Contents

1 Reconstruction Overview

1.1 From Single Image

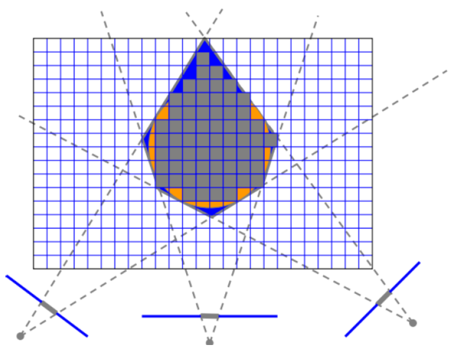
3D reconstruction from a single image is an extremely ill-posed problem as it's not constrained at all and can have infinite solutions that generate the same input image. So this type of reconstruction algorithm are often:

- **Prior-based**. Using DL models trained on specifically designed dataset to "guess" the 3D model.
- **Shape from shading**. Reconstruct the visible part of the 3D model recorded in the input image by inverting the rendering pipeline.

1.2 From Multiple Images

1.2.1 Visual Hull Carving

Visual hull is a Shape-From-Silhouette approach that creates a 3D representation of an object by its silhouettes within several images from different viewpoints. Each of these silhouettes by different camera views form in their projection a cone, called **visual cone**, and an intersection of all these cones form a description of the real object's shape. The below figure illustrates the reconstruction result for 2D object and in general the final result is improved by applying various smoothing kernels:



In order to extract the silhouettes the foreground (object) need to be segmented, i.e. separated from the background. Normally this method assumes an indoor environment with static light, static cameras and static background (greenscreen).

Besides of the RGB data we can also incorporate depth data in order to add more constraints and thus get a better result.

In general, this method:

- Is very easy to compute.
- Requires calibrated setups.
- Either background or foreground has to be segmented in order to extract the silhouette.
- Is not very robust to noise.

1.3 Structure from Motion (SfM)

The main idea of this approach is to simultaneously estimate both 3D points and camera positions. In general this approach follows the following steps:

1. Take set of images.
2. Detect key points in each image.
3. Find correspondences between images (i.e., match key points).
4. Given all matches, jointly solve for T_i and 3D location of points X_j for each image I_i.

1.3.1 Keypoints

Find correspondences for every pixel is in general not possible. So what we like to do is to detect a sparse set of discriminative keypoints and match these keypoints instead. Discriminative means that the same keypoint **P** taken from a viewpoint V_A must also be a keypoint for all other viewpoints in which **P** is visible. Keypoints are often determined by examining local gradients and **corner points are often a good choice as it defines a local maximum/minimum**.

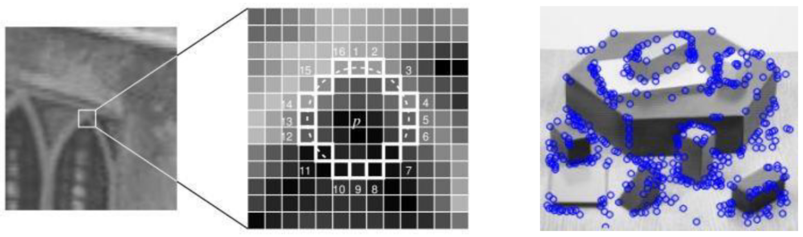
There are a bunch of keypoint detectors:

- Harris corner detector.
- FAST corner detector.
- Shi-Tomasi corner detector.
- SIFT detector.
- etc.

However in this lecture we are only going to discuss 2 of them: Harris and FAST.

1.3.1.1 FAST Keypoint Detector

The FAST keypoint detector uses a circular kernel (only the boundary is considered) of radius **N** (in pixels) to determine whether a pixel is a corner or not. If a set of **k** contiguous pixels at the boundary of the circle are brighter than the current pixel's intensity plus a threshold value **t** or darker than the current pixel's intensity minus **t**, then the current pixel is classified as corner.



$\geq$ 12 contiguous pixels brighter than p + thresh

1.3.1.2 Harris Corner Detector

The key idea behind the Harris corner detector is that there should be large variations in the neighborhood of the corner in all directions. So if we put a small window on a corner **P**, then shifting the window in **any direction** should yield a large change in appearance. Observe that this only occurs at corners as:

- For points in the flat region there's no change in all directions.
- For edge points there's no change along the edge direction.

Harris detector uses the following energy function to measure the change produced by a shift (u, v):

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + u) - I(x, y)]^2$$

Where:

- $w(x, y)$ is the window function. 1 if it's in window and 0 otherwise. Or a Gaussian function so it assigns higher weight to nearby neighbors and lower weight to far neighbors.
- $I(x + u, y + u)$ is the shifted intensity.
- $I(x, y)$ is the current intensity.

So for a corner point it must maximizes the sum $E = \sum_{u,v} E_{u,v}$. But actually we can simplify the computation by not computing all the $E_{u,v}$. Assuming small u, v, the above function can be approximated using Taylor series in following way:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Where $M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$ and I_x, I_y are the partial derivatives along axis x and y in the point (x, y). Now the corner response can be computed as:

$$R = det(M) - k(trace(M))^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Where λ_1, λ_2 are the eigenvalues of M and if

- $|R|$ samll $\Rightarrow$ region is flat (λ_1, λ_2 are small), no change in either x or y direction.
- $R < 0 \Rightarrow$ region is edge ($\lambda_1 >> \lambda_2$ or $\lambda_1 << \lambda_2$), there's change in only one direction.
- R is large $\Rightarrow$ corner(λ_1, λ_2 both large), there's change in both directions.

Important, Harris detector is rotation invariant and lighting invariant but not scale invariant.

1.3.2 Keypoint Descriptors and Matching

In order to establish correspondences between keypoints we need somehow to find a distinctive description for each detected keypoint. The assigned description must:

- Can perform fast comparisons.
- Be scale invariant.
- Be view invariant.
- Be lighting invariant.

In general we call these descriptions **features**. The feature of a keypoint is usually defined by it's neighbourhood and consists of a mapping from a high-dimensional space to a lower-dimensional space. There are a lot of feature descriptors for matching, such as:

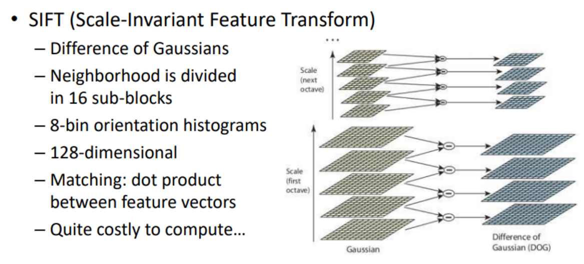
- SIFT
- SURF
- ORB
- FREAK
- Deep learning based descriptors
- etc

In this lecture we are only going to discuss SIFT and give a high level overview of DL based descriptors.

1.3.2.1 SIFT (Scale-Invariant Feature Transform)

SIFT is in fact a keypoint detection + feature descriptor algorithm. It extracts keypoints under different scales by comparing DoGs computed at each scale. In order to do that, a Gaussian filter is used in order to create different level of smoothed/blurred version of the original image by increasing the standard variation, and each consecutive pair creates a DoG. One pixel in a DoG is compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales (scale in terms of σ not the size of the image). This way, a total of 26 checks are made. If it is a local extrema, it is a potential keypoint. Finally, a pruning step is performed in order to remove edge points.

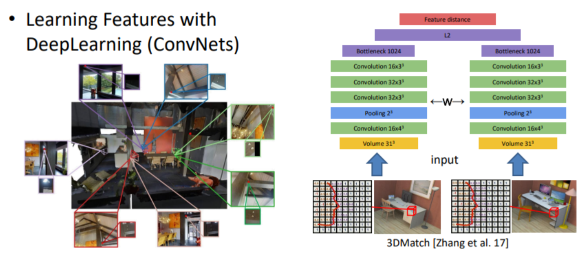
Each detected keypoint is described by its neighborhood using 8-bin HOG. The neighborhood is defined by a 16x16 grid centered in the keypoint and is divided into 16 sub-blocks of size 4x4. Then for each sub-block we extract a 8-bin HOG which means the final composed feature vector is of size 16x8=128.



- SIFT (Scale-Invariant Feature Transform)
  - Difference of Gaussians
  - Neighborhood is divided in 16 sub-blocks
  - 8-bin orientation histograms
  - 128-dimensional
  - Matching: dot product between feature vectors
  - Quite costly to compute…

1.3.2.2 DeepLearning based descriptors

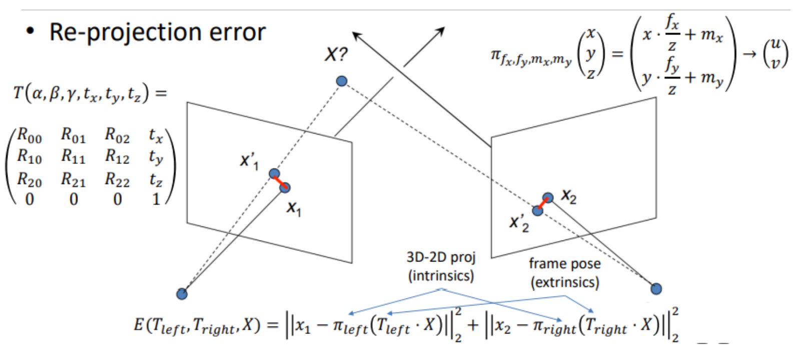
DL-based feature descriptors are often extracted using a Siamese network with the triple loss function.



- Learning Features with DeepLearning (ConvNets)

1.3.3 Bundle Adjustment

Once we have detected keypoints and have established correspondences between them then we can start estimating the 3d points and the camera positions. This problem is called bundle adjustment.



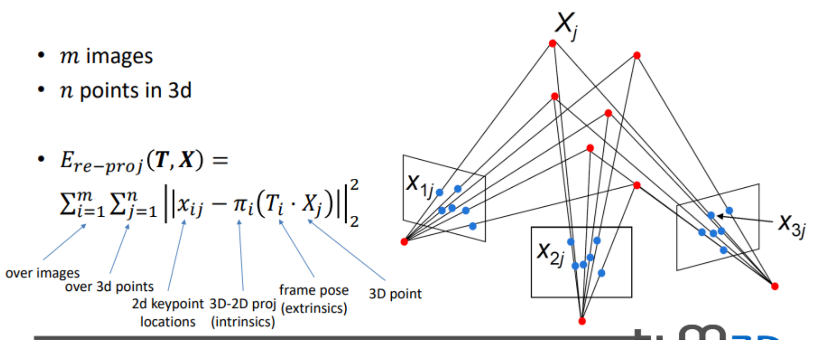
- Re-projection error

$$T(\alpha,\beta,\gamma,t_x,t_y,t_z) = \begin{pmatrix} R_{00} & R_{01} & R_{02} & t_x \\ R_{10} & R_{11} & R_{12} & t_y \\ R_{20} & R_{21} & R_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\pi_{f_x,f_y,m_x,m_y}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \cdot \frac{f_x}{z} + m_x \\ y \cdot \frac{f_y}{z} + m_y \end{pmatrix} \rightarrow \begin{pmatrix} u \\ v \end{pmatrix}$$

$$E(T_{left}, T_{right}, X) = \left\|x_1 - \pi_{left}(T_{left} \cdot X)\right\|_2^2 + \left\|x_2 - \pi_{right}(T_{right} \cdot X)\right\|_2^2$$

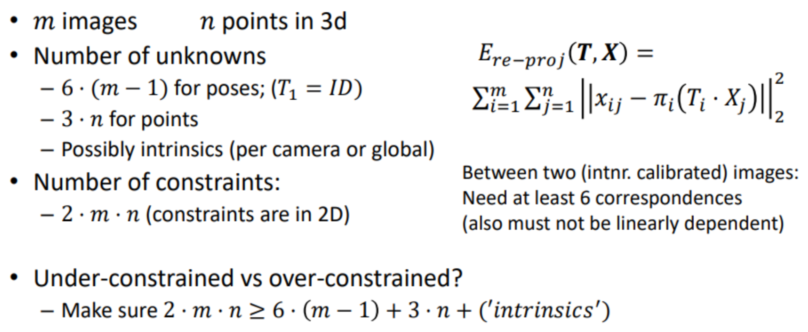
The main idea is to reduce the re-projection error. The estimated 3d points and camera positions will be used to compute the estimated projected 2d points. These 2D points will be then compared to the ground truth and our object is to reduce the error between them. We have to reduce the error over all points over all images:



- m images
- n points in 3d
- $E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) =$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\left\|x_{ij} - \pi_i(T_i \cdot X_j)\right\|_2^2$$

The constraints and number of unknowns are described in the following picture:



- m images n points in 3d
- Number of unknowns
  - $6 \cdot (m-1)$ for poses; ($T_1 = ID$)
  - $3 \cdot n$ for points
  - Possibly intrinsics (per camera or global)
- Number of constraints:
  - $2 \cdot m \cdot n$ (constraints are in 2D)

$$E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left|\left|x_{ij} - \pi_i(T_i \cdot X_j)\right|\right|_2^2$$

Between two (intnr. calibrated) images:
Need at least 6 correspondences
(also must not be linearly dependent)

- Under-constrained vs over-constrained?
  - Make sure $2 \cdot m \cdot n \geq 6 \cdot (m-1) + 3 \cdot n + ('intrinsics')$

In general we have to make sure that our problem is over-constrained, i.e. the number of constraints is significantly larger than the number of unknowns. Also, we have to make sure that keypoints are not linearly dependent because otherwise all the terms will have the same error and the optimization won't work.

There are basically two key challenges in SfM:

- Find good feature matches
- Find good initialization for optimization

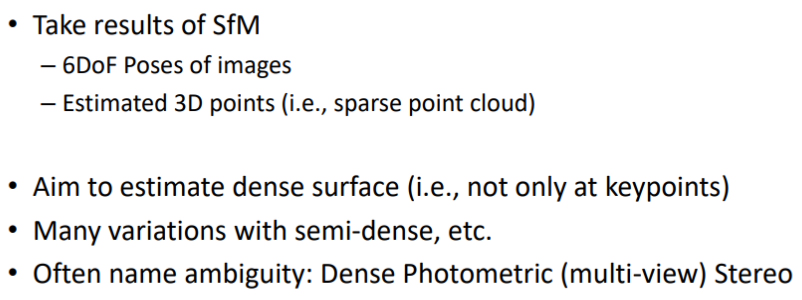
And also **keep in mind that SfM only estimates a sparse point cloud as it only focus on keypoints**. Some of the applications of SfM are:

- **Building Rome in a Day**. Reconstruct a city from a set of images.
- **Photo Tourism**. Based on the selected keypoints extract from the dataset images that contains these keypoints.
- **Hyperlapse**. Stabilize videos recorded from carry-on devices like GoPro.

1.4 Multi-view Stereo (MVS)

SfM only estimates camera positions and a sparse set of keypoints. But sometimes we would like to estimate dense surface, here is where MVS comes into play.

In general MVS just takes the result of SfM and aim to estimate a much denser surface. There are many variations of this method such like semi-dense surface, etc. And often there is a name ambiguity as is also called Dense Photometric (multi-view) Stereo.



- Take results of SfM
  - 6DoF Poses of images
  - Estimated 3D points (i.e., sparse point cloud)

- Aim to estimate dense surface (i.e., not only at keypoints)
- Many variations with semi-dense, etc.
- Often name ambiguity: Dense Photometric (multi-view) Stereo

The high-level idea behind the simplest MVS is just do dense stereo matching between every frame pair as we already have the alignment and intrinsic from SfM. However, there are other more complex methods such as:

- **Global optimization**. Solve for all disparities simultaneously using MRF.
- **Semi-global Matching**

- Simple idea: just do dense stereo matching between every frame pair (we have the alignment + intrinsic)
  - Somehow ‘vote’ for results
- Global optimization:
  - Solve for all disparities simultaneously
  - Can be mapped to a Markov Random Field problem (MRF)
    - Matching cost term and pairwise terms
- Semi-global Matching (SGM) [Hirschmuller 08]
  - Approximate for tractable runtime

In simplest words the goal of MVS is to create an as-dense-as-possible point cloud. And once we have the point cloud we will follow the following pipeline to get the final reconstruction:

- Use (dense) point cloud as input
  - Compute normals (e.g., using PCA)
- Feed into some surface fitting / surface optimization
  - E.g., Poisson Surface Reconstruction [Kazhdan et al. 06/13]
- Texture the resulting surface (e.g., optimize for texture warp on top of surface)

The computation of normals through PCA is very straightforward. The normal of a point on the surface can be approximated by the normal of a plane tangent to the surface. Given a point **X**, a robust plane **P** tangent to **X** is a plane that minimizes the sum of squared point-to-plane distances of neighbors of **X**. Thus, the normal estimation problem turn becomes a least-square plane fitting estimation problem, which is what PCA tries to solve.

Once we have found the new orthogonal basis after decompose the covariance matrix computed from **X** and it’s neighbours, the eigen vector associated to the biggest eigen value is the plane tangent to **X** and the eigen vector associated to the smallest eigen value is chosen to be the raw normal of **X**. The raw normal are often ambiguous and not consistently oriented over the entire point cloud, so one might have to use techniques like propagation to improve the estimated results.

So, as conclusion, a typical pipeline for a 3d reconstruction task from images is **SfM** + **MVS** + **Reconstruction**.

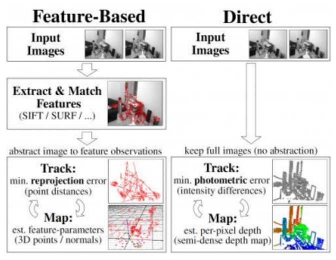
1.5 Simultaneous Localization and Mapping (SLAM)

- Similar idea than SfM + MVS
  - But online; i.e., ‘model’ is built up incrementally
    - Takes real-time RGB stream (e.g., webcam)
  - Typically smaller compute budget
  - Often only frame-to-frame tracking (no global optimization)

3D Scanning & Motion C
Prof. Nießner

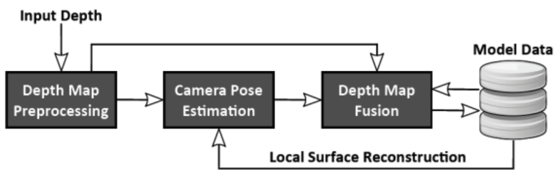
1.5.1 LSD-SLAM

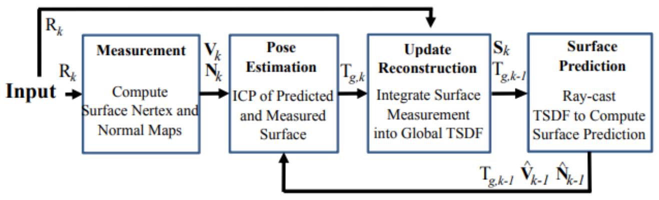
- State-of-the-art SLAM
  - Similar to Bundle Adjustment
  - Extract keyframes (dynamically)
  - BA on local keyframe window
  - Typically ORB features (faster)
- Output
  - Poses + sparse point cloud
  - Various 'semi-dense methods'


Feature-Based
Direct
Input Images
Input Images
Extract & Match Features (SIFT / SURF / ...)
abstract image to feature observations
keep full images (no abstraction)
Track: min. reprojection error (point distances)
Map: est. feature-parameters (3D points / normals)
Track: min. photometric error (intensity differences)
Map: est. per-pixel depth (semi-dense depth map)
LSD-SLAM [Engel et al. 14]

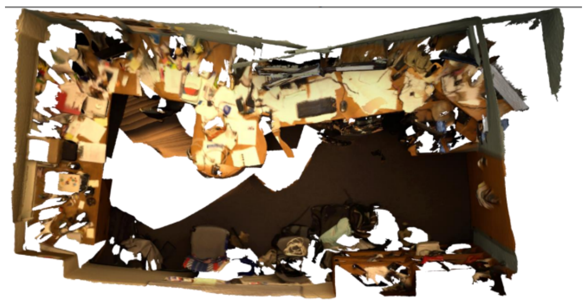
1.6 RGB-D Reconstruction

- We already have depth data (in real time)!
- Typical approach:
  - Frame-to-frame tracking (or frame-to-model)
  - Accumulate depth data in 'model' (implicit surface rep.)




R_k
Input
R_k
Measurement
Compute Surface Nertex and Normal Maps
V_k N_k
Pose Estimation
ICP of Predicted and Measured Surface
$T_{g,k}$
Update Reconstruction
Integrate Surface Measurement into Global TSDF
S_k $T_{g,k-1}$
Surface Prediction
Ray-cast TSDF to Compute Surface Prediction
$T_{g,k-1}$ $\hat{V}_{k-1}$ $\hat{N}_{k-1}$

1.6.1 Loop closure problem



The RGB-D reconstruction implements frame-to-model tracking and every time a frame is aligned to the model a small error is accumulated. This problem can be improved by using RGBD-bundling by incorporating the depth data as well:

$$E_{bundle}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

$$E_{depth}(T) = \sum_{i,j}^{\#\text{frames}} \sum_{k}^{\#\text{pixels}} \left\| \left(p_k - T_i^{-1} T_j \pi_d^{-1}\left(D_j\left(\pi_d\left(T_j^{-1} T_i p_k\right)\right)\right)\right) \cdot n_k \right\|_2^2$$

$$E_{color}(T) = \sum_{i,j}^{\#\text{frames}} \sum_{k}^{\#\text{pixels}} \left\| \nabla I(\pi_c(p_k)) - \nabla I\left(\pi_c\left(T_j^{-1} T_i p_k\right)\right) \right\|_2^2$$

In the first term we can see that there's no projection matrix as in traditional bundle adjustment problem because here we already have the depth data so we can compare points in world space.