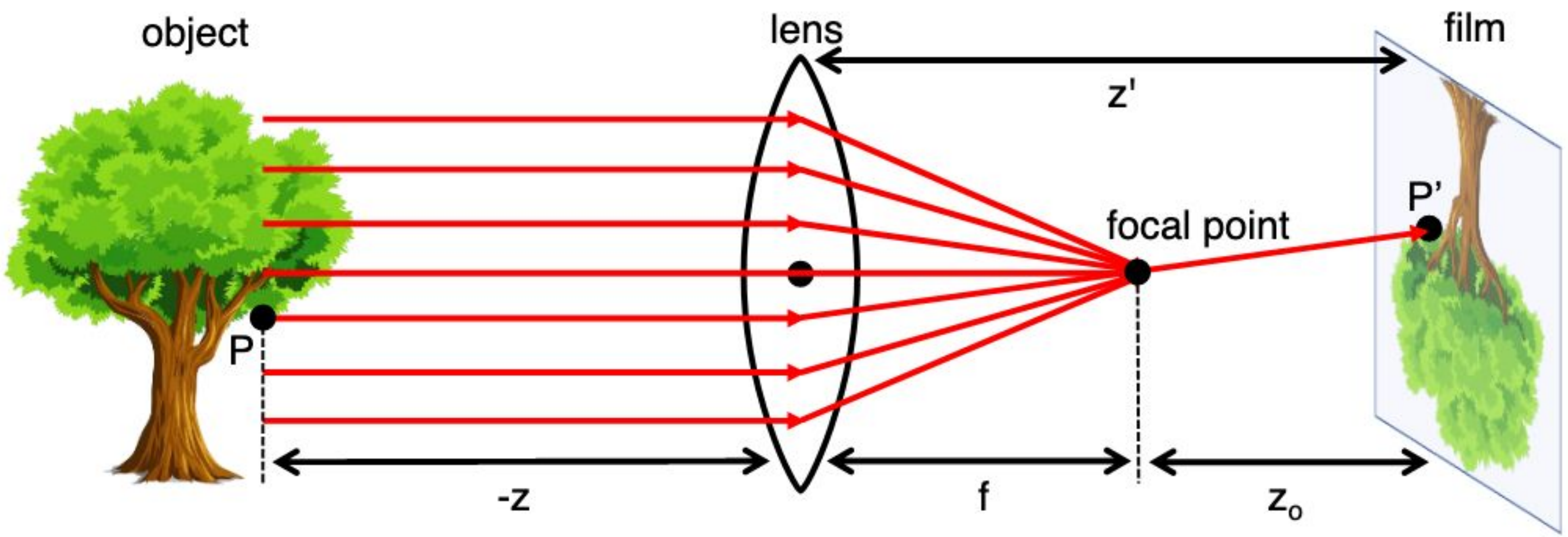
Lens Cameras

- Lenses are devices that can focus or disperse light
  - This mitigates the brightness vs sharpness problem
  - But a point doesn't necessarily always map to a point
    - This is only valid in a plane of constant depth
    - ie: lenses have a specific distance for which objects are in focus



Lens Cameras

- Lenses are devices that can focus or disperse light
  - This mitigates the brightness vs sharpness problem
  - But a point doesn't necessarily always map to a point
    - This is only valid in a plane of constant depth
    - ie: lenses have a specific distance for which objects are in focus



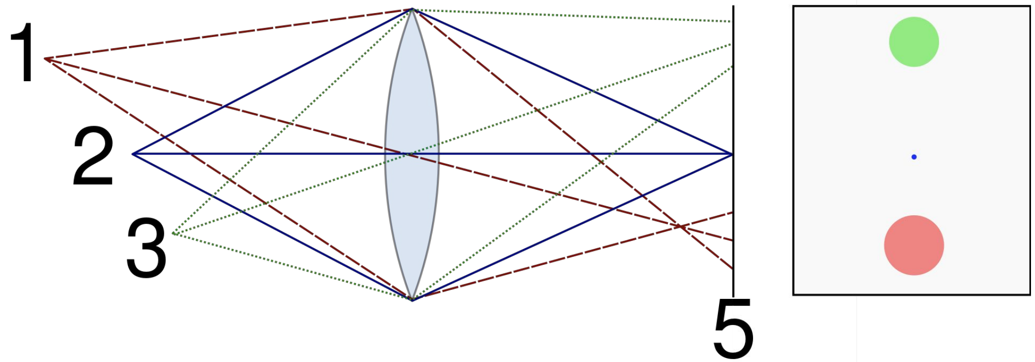
Depth of Field

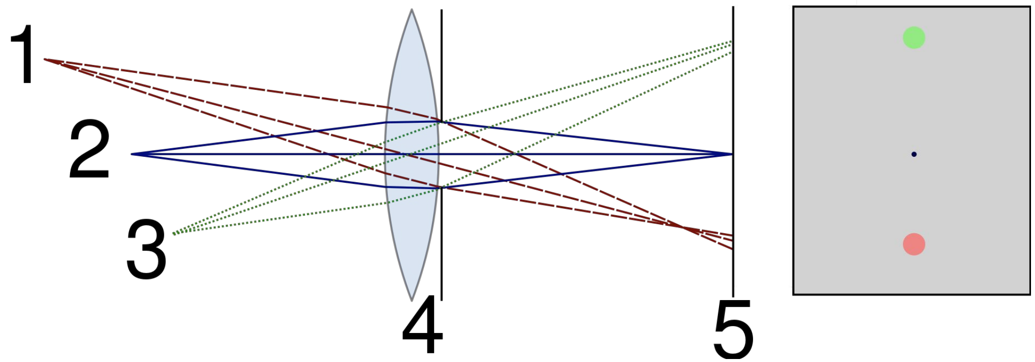
- Depth of Field is the distance between the nearest and farthest object that are in focus

Large Aperture: Shallow depth of field

Short Aperture: Large depth of field

Pinhole Camera: ???

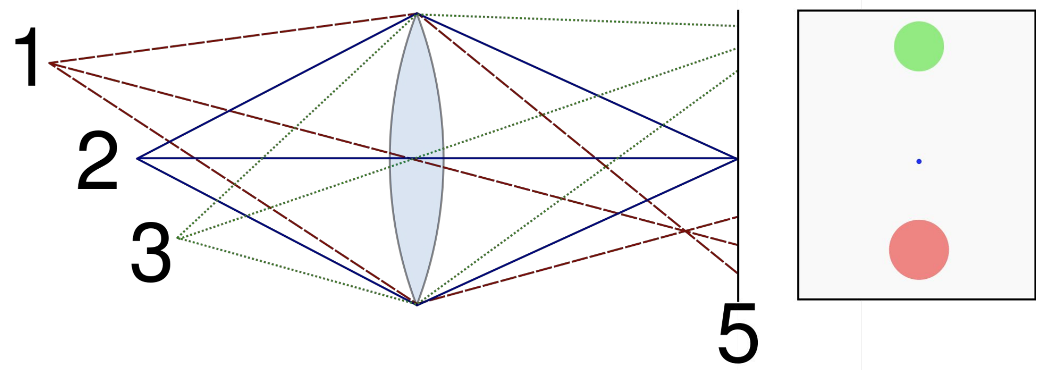




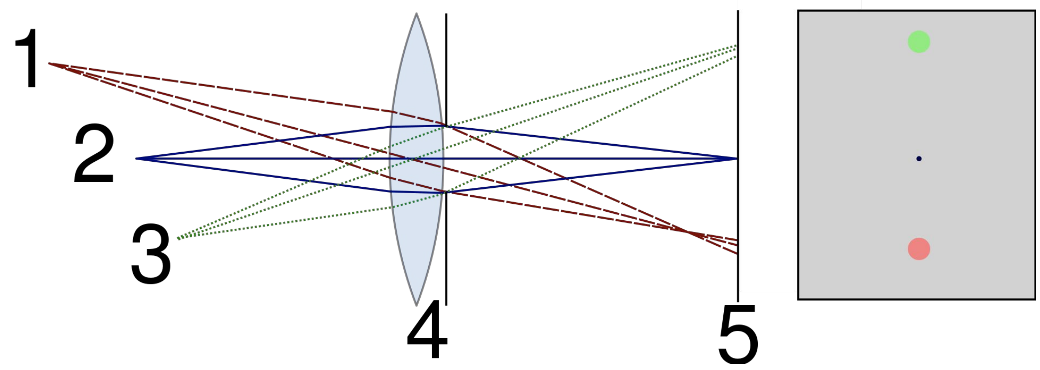
Depth of Field

- Depth of Field is the distance between the nearest and farthest object that are in focus

Large Aperture: Shallow depth of field

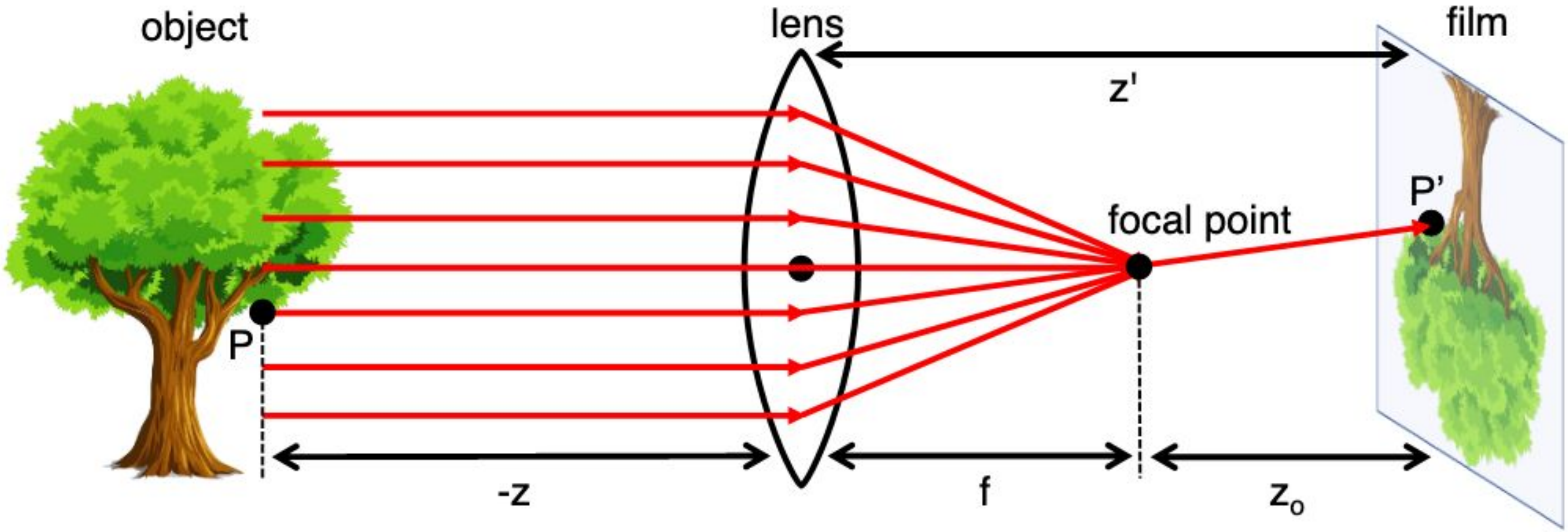


Short Aperture: Large depth of field



Pinhole Camera: Infinite depth of field

Deriving the camera matrix



First: you project: (Note that the points at the center of the lens are not deviated)

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} z' \frac{x}{z} \\ z' \frac{y}{z} \end{bmatrix}$$

Typically, you can assume that z' = f (pinhole model)

Deriving the camera matrix (cont)

$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} z' \frac{x}{z} \\ z' \frac{y}{z} \end{bmatrix}$ Implies that the origin in the image plane is centered, but in digital images it is in a corner. We must shift the origin to a corner:

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} + c_x \\ f\frac{y}{z} + c_y \end{bmatrix}$$

Right now, in what unit are x’ and y’ expressed?

Deriving the camera matrix (cont)

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} z' \frac{x}{z} \\ z' \frac{y}{z} \end{bmatrix}$$

Implies that the origin in the image plane is centered, but in digital images it is in a corner. We must shift the origin to a corner:

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f \frac{x}{z} + c_x \\ f \frac{y}{z} + c_y \end{bmatrix}$$

Right now, in what unit are x' and y' expressed?

=> As a distance. In digital images, we would like a value in pixels: (k and l are in pixels/cm)

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} fk \frac{x}{z} + c_x \\ fl \frac{y}{z} + c_y \end{bmatrix} = \begin{bmatrix} \alpha \frac{x}{z} + c_x \\ \beta \frac{y}{z} + c_y \end{bmatrix}$$

Switch to homogeneous coordinates

To express this nonlinear transformation as a matrix product we use homogeneous coordinates:

$$P' = \begin{bmatrix} x' \\ y' \\ z \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P = MP$$

Which gives us the camera matrix K:

$$P' = MP = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \end{bmatrix} P = K \begin{bmatrix} I & 0 \end{bmatrix} P$$

RANdom SAmple Consensus

How do I fit a line that best represents my data when there are a lot of *outliers*?

You can see outliers as data points that were perturbed by noise, that don't fit within the statistics of your data

Regular methods have a low tolerance to outliers, they try to find a model that 'accommodates' all points, outliers degrade performance for all points

Instead, RANSAC tries to identify a set of points it can 'ignore' by labelling them as outliers

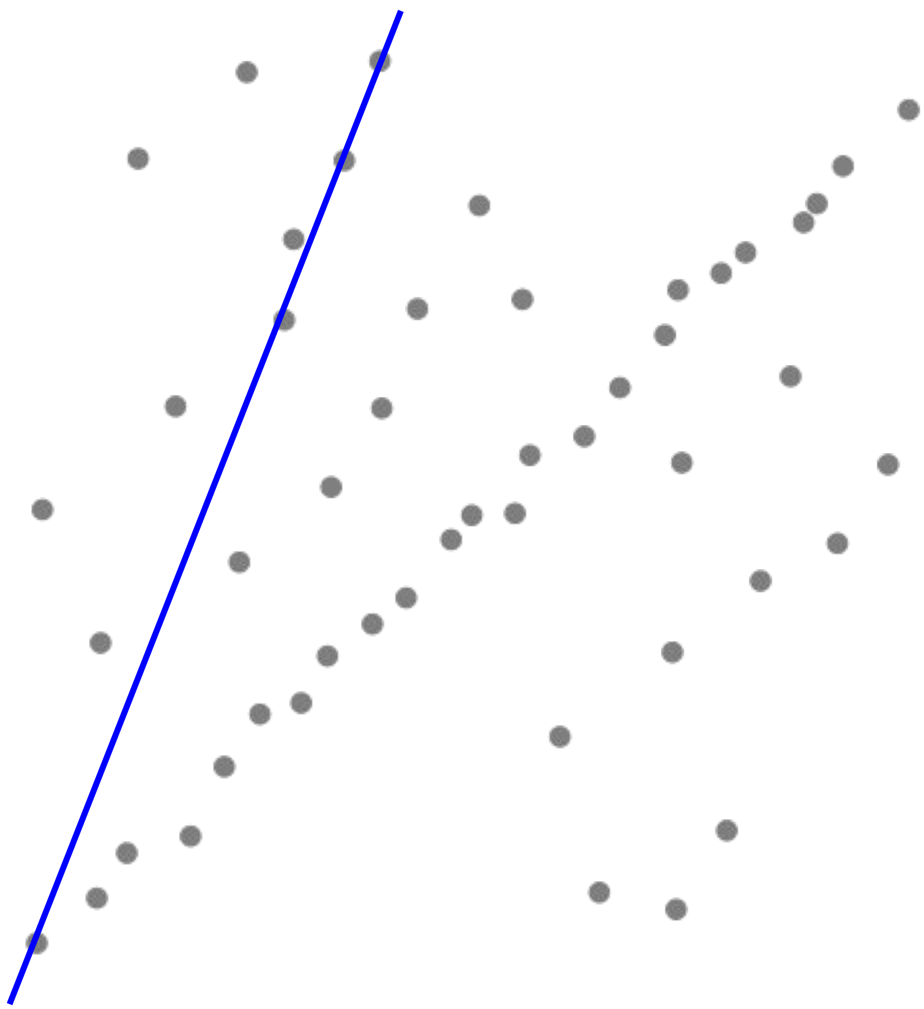
The idea

- Sample a minimal number of data points to be able to estimate your model parameters, at random.
  - Eg: for a line, pick

The idea

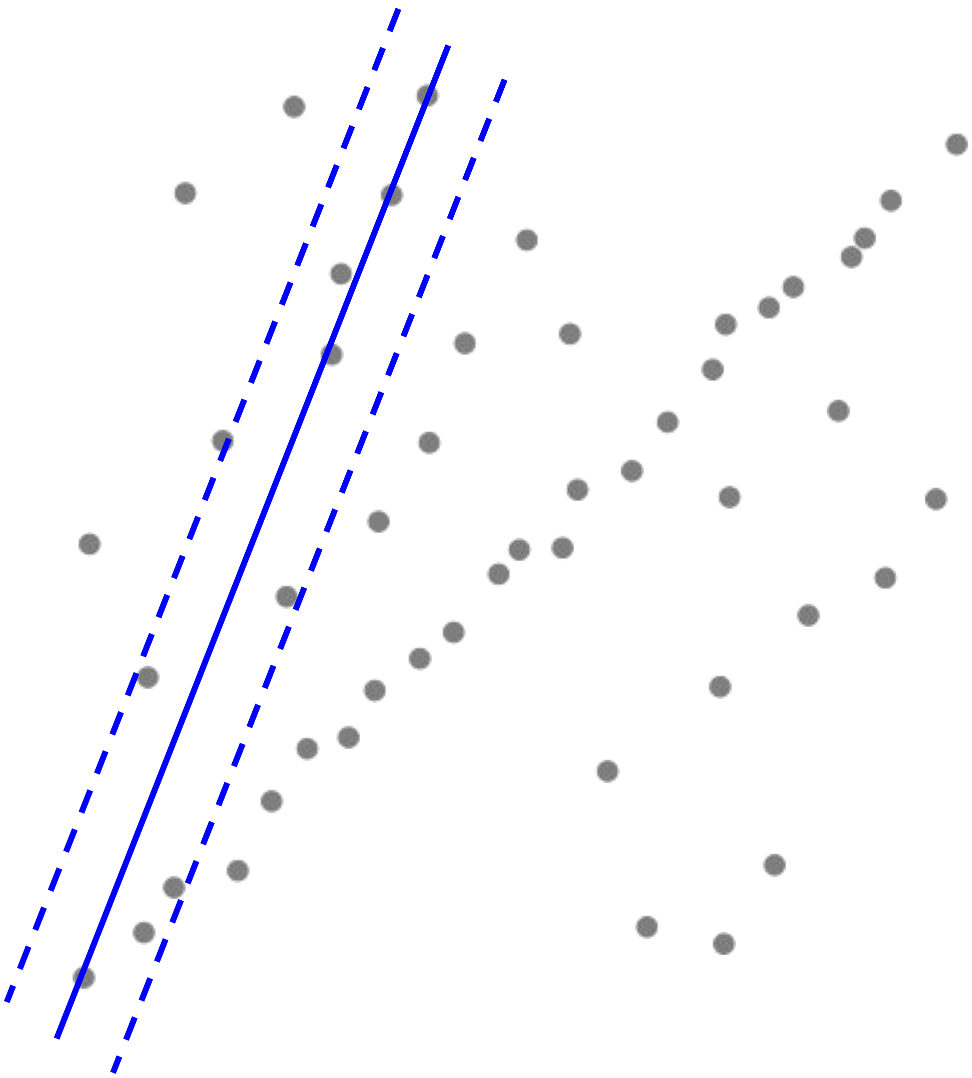
- Sample a minimal number of data points to be able to estimate your model parameters, at random.
  - Eg: for a line, pick 2 points (the minimal number of points that allow you to define a line)
- With these model parameters, estimate which points are inliers and outliers
  - Inliers are the points that are well represented by the model
  - Inliers form the ‘consensus set’
- Iterate until a model with a large enough consensus set is found

An illustration: linear regression



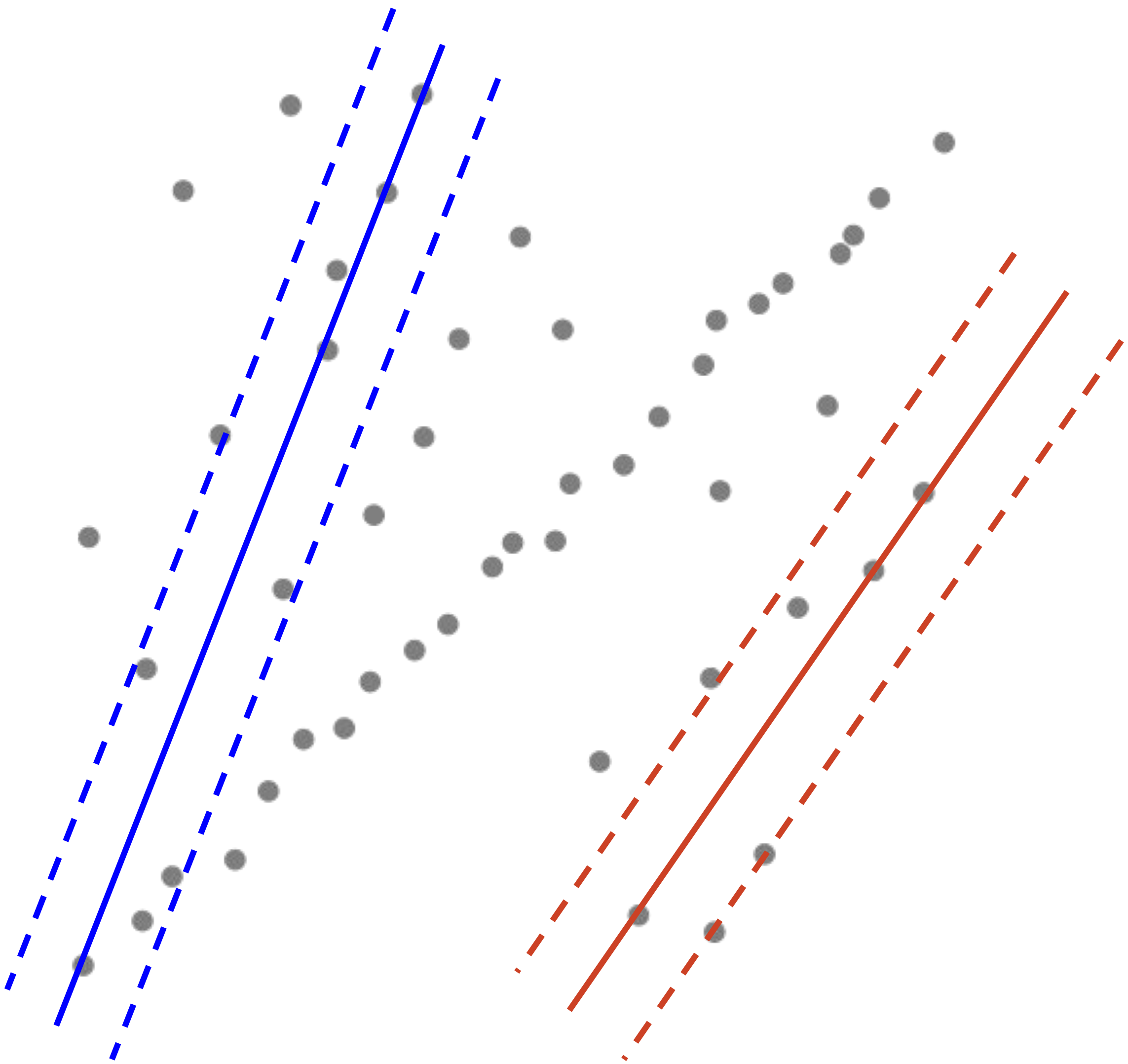
Iteration 1:
1) pick 2 points at random and fit a line (blue)

An illustration: linear regression



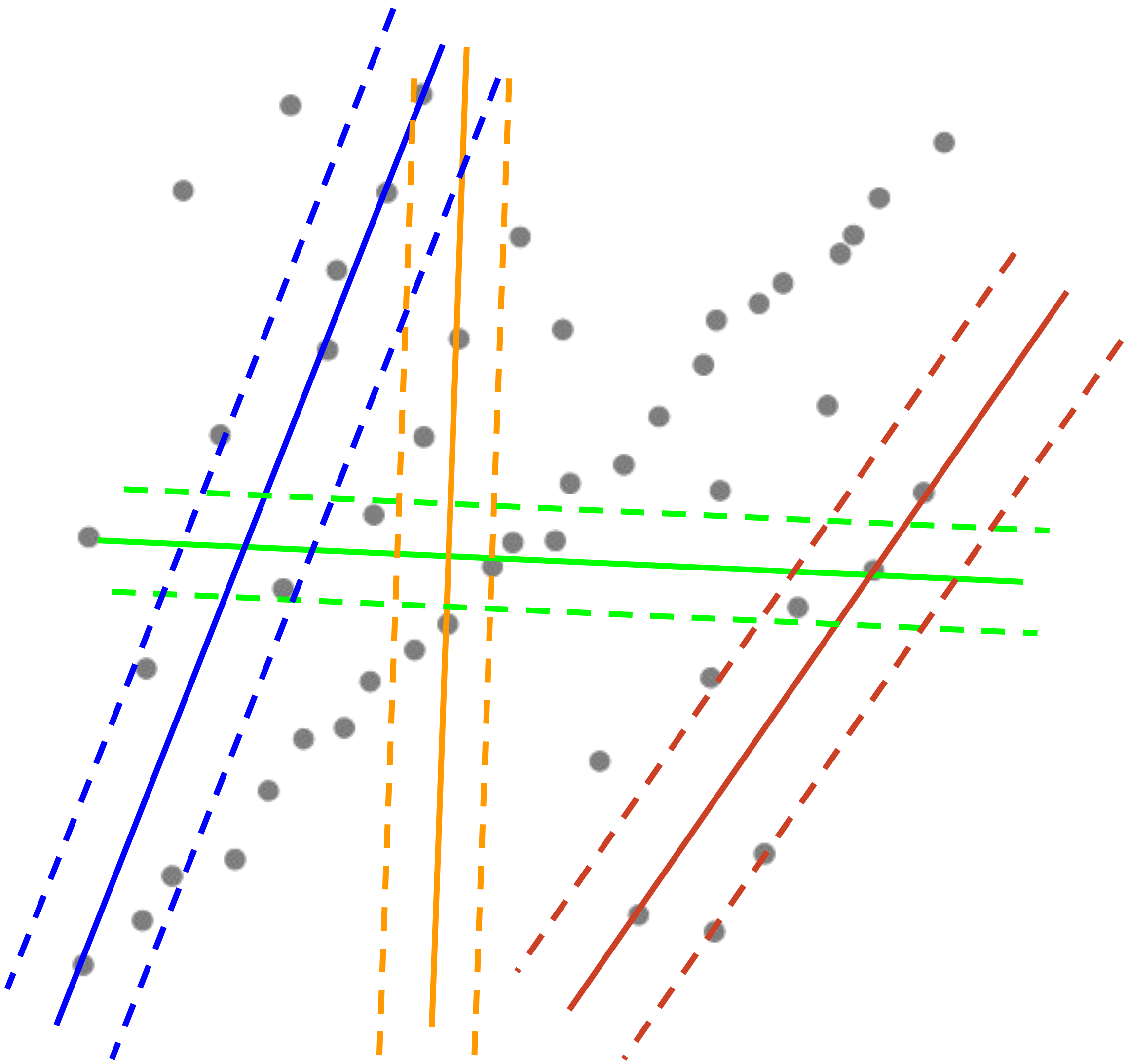
Iteration 1:
1) pick 2 points at random and fit a line (blue)
2) Count inliers (points within the margin)

An illustration: linear regression



<u>Iteration 1:</u>
1) pick 2 points at random and fit a line (blue)
2) Count inliers (points within the margin)
<u>Iteration 2:</u>
1) 1) pick 2 points at random and fit a line (red)
2) Count inliers (points within the margin)

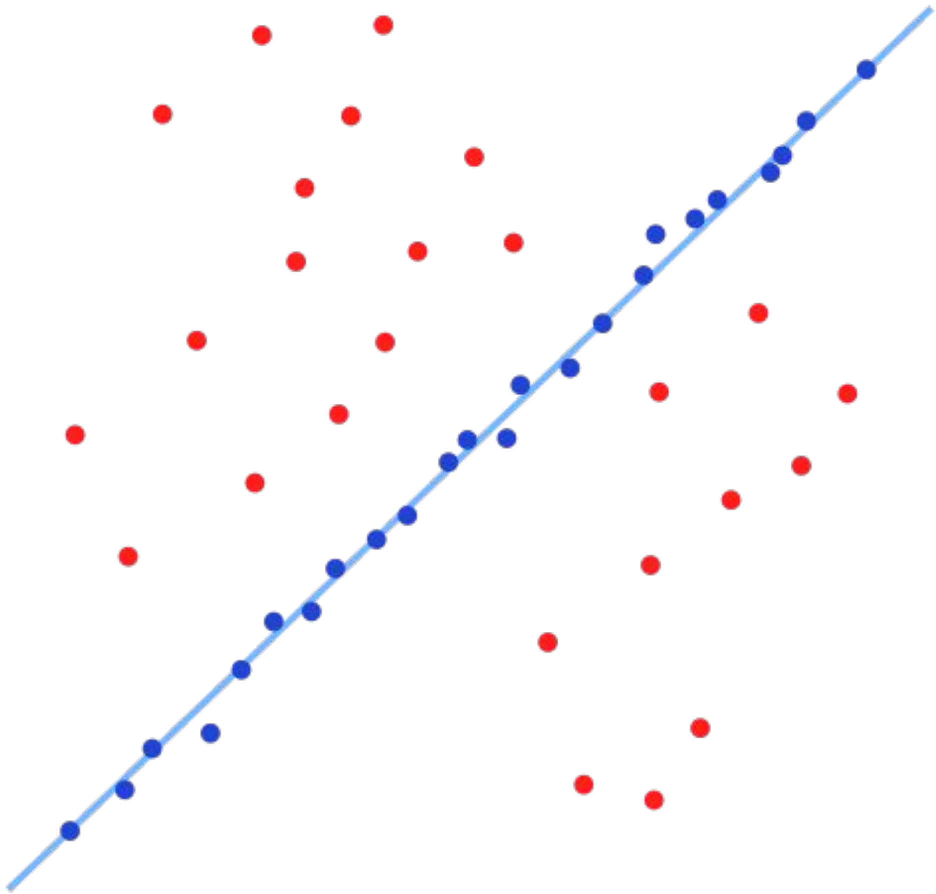
An illustration: linear regression



<u>Iteration 1:</u>
1) pick 2 points at random and fit a line (blue)
2) Count inliers (points within the margin)
<u>Iteration 2:</u>
1) pick 2 points at random and fit a line (red)
2) Count inliers (points within the margin)
<u>Iteration 3:</u>
1) pick 2 points at random and fit a line (green)
2) Count inliers (points within the margin)
<u>Iteration 4:</u>
1) pick 2 points at random and fit a line (orange)
2) Count inliers (points within the margin)
...

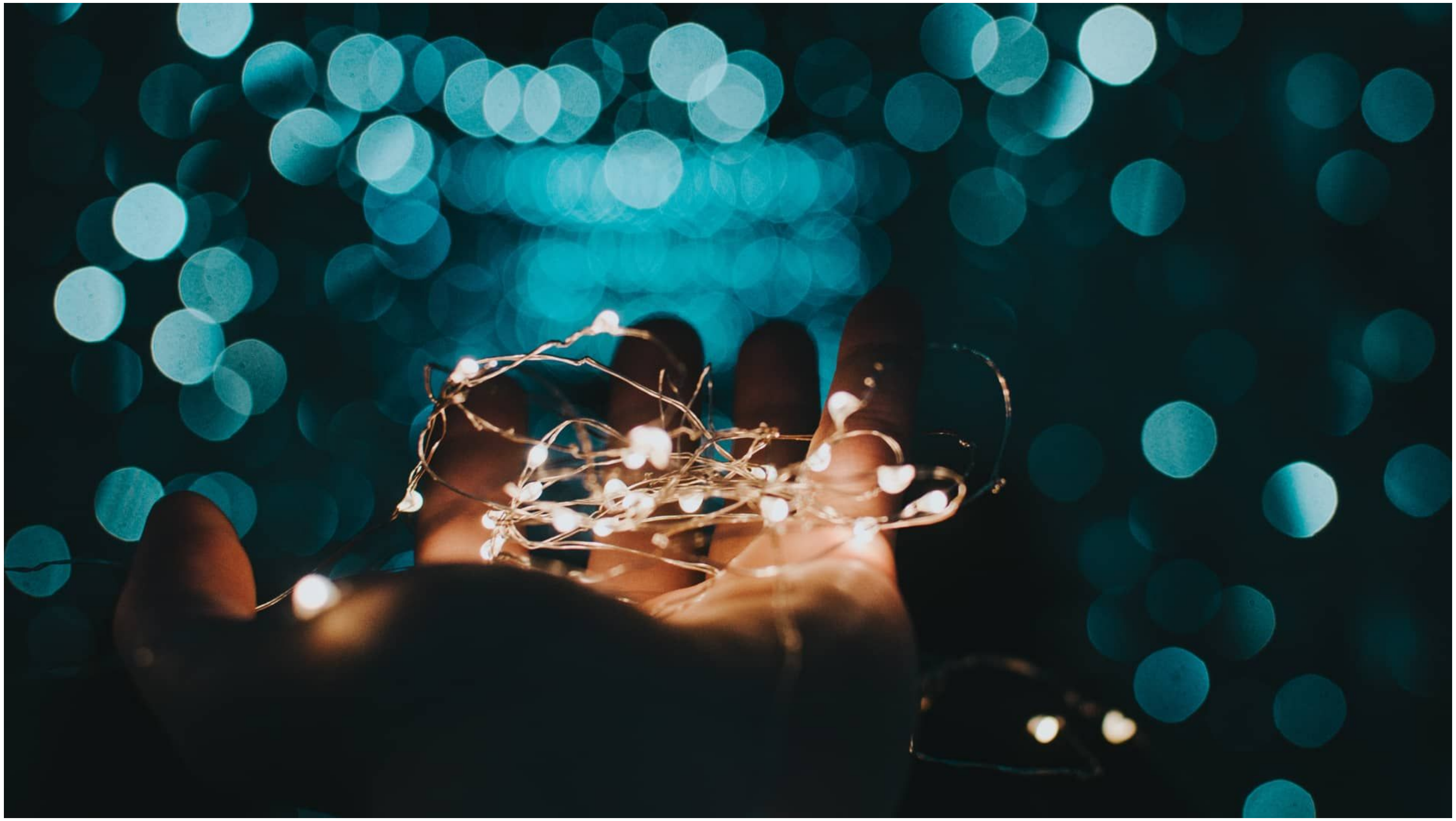
An illustration: linear regression

Iteration 250:



Aperture and the “bokeh” effect

Bokeh: a soft out-of-focus effect, great for portraits, generally considered aesthetically pleasing



Realizing a bokeh

- Bokeh requires that within the image you have both objects in focus and out of focus.
    - Therefore the depth-of-field has to be

Realizing a bokeh

- Bokeh requires that within the image you have both objects in focus and out of focus.
  - Therefore the depth-of-field has to be **small**
    - Remember that objects within the depth-of-field are in focus
  - Therefore the aperture has to be

Realizing a bokeh

- Bokeh requires that within the image you have both objects in focus and out of focus.
    - Therefore the depth-of-field has to be **small**
        - Remember that objects within the depth-of-field are in focus
    - Therefore the aperture has to be **large**
        - As aperture goes to 0, depth-of-field goes to infinity
        - This is the pinhole camera model, every point in the scene is mapped to a single point in the image plane
    - Therefore the lens has to be

Realizing a bokeh

- Bokeh requires that within the image you have both objects in focus and out of focus.
  - Therefore the depth-of-field has to be **small**
    - Remember that objects within the depth-of-field are in focus
  - Therefore the aperture has to be **large**
    - As aperture goes to 0, depth-of-field goes to infinity
    - This is the pinhole camera model, every point in the scene is mapped to a single point in the image plane
  - Therefore the lens has to be **large**
    - The aperture is limited by the lens size
    - You will need a large (and expensive) camera lens!
- Could we digitally create a bokeh effect using a small lens, such as the one found in a smartphone?
  - The answer is: yes, of course!
  - And your task is to do it.

The idea

- A smartphone has a large depth of field
- You will take a series of picture instead of a single one
- You will pick an object, that you want in focus
- You will track that object across all frames, and measure its displacement in different frames
- You will use this displacement to combine the series of images into a single one that is sharp *only at the object of interest*
- This will recreate a “bokeh”

Logistics

- It will be released tonight, on CCLE
- You have 2 weeks to complete it (due on 11/15)
- You will be provided with a single .pdf, that contains all instructions
- You will need to fill out the .pdf (like you would an exam) with answers and figures
  - To do so you will need to write Python code
  - You can use **whatever packages you want**
- You will submit your code and edited .pdf on CCLE
  - You can structure your code **however you want**
  - We will just compare your files

Step 1: setting up a scene

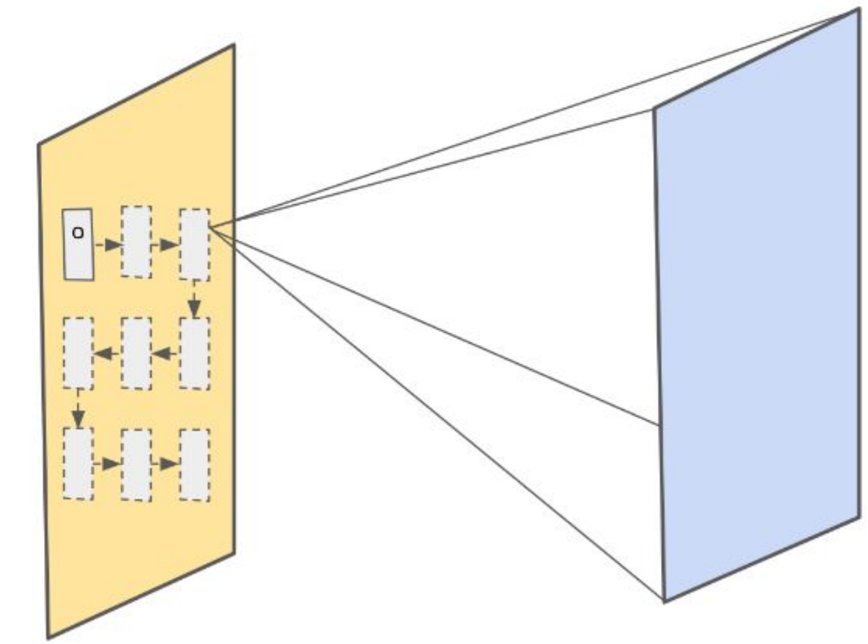
- Your first step will be to take a video of a **static** scene using your smartphone
  - A scene is static if nothing within it is moving
  - Otherwise the scene is called dynamic
- What are the requirements for the scene?
  - Have a few objects (at least 3) at **different depths**
    - If all objects are at the same depth, adjusting the depth-of-field will not change focus!
  - An example:

(a) All-in focus image.

(b) Post-processed to blur the background

Step 2: Process the video

Camera motion:



You will extract frames from the video

- A frame is just an image within the video

Step 3: Template matching

- You now have a set of images of the same scene from different viewpoints
- You would like to:
  - Select an object in the first image
  - Identify it in all subsequent images (to match it)
- To do so, you will use a naive approach, called template matching
  - This is for simplicity, and should be good enough for the current problem
  - Ideally, you would SIFT + RANSAC, but this adds a lot of work
- Template matching is easy:
  - Select a rectangular subset of pixels in the first image (template)
  - Find the maxima of the normalized-cross correlation of the template in subsequent images

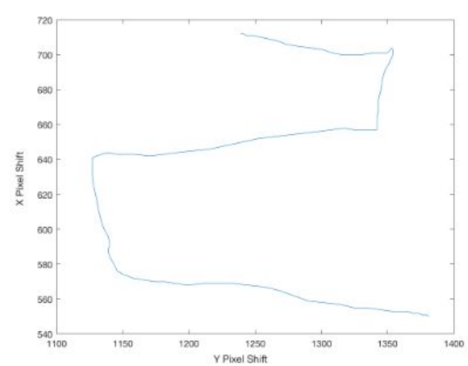
Step 3: Illustration

To track the helmet, you will select the template (in red)

You will overlay the template in all positions within a window (optional) in all subsequent images, and find the position that 'best fits'.

This gives you a shift (in pixels) of the template.

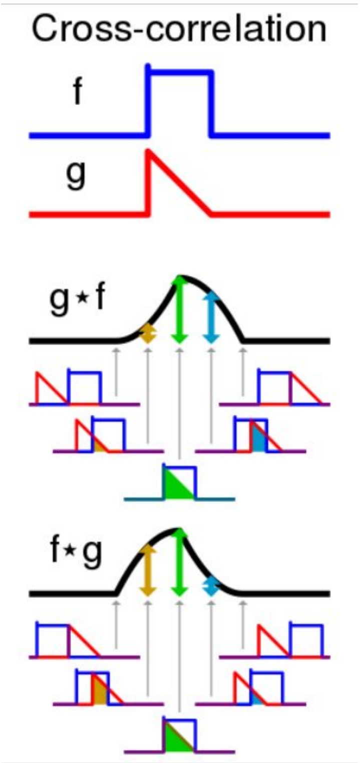
You can plot that shift

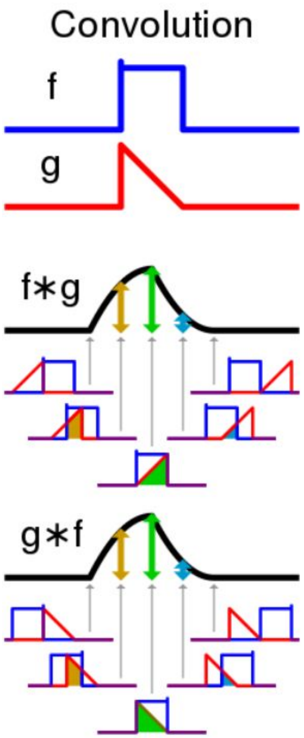


Cross-Correlation

- Very similar to a convolution:

$$(f \star g)(\tau) \triangleq \int_{-\infty}^{\infty} \overline{f(t-\tau)} g(t)\, dt \qquad (f * g)(t) \triangleq \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau.$$





(Normalized) Cross Correlation (NCC)

- Generally used to measure the similarity of 2 signals
  - One signal ‘slides’ across the other, at every position the area under the curve is computed
  - The maximum value gives us a time difference that best aligns thee signals
- For template matching: one signal is shorter than the other. We try to find where the shorter signal ‘best fits’ in the longer signal
  - In your case: where does the template ‘best fit’ in the image
  - To get a metric of ‘fit’ you will use NCC.
  - Why not just CC?

(Normalized) Cross Correlation (NCC)

- Generally used to measure the similarity of 2 signals
  - One signal ‘slides’ across the other, at every position the area under the curve is computed
  - The maximum value gives us a time difference that best aligns thee signals

- For template matching: one signal is shorter than the other. We try to find where the shorter signal ‘best fits’ in the longer signal
  - In your case: where does the template ‘best fit’ in the image
  - To get a metric of ‘fit’ you will use NCC.
  - Why not just CC?
    - Illumination changes can cause a big issue!

Step 4: Defocusing

- Let’s assume that you correctly tracked the movement of the template in your series of images, so that your pixel shifts are correct. You now want to ‘blur’ your image, save for the template. How would you achieve this?

Step 4: Defocusing

- Let's assume that you correctly tracked the movement of the template in your series of images, so that your pixel shifts are correct. You now want to 'blur' your image, save for the template. How would you achieve this?
- First, ask yourself: for what points in the scene are the pixel shifts you found accurate?

Step 4: Defocusing

- Let's assume that you correctly tracked the movement of the template in your series of images, so that your pixel shifts are correct. You now want to 'blur' your image, save for the template. How would you achieve this?
- First, ask yourself: for what points in the scene are the pixel shifts you found accurate?
  - Points at the same depth!
  - Points that are closer shifted ...
  - Points that are further shifted ...

Step 4: Defocusing

- Let’s assume that you correctly tracked the movement of the template in your series of images, so that your pixel shifts are correct. You now want to ‘blur’ your image, save for the template. How would you achieve this?
- First, ask yourself: for what points in the scene are the pixel shifts you found accurate?
    - Points at the same depth!
    - Points that are closer shifted by fewer pixels
    - Points that are further shifted by more pixels

Step 4: Defocusing

- Let's assume that you correctly tracked the movement of the template in your series of images, so that your pixel shifts are correct. You now want to 'blur' your image, save for the template. How would you achieve this?
- First, ask yourself: for what points in the scene are the pixel shifts you found accurate?
  - Points at the same depth!
  - Points that are closer shifted by fewer pixels
  - Points that are further shifted by more pixels
  - This is why you need objects at different depths in your scene!
- What would happen if you superimposed all images, shifted *in the opposite direction* of the pixel shifts?

Step 4: Defocusing (cont)

- You would compute an average of shifted pixel values
- What pixel values would be the same across all images?

Step 4: Defocusing (cont)

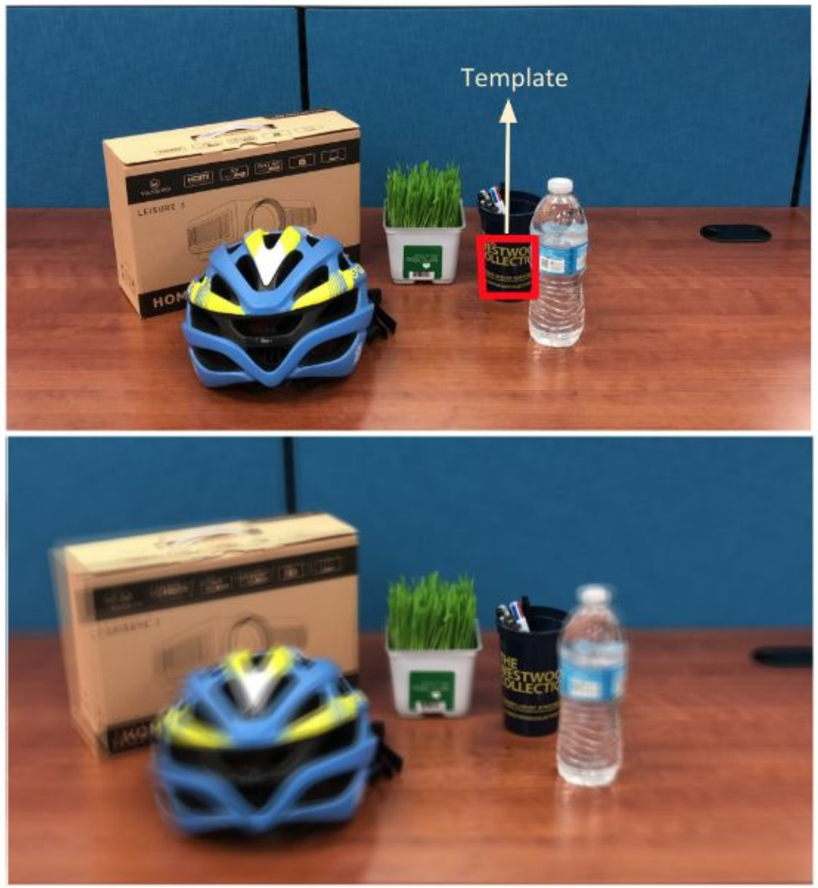
- You would compute an average of shifted pixel values
- What pixel values would be the same across all images?
  - Those in your template! By moving in the opposite direction you 'cancelled' the shift calculated *for these pixels*
- What about the other pixels?

Step 4: Defocusing (cont)

- You would compute an average of shifted pixel values
- What pixel values would be the same across all images?
  - Those in your template! By moving in the opposite direction you 'cancelled' the shift calculated *for these pixels*
- What about the other pixels?
  - They shifted more, or less, depending on depth, so they won't match up across images
  - You're computing an average of pixel values in a neighborhood
  - This is blurring!
- You therefore achieve a synthetic bokeh effect!
  - In your original images, everything is in focus
  - In the output, only the template is in focus

Step 4: Illustration

Step 4: Another template



Step 5: Theory

- The last step is a series of short mathematical derivations, for you to explain exactly *why* this works
  - You will be guided and be able to answer directly on the pdf.

- More on that next discussion, I don't want to spoil anything ...