EE 046746 - Technion - Computer Vision

Tutorial 10 - Structure From Motion

Elias Nehme

Agenda

SfM Intro

Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

Elad Osherov Computer Vision

Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

Elad Osherov Computer Vision

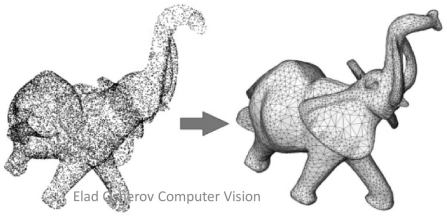
SfM Intro

Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

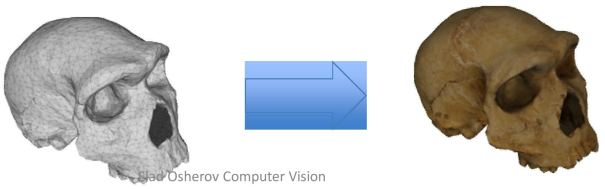
Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



SfM Intro

- Reconstruction = Two-view Structure from Motion (SfM)

	Structure (scene geometry)	Motion (camera geometry)	Measurements
Pose Estimation	known	estimate	3D to 2D correspondences
Triangulation	estimate	known	2D to 2D coorespondences
Reconstruction	estimate	estimate	2D to 2D coorespondences

Pose Estimation

Pose Estimation = Estimating M

Given a set of matched points

$$\{\boldsymbol{P_i}, \boldsymbol{p_i}\}$$

point in 3D space — point in the image

and camera model

$$\boldsymbol{p} = \underbrace{\boldsymbol{f}}_{\text{projection model}}(\boldsymbol{P}; \underbrace{\boldsymbol{m}}_{\text{parameters}}) = \underbrace{\boldsymbol{M}}_{\text{Camera matrix}} \boldsymbol{P}$$

Find the (pose) estimate of

$$\boldsymbol{M}$$

We'll use a **perspective** camera model for pose estimation

Pose Estimation = Estimating M

- Rearrange into a matrix for N_p points:

$$\begin{bmatrix} P_i^T & 0^T & -\tilde{x}_i P_i^T \\ 0^T & P_i^T & -\tilde{y}_i P_i^T \\ . & . & . \\ . & . & . \\ P_{N_p}^T & 0^T & -\tilde{x}_{N_p} P_{N_p}^T \\ 0^T & P_{N_p}^T & -\tilde{y}_{N_p} P_{N_p}^T \end{bmatrix} \begin{bmatrix} | \\ m_1 \\ | \\ m_2 \\ | \\ m_3 \\ | \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 0 \\ 0 \end{bmatrix} \leftrightarrow Am = 0$$

- What are the dimensions? How much points N_p do we need?

Pose Estimation = Estimating M

- rewrite M:

$$M = K\,[R|t] = K\,[R|-Rc] = [N|-Nc]$$

- c can be found via SVD of M due to the relation:

$$Mc = 0$$

- Then N can be further decomposed into $N = KR$:
  - How? using QR decomposition because K is upper triangular and R is orthogonal
- However..

- Does not take into account noise, radial distortions, hard to impose prior knowledge (e.g. f), etc.
- Solution? Minimize Reprojection errors = non-linear least squares

Pose Estimation = Estimating M

- Assuming the camera intrinsics are calibrated (K is known), how much DoF do we have?
- How much points N_p do we need in this case?
- This problem is usually referred to as Perspective-n-Point (PnP).
- Can be solved in openCV using `cv2.solvePnP()`

In [1]:
```
import numpy as np
import cv2
import glob

# load the camera calibration results from tut 9: K (mtx) and distortion coeffs.
with np.load('tut9_camera_calib.npz') as X:
    mtx, dist, _, _ = [X[i] for i in ('mtx','dist','rvecs','tvecs')]
```

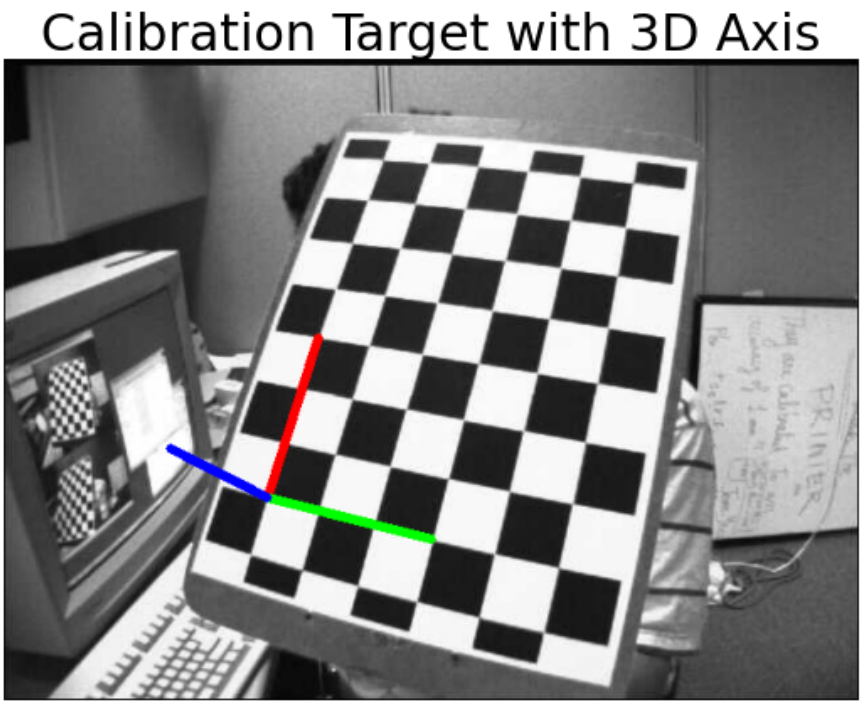
In [2]:
```
# function that draws 3D axis at given image points starting from (0,0)
def draw(img, corners, imgpts):
    corner = tuple(corners[0].ravel())
    img = cv2.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0), 5)
    img = cv2.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0), 5)
    img = cv2.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255), 5)
    return img
```

In [3]:
```
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# prepare object pts
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
# define points of 3D axis (length=3 & origin at first detected corner)
axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
```

In [4]:
```
# plot 3D axis on each image (X = Blue, Y = Green, Z = Red)
k = 0
for fname in glob.glob('./assets/left*.jpg'):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
    if ret == True:
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        # Find the rotation and translation vectors.
        ret, rvecs, tvecs = cv2.solvePnP(objp, corners2, mtx, dist)
        # project 3D points to image plane
        imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
        img = draw(img,corners2,imgpts)
        k += 1
        if k == 8:
            imexample = img
        cv2.imshow('img',img)
        cv2.waitKey(500)
cv2.destroyAllWindows()
```

In [5]:
```
# show the last image and the projected axis
import matplotlib.pyplot as plt
plt.figure(figsize=(11,7))
plt.imshow(imexample)
plt.axis('off')
```

```
plt.title('Calibration Target with 3D Axis', fontsize=30)
plt.show()
```



In [6]:

```
# modified axis points to be corners of cube
axis = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],
                   [0,0,-3],[0,3,-3],[3,3,-3],[3,0,-3] ])
# new drawing function
def draw(img, corners, imgpts):
    imgpts = np.int32(imgpts).reshape(-1,2)
    # draw ground floor in green
    img = cv2.drawContours(img, [imgpts[:4]],-1,(0,255,0),-3)
    # draw pillars in blue color
    for i,j in zip(range(4),range(4,8)):
        img = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]),(255),3)
    # draw top layer in red color
    img = cv2.drawContours(img, [imgpts[4:]],-1,(0,0,255),3)
    return img
```

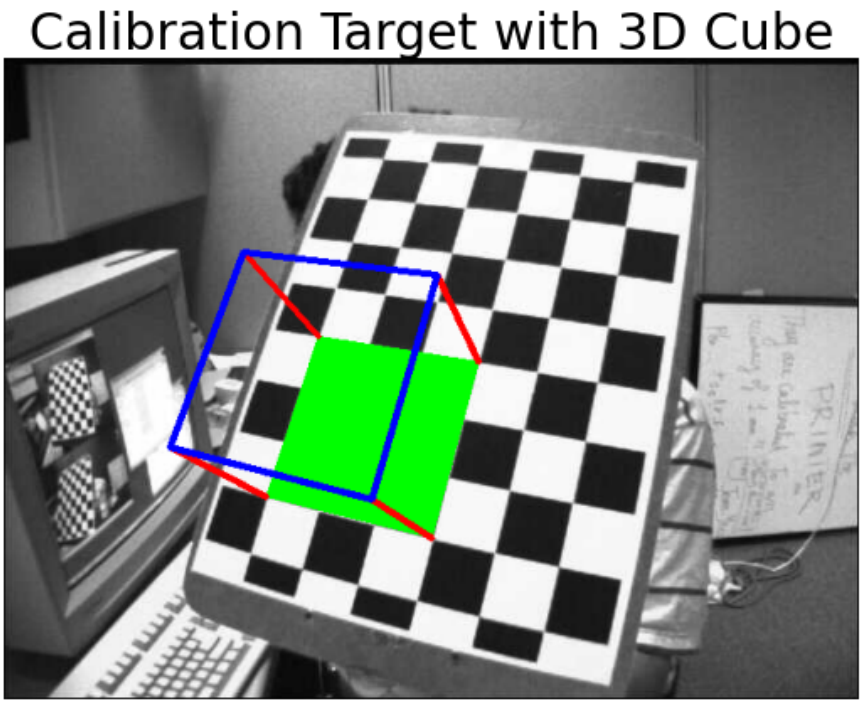
In [7]:

```
# plot 3D cube on each image (X = Blue, Y = Green, Z = Red)
k = 0
for fname in glob.glob('./assets/left*.jpg'):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
    if ret == True:
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        # Find the rotation and translation vectors.
        ret, rvecs, tvecs = cv2.solvePnP(objp, corners2, mtx, dist)
        # project 3D points to image plane
        imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
        img = draw(img,corners2,imgpts)
        k += 1
        if k == 8:
            imexample = img
        cv2.imshow('img',img)
        cv2.waitKey(500)
cv2.destroyAllWindows()
```

In [8]:

```
# show the last image and the projected cube
import matplotlib.pyplot as plt
plt.figure(figsize=(11,7))
```

```
plt.imshow(imexample)
plt.axis('off')
plt.title('Calibration Target with 3D Cube', fontsize=30)
plt.show()
```



Triangulation

Triangulation

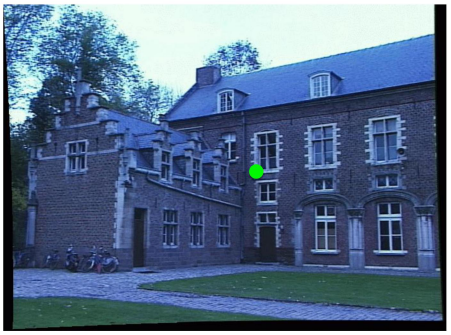
How would you reconstruct 3D points?

Left image

Right image

Triangulation

How would you reconstruct 3D points?



Left image

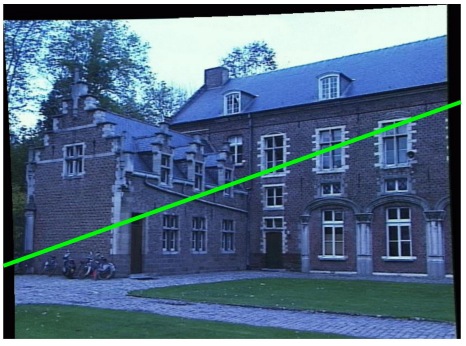
Right image

1. Select point in one image (how?)

Triangulation

How would you reconstruct 3D points?

Left image



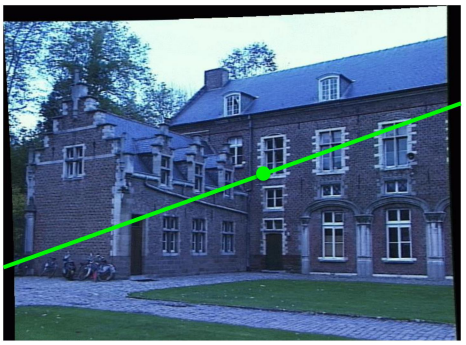
Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)

Triangulation

How would you reconstruct 3D points?

Left image

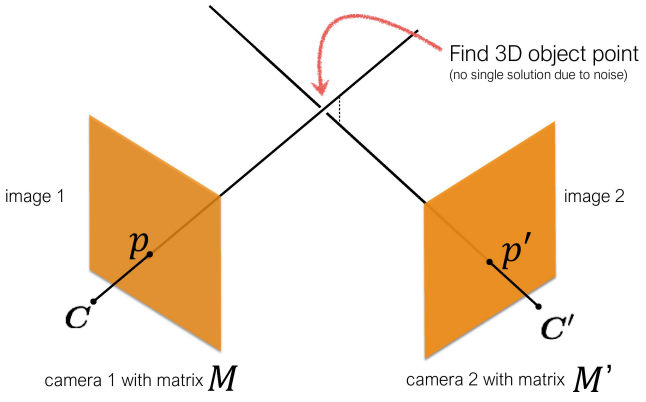


Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)
3. Find matching point along line (how?)

Triangulation

Triangulation



Triangulation

- Switch to row-wise representation of the projection matrix:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} - & m_1^T & - \\ - & m_2^T & - \\ - & m_3^T & - \end{bmatrix} P$$

- Use the fact that the cross product should be zero:

$$p \times MP = 0$$

Triangulation

- Leads to two independent equations:

$$p \times MP = 0 \rightarrow \begin{bmatrix} ym_3^T P - m_2^T P \\ m_1^T P - xm_3^T P \end{bmatrix} = 0$$

- Using the matched point in image 2 as well $\rightarrow$ Solution by SVD:

$$\begin{bmatrix} ym_3^T - m_2^T \\ m_1^T - xm_3^T \\ \tilde{y}\tilde{m}_3^T - \tilde{m}_2^T \\ \tilde{m}_1^T - \tilde{x}\tilde{m}_3^T \end{bmatrix} P = 0 \leftrightarrow AP = 0$$

Reconstruction

Reconstruction

Reconstruction

(2 view structure from motion)

Given a set of matched points

$$\{p_i, p_i'\}$$

Estimate the camera matrices

M, M' ← 'motion' (of the cameras)

Estimate the 3D point

P ← 'structure'

Reconstruction

Two-view SfM

1. Compute the Fundamental Matrix **F** from points correspondences
   8-point algorithm
2. Compute the camera matrices **M,M'** from the Fundamental matrix
   M = [**I** | **0**] and **M'** = [V(**F**) | **e'**]
3. For each point correspondence, compute the point **P** in 3D space (triangularization)
   DLT with p = **M** P and p' = **M'** P

Reconstruction - Pose ambiguity

Camera matrices from essential matrix

$$E = [t_\times]R \qquad E = R[dC_\times]$$

SVD: $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ Let $\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

We get FOUR solutions:

$$M' = [R|T]$$

$$\mathbf{R}_1 = \mathbf{U}\mathbf{W}\mathbf{V}^\top \quad \mathbf{R}_2 = \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top \qquad \mathbf{T}_1 = U_3 \quad \mathbf{T}_2 = -U_3$$

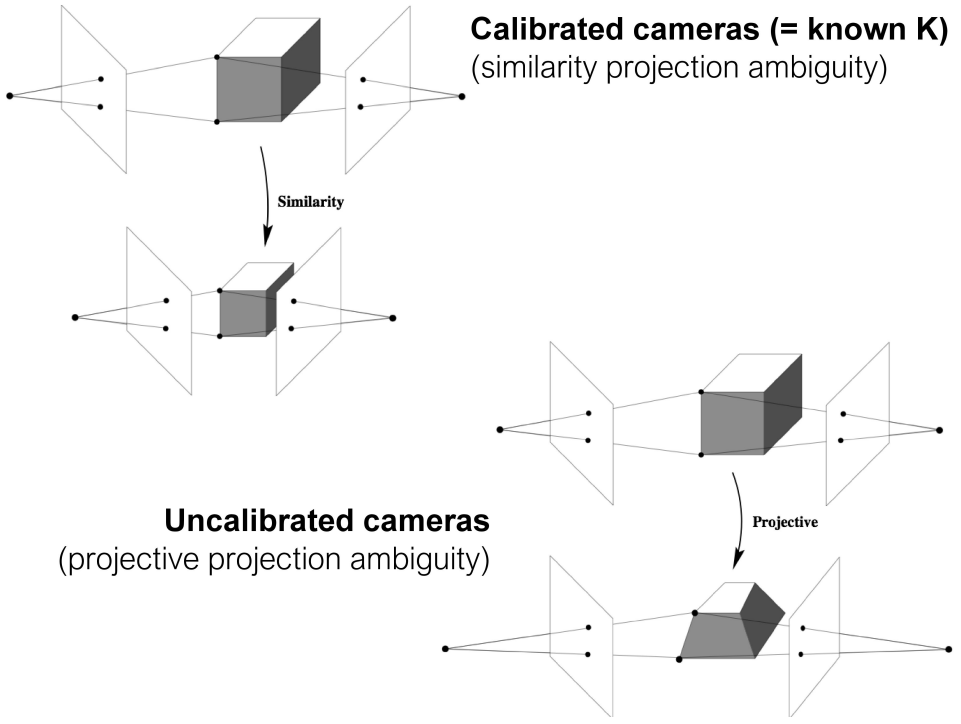
two possible rotations — two possible translations

(See Hartley and Zisserman C.9 for proof)

Reconstruction - Pose ambiguity

- Step 2 is ambiguous!
- Assuming calibrated camera intrinsics (K,K' known) we have 4 options for M,M'.
- This happens because a projection matrix models also $z < 0$, although not phyiscally feasible.
- To decide between the different options we choose the one with 3D pts in front of the camera.

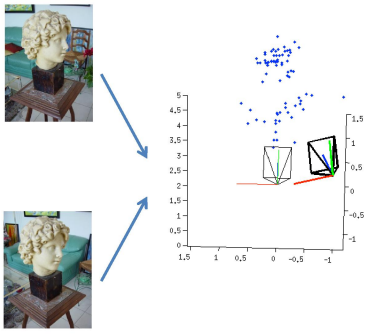
Reconstruction - Scene ambiguity



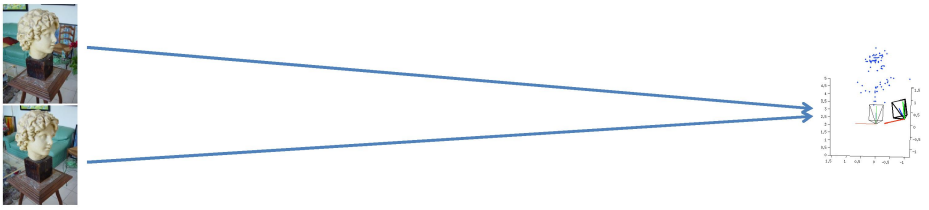
SfM Pipeline Summary

SfM Pipeline Summary

Two-view Reconstruction

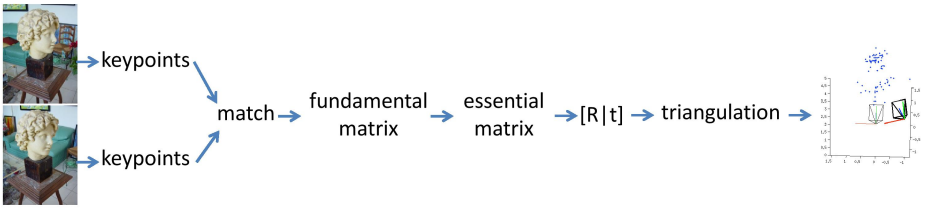


Two-view Reconstruction



SfM Pipeline Summary

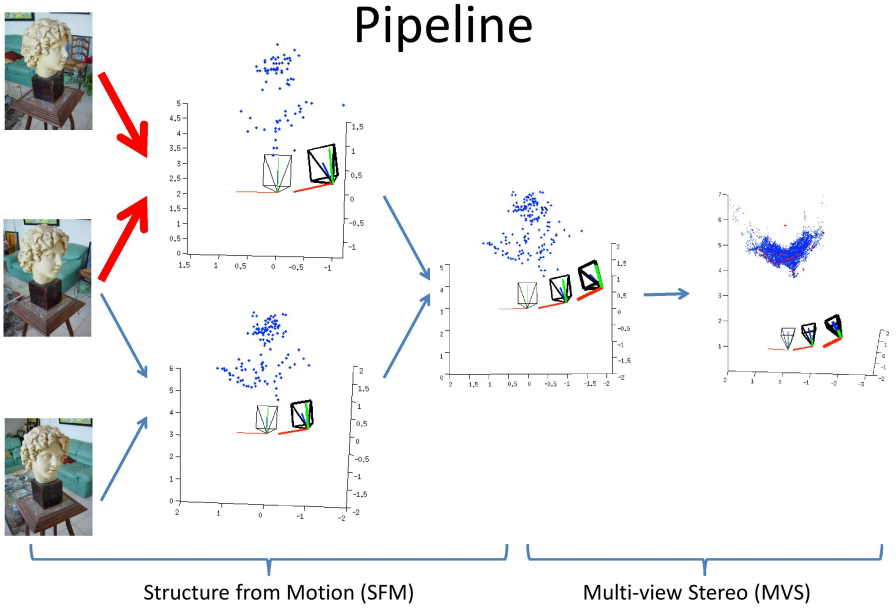
Two-view Reconstruction



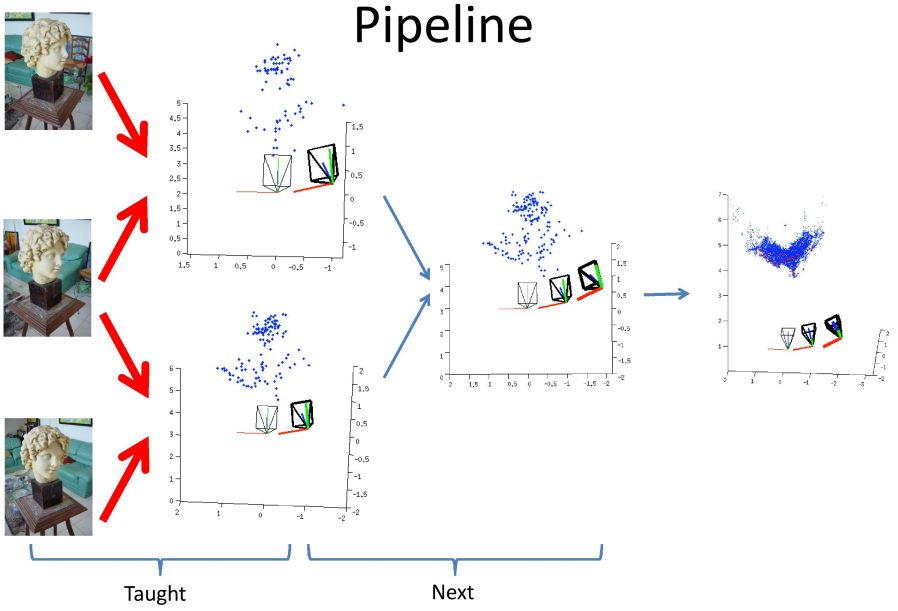
Multiview SfM

Multiview SfM

Pipeline

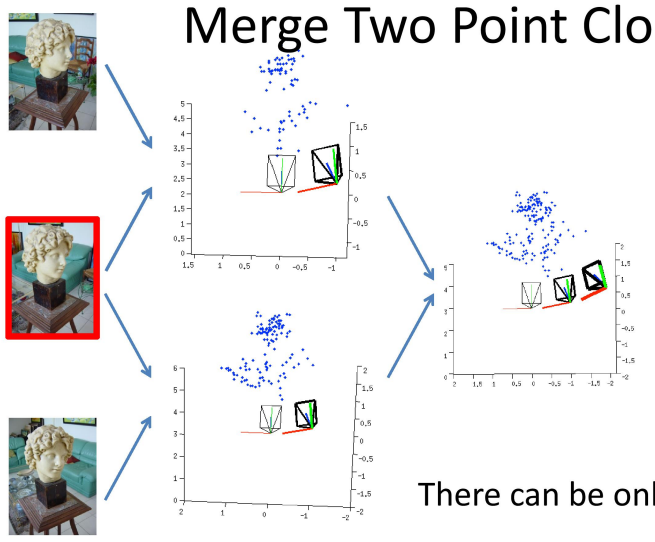


Pipeline



Multiview SfM

Merge Two Point Cloud



There can be only one $\left[\mathbf{R}_2|\mathbf{t}_2\right]$

Bundle Adjustment

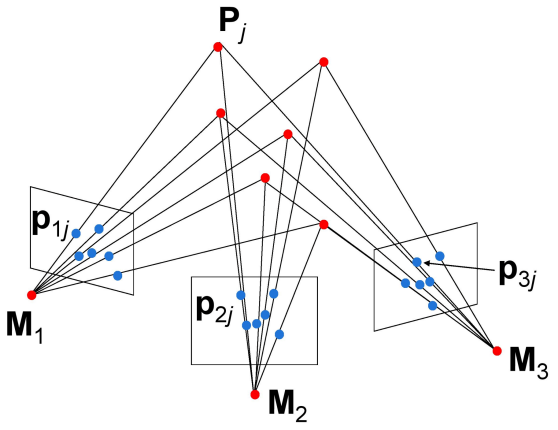
Bundle Adjustment

Projective structure from motion

- Given: *m* images of *n* fixed 3D points

$$z_{ij}\,\mathbf{p}_{ij} = \boldsymbol{M}_i\,\mathbf{P}_j, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n$$

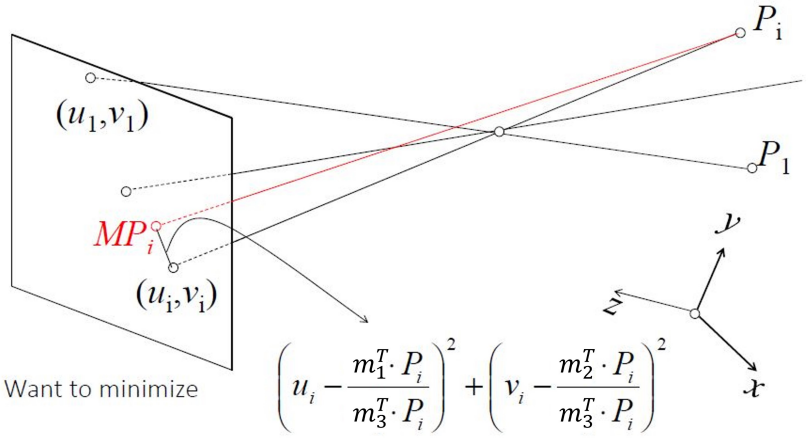
- Problem: estimate *m* projection matrices $\mathbf{P}_i$ and *n* 3D points $\mathbf{P}_j$ from the *mn* correspondences $\mathbf{p}_{ij}$



Bundle Adjustment

- We assume we have m cameras observing a **stationary** 3D scene of n points.
- Not all points (landmarks) are observed in each image. Can happen due to occlusions or noise (e.g. missed detections).
- The objective function is given by minimizing the re-projection error for all observed projections of the n pts.

Bundle Adjustment



Bundle Adjustment

- Minimization problem, assuming a calibrated moving camera with intrinsics K:

$$\hat{R}_i, \hat{t}_i, \hat{P}_j = \arg\min_{R_i,t_i,P_j} \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} \| p_{ij} - K \left[R_i | t_i \right] P_j \|_2^2$$

where $\alpha_{ij} = 1$ if point j is present in camera i and zero otherwise.

Bundle Adjustment

- The resulting problem is non-linear in $M_1, \ldots, M_m$ and $P_1, \ldots, P_n$.
- Solution obtained through some greedy optimization algorithm (e.g. Levenberg-Marquardt).
- Initialization matters in this case!
  - Good initialization = Good local minima.

Bundle Adjustment

- The amount of unknowns is 11 per camera matrix M_i, and 3 per 3D point P_j.
- This is all true up to a projective transform Q with 15 DoF.
- Since every 2D-2D correpondence gives 2 constraints, we end up with the condition:

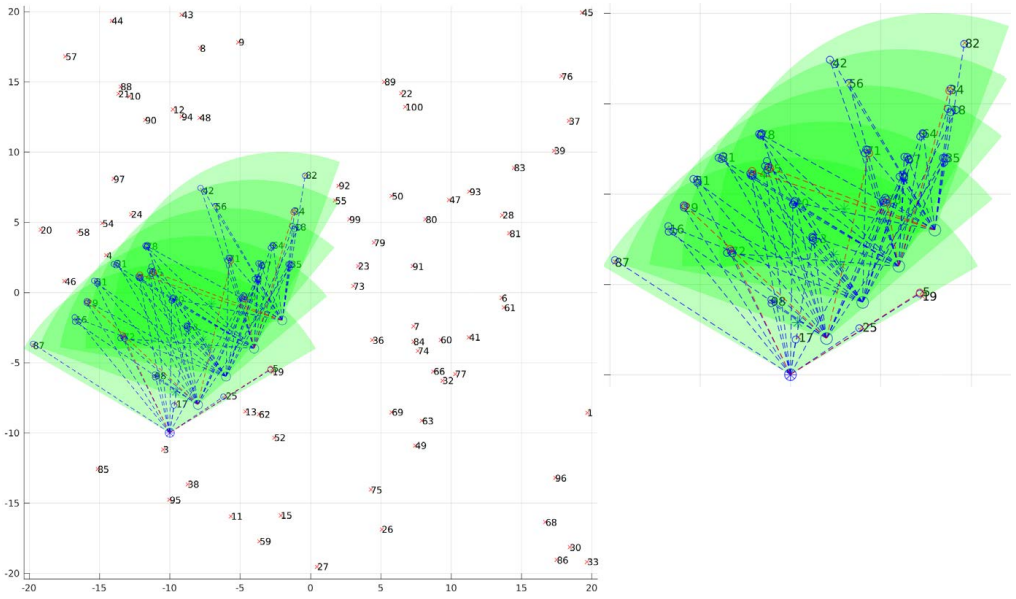
$$2mn \geq 11m + 3n - 15$$

Incremental Smoothing and Mapping

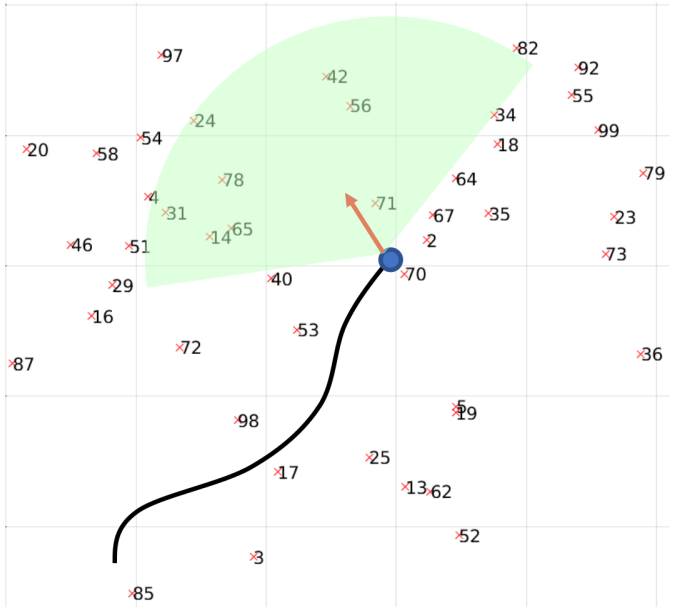
Incremental Smoothing and Mapping

- Bundle Adjustment is extremely slow for large-scale problems (8-10 images can take up to 8-10 hours).
- In 2012, Prof. Frank Dellaert published a hallmark paper called GTSAM for solving "Pose Graph Optimization" problems.
- The solution relies on the concept of Factor Graphs, which we will not go into in this class.

Incremental Smoothing and Mapping



Incremental Smoothing and Mapping



Incremental Smoothing and Mapping

- If you're interested in this topic, consider taking the VAN class by Prof. Vadim Indelman at Aerospace Engineering.
- You can read more about it here.
- Recommended Software: GTSAM package with a python wrapper.

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.

- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Computer Vision First Principles - Shree Nayar
- Structure from Motion Pipeline - Calle Olsson

Credits

- GTSAM blog in Computer Vision Class (U. Maryland) - Nitin J. Sanket
- Slides (Princeton) - Jianxiong Xiao
- Slides (CS Technion) - Elad Osherov
- Slides (CMU) - Ioannis Gkioulekas, Kris Kitani, Srinivasa Narasimhan
- Multiple View Geometry in Computer Vision - Hartley and Zisserman - Chapter 9, 18
- Computer Vision: Algorithms and Applications - Richard Szeliski - Chapter 7

- Icons from Icon8.com - https://icons8.com