On large scale 3D reconstruction from images and videos

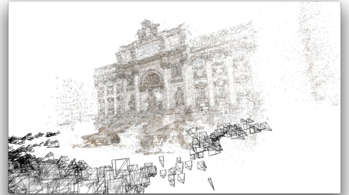
Subhashis Banerjee

Computer Science and Engineering
IIT Delhi

NCVPRIPG 2015, IIT Patna
Dec 16, 2015

Build Rome in a day?

Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz and Richard Szeliski, 2009. Also, Visual Structure from Motion (Wu, 2013).



Large datasets

- 2000 to 30000 images.
- Reconstruction examples:

Applications

- Engineering.
- Dense reconstruction.
- Heritage and tourism.
- Localization and SLAM.
- Mobile and hand-helds?

What makes it possible?

- **Linear perspective. Geometry**.
- Rich feature extraction.
- Feature grouping, clustering and data structures.
- Graph algorithms for image clustering.
- Large scale non-linear optimization.
- Global motion averaging.
- Divide and conquer?

An infinitely strange perspective

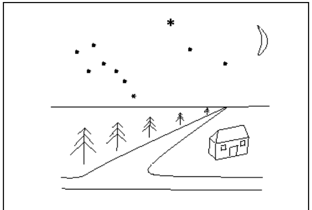


Figure 1.1: Landscape with horizon

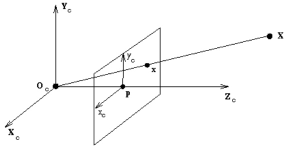
- Parallel lines in 3D space converge in images.
- The line of the horizon is formed by ‘infinitely’ distant points (vanishing points).
- Any pair of parallel lines meet at a point on the horizon corresponding to their common direction.
- All ‘intersections at infinity’ stay constant as the observer moves.

3D reconstruction from pin-hole projections

La Flagellazione di Cristo (1460) Galleria Nazionale delle Marche by Piero della Francesca (1416-1492) (Robotics Research Group, Oxford University, 2000)

Pin-hole camera

- The effects can be modelled mathematically using the 'linear perspective' or a 'pin-hole camera' (realized first by Leonardo?)

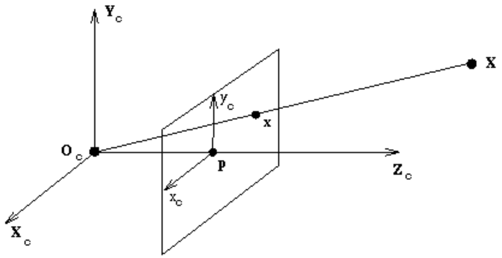


- If the world coordinates of a point are (X, Y, Z) and the image coordinates are (x, y), then

$$x = fX/Z \text{ and } y = fY/Z$$

- **The model is non-linear**.

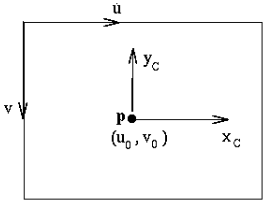
Pin-hole camera revisited



$$\begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = k \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

$$k = f / Z_c$$

Intrinsic/internal parameters

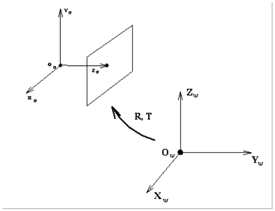


$$\begin{aligned} k_u x_c &= u - u_0 \\ k_v y_c &= v_0 - v \end{aligned}$$

or

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \mathbf{K} \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix}$$

External parameters



$$\mathbf{X_c} = \mathbf{R}\mathbf{X_w} + \mathbf{T}$$

or

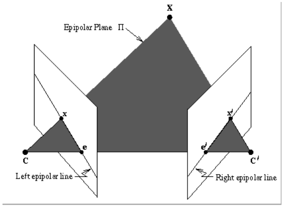
$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Calibrated camera

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K}\,[\mathbf{R} \mid \mathbf{T}] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Image to world correspondences of six points in general position gives camera calibration.

Multiple views: epipolar geometry



epipole: The epipole is the image in one camera of the optical center of the other camera.

epipolar plane: is the plane defined by a 3D point and the optical centers.

epipolar line: is the line of intersection of the epipolar plane with the image plane.

Epipolar constraint

- Any point $\mathbf{x}'$ on this epipolar line satisfies

$$\mathbf{x}'^T \mathbf{F}\mathbf{x} = 0$$

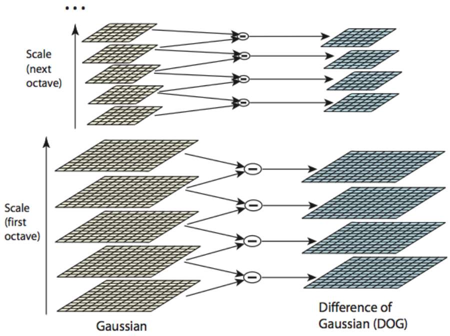
- $\mathbf{F}$ is called the **fundamental matrix**. It is of rank 2 and can be computed from 8 point correspondences.
- $\mathbf{F} = \mathbf{K}'^T \mathbf{E}\mathbf{K}^{-1}$
- $\mathbf{E} = [\mathbf{T}]_\times \mathbf{R}$

What makes it possible?

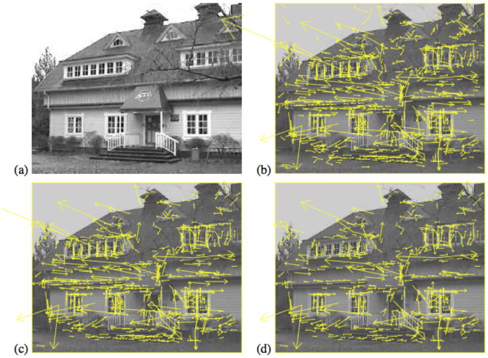
- Linear perspective. Geometry.
- **Rich feature extraction**.
- Feature grouping, clustering and data structures.
- Graph algorithms for image clustering.
- Large scale non-linear optimization.
- Global motion averaging.
- Divide and conquer?

What to match? : SIFT

Features are scale-space maximas in DOGs.

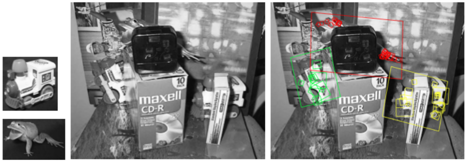


Features: SIFT:example



Lowe, 2004; Wu, 2007 (SiftGPU).

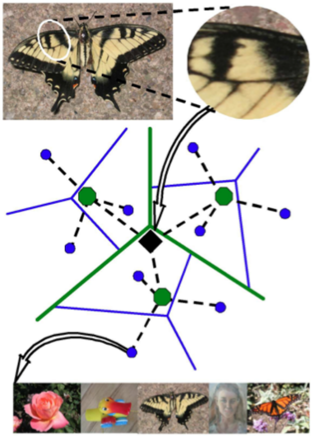
SIFT: matching



What makes it possible?

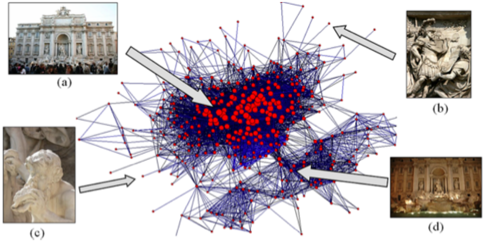
- Linear perspective. Geometry.
- Rich feature extraction.
- **Feature grouping, clustering and data structures**.
- **Graph algorithms for image clustering**.
- Large scale non-linear optimization.
- Global motion averaging.
- Divide and conquer?

Vocabulary tree



Nister and Stewenius, 2006.

Vocabulary tree match graph



What makes it possible?

- Linear perspective. Geometry.
- Rich feature extraction.
- Feature grouping, clustering and data structures.
- Graph algorithms for image clustering.
- **Large scale non-linear optimization**.
- Global motion averaging.
- Divide and conquer?

Sparse bundle adjustment

Simultaneously refine

- 3D coordinates describing the scene geometry
- Parameters of the relative motion.
- Optical characteristics of the camera.

from corresponding image projections of all points. The process minimizes the reprojection error defined by

$$\min_{\mathbf{a_j},\mathbf{b_i}} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} d(\mathbf{P}(\mathbf{a}_j, \mathbf{b}_i), \mathbf{x}_{ij})^2 \tag{1}$$

where, $v_{ij} = 1$ if point i is visible in image j.
$\mathbf{P}$ is the projection matrix, d is the euclidean distance, $\mathbf{x_{ij}}$ is projection of $\mathbf{a_j^{th}}$ point on image $\mathbf{b_i}$

Sparse bundle adjustment

- Require projective invariance to choice of basis
- Measure error in the image plane
- Error measure is defined in terms of “reprojection error”
- Measure distance from point to its reprojected image
- Take a least-squares approach
- Lsq justified as MLE given Gaussian noise in image point location

$$\min_{\mathbf{a_j},\mathbf{b_i}} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} d(\mathbf{P}(\mathbf{a}_j, \mathbf{b}_i), \mathbf{x}_{ij})^2 \qquad (2)$$

Sparse bundle adjustment

- Adjust bundle of rays between each camera centre and 3D points
- Can tolerate missing data (i.e. matches not visible in some images)
- Provides a true MLE despite missing data
- $d(.,.)$ can be modified to incorporate error covariances
- Minimisation of cost function is a complicated problem
- Use general non-linear least-squares minimisation methods
- Levenberg-Marquardt minimisation is the method of choice
- Works very well in practice
- Needs a good initialisation to avoid getting stuck in local minima

Sparse bundle adjustment: nonlinear least squares

- Let the function to be minimised be $f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(\mathbf{x})$, where $\mathbf{x} = (x_1, \ldots, x_n)$, $r_j : \mathbb{R}^n \to \mathbb{R}$ and $r(\mathbf{x}) = (r_1(\mathbf{x}), \ldots, r_m(\mathbf{x}))$.
- Jacobian $J(x) = \frac{\partial r_j}{\partial x_i}$, $1 \leq j \leq m$, $1 \leq i \leq n$.
- Suppose each r_j is linear. Then Jacobian is constant and r is a hyperplane through space. We can write $f(\mathbf{x}) = \frac{1}{2}\|J\mathbf{x} + r(\mathbf{0})\|^2$.
- Then $\nabla f(\mathbf{x}) = J^T(J\mathbf{x} + r)$ and $\nabla^2 f(\mathbf{x}) = J^T J$.
- Solving for the minimum by setting $\nabla f(\mathbf{x}) = 0$, we obtain $\mathbf{x}_{min} = -(J^T J)^{-1} J^T r$ which is the solution to the set of *normal equations*.

Sparse bundle adjustment: nonlinear least squares

- In the general non-linear case we have

$$\nabla f(\mathbf{x}) = \sum_{j=1}^{m} r_j(\mathbf{x}) \nabla r_j(\mathbf{x}) = J(\mathbf{x})^T r(\mathbf{x})$$
$$\nabla^2 f(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \sum_{j=1}^{m} r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x})$$

- Given the Jacobian J, we can get the Hessian $\nabla^2 f(\mathbf{x})$ for free if it is possible to approximate r_js as linear functions ($\nabla^2 r_j(\mathbf{x})$ are small), or if the residuals ($r_j(\mathbf{x})$) themselves are small.

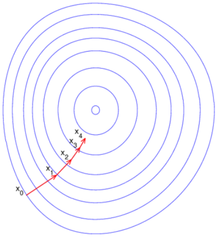
Sparse bundle adjustment

Before we arrive at the Levenberg-Marquardt method,
let's very quickly review

- Gradient Descent
- Newton algorithm

Then we can see how Levenberg-Marquardt skillfully blends two different methods while retaining their advantages

Sparse bundle adjustment



Gradient Descent

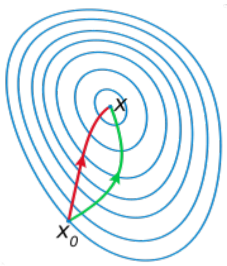
- Given a multivariate function $F(\boldsymbol{x})$
- Start at a point $\boldsymbol{x}_0$
- Move along the direction of the gradient $\nabla F(\boldsymbol{x})$
- Update equation $\boldsymbol{x} \leftarrow \boldsymbol{x} - \lambda \nabla F(\boldsymbol{x})$
- Repeat till convergence (hopefully!)

Sparse bundle adjustment

Limitations of Gradient Descent

- Gradient Descent is simple and easy to implement
- Many iterations to converge if curvature varies in different directions
- Figuring out optimal step-size λ is time consuming
- Small λ controls convergence but is slow
- Large λ results in speed-up but can overshoot
- Step-size is not constant but dependent on gradient magnitude
- Shallow areas result in small steps (bad)
- Conversely, in steep areas one could overshoot
- Consider long valley

Sparse bundle adjustment



$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Newton Algorithm

- Can use *second order* information
- Use curvature along with gradient direction information
- Consider $F(\boldsymbol{x} + \triangle\boldsymbol{x}) = F(\boldsymbol{x}) + F'(\boldsymbol{x})^T \triangle\boldsymbol{x} + \triangle\boldsymbol{x}^T F''(\boldsymbol{x})\triangle\boldsymbol{x}$
- Minimisation done by solving for $\triangle\boldsymbol{x}$
- Update is $\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{H}^{-1}\nabla F(\boldsymbol{x})$

Sparse bundle adjustment

Comparison

$$\begin{aligned} \boldsymbol{x} &\leftarrow \boldsymbol{x} - \nabla F \\ \boldsymbol{x} &\leftarrow \boldsymbol{x} - \boldsymbol{H}^{-1} \nabla F \end{aligned}$$

What does this imply ?

- We approximate $F(\boldsymbol{x})$ locally as quadratic function
- Can jump to local optimum
- If close to solution, we can guess minimum well
- If approximation is good, we converge faster than steepest descent

Sparse bundle adjustment

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - (\boldsymbol{H} + \lambda \boldsymbol{I})^{-1} \nabla F$$

Levenberg's Algorithm

- Can blend Newton's method with steepest descent
- For large λ, rule is $\boldsymbol{x} \leftarrow \boldsymbol{x} - \frac{1}{\lambda} \nabla F$ (s.d.)
- For small λ, rule is the Newton update
- Start with steepest descent and gradually move towards quadratic rule
- Control the shift from one regime to another using error improvement

Sparse bundle adjustment

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - (\boldsymbol{H} + \lambda \boldsymbol{I})^{-1} \nabla F$$

Steps in Levenberg's Algorithm

- Carry out an update using rule
- Evaluate error measure at new location and compare with previous position's error
- If error *increases*, go back to previous position and *increase* λ
- If error *decreases*, keep update and *decrease* λ
- Scaling of λ at each step done using a fixed constant, say 10

Sparse bundle adjustment

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - (\boldsymbol{H} + \lambda \boldsymbol{I})^{-1} \nabla F$$

Justification for Levenberg's Algorithm

- If error increases after step, quadratic approximation is poor
- Need to move towards steepest descent as likely far from minima
- Done by increasing λ
- If error decreases, approximation is good
- Can converge well using quadratic approximation
- Move away from steepest descent approach by decreasing λ

Sparse bundle adjustment

Comparison

$$\begin{aligned} \boldsymbol{x} &\leftarrow \boldsymbol{x} - (\boldsymbol{H} + \lambda \boldsymbol{I})^{-1} \nabla F \quad \textit{Levenberg} \\ \boldsymbol{x} &\leftarrow \boldsymbol{x} - (\boldsymbol{H} + \lambda\, diag(\boldsymbol{H}))^{-1} \nabla F \quad \textit{Marquardt} \end{aligned}$$

What did Marquardt add to this ?

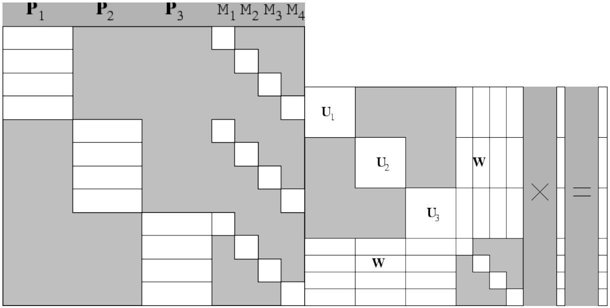
- Insight that incorporated local curvature information
- When λ is high, doing gradient descent
- Can do better than Levenberg by using Hessian information
- Move further in the direction where gradient is small
- Improves convergence in this process
- Combination known as **Levenberg-Marquardt algorithm**

Sparse bundle adjustment

Issues with Bundle Adjustment

- **Too many points**
  - Do not include all views or all points at the same time
- **Interleave Speed-up**
  - Alternately adjust structure and motion
  - Reduces size of Hessian to be inverted to max of 11×11
- **Sparse Methods**
  - Effectively exploit the sparsity of the Jacobian involved
  - Jacobian form is used to approximate the Hessian
  - Approximation of form $(\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T$

Sparse bundle adjustment



Exploiting Sparse Structure

- Jacobian has a sparse structure
- Separates out relationship between point structure and cameras
- Can exploit this in solving the normal equations

Iterative bundle adjustment

1. Initially choose two cameras having most inliers using fundamental matrix
2. Obtain structure and motion by minimization using *LM*.
3. Remove the points and correspondences with large reprojection errors
4. Remove points which violate the cheirality condition
5. Add images that are more consistent with the already recovered cameras.
6. Apply DLT to initialise the new camera internal and external calibration.
7. Go to 2

Sparse bundle adjustment

Global bundle adjustment

- Triangulate all points using camera parameters obtained using rotation and translation averaging.
- Discard 3D points with too low or too high apical angles.
- Remove cameras which have very few points visible. Remove those 3D points as well.
- Run Sparse bundle adjustment (PBA).
- For each camera check the re-projection errors. Remove a 3D point as outlier from the camera if the re-projection error is large.
- Check for cameras which have very few 3D points after removal. Remove those cameras.
- Repeat till "no more outliers" or max iteration reached.

What makes it possible?

- Linear perspective. Geometry.
- Rich feature extraction.
- Feature grouping, clustering and data structures.
- Graph algorithms for image clustering.
- Large scale non-linear optimization.
- **Global motion averaging**.
- Divide and conquer?

Global motion averaging: rotation

- Compute t_{ij} and R_{ij} from the pairwise epipolar geometry of edges of the matchgraph.
- Let $(R_i, \mathbf{C}_i)$ and $(R_j, \mathbf{C}_j)$ are absolute rotation and position of i and j in global frame of reference. Then the rotations are related as $R_j R_i^T = R_{ij}$.
- Let $R_{global} = \{R_1, ..., R_N\}$. Compute the rotations in the global frame as

$$\underset{R_{global}}{\operatorname{argmin}} \sum_{(i,j)\in E} d^2(R_j R_i^T - R_{ij})$$

where

$$d(R_1, R_2) = \frac{1}{\sqrt{2}} || \log(R_2 R_1^{-1}) ||_F$$

is the intrinsic bivariate distance on $SO(3)$.

Chatterjee and Govindu, 2013.

Global motion averaging: translation

- $$\mathbf{t}_{ij} \propto R_j(\mathbf{C}_i - \mathbf{C}_j) \Rightarrow R_j^T \mathbf{t}_{ij} \propto (\mathbf{C}_i - \mathbf{C}_j) \Rightarrow \mathbf{C}_{ij} \propto (\mathbf{C}_i - \mathbf{C}_j)$$
- Remove ourliers if
  1. The epipolar estimate of an edge in a match graph is not reliable.
  2. The errors in estimated pairwise rotations after averaging is greater than threshold.
- Minimize
$$\sum_{(i,j)\in E} dist(\mathbf{C}_{ij}, \frac{\mathbf{C}_j - \mathbf{C}_i}{||\mathbf{C}_j - \mathbf{C}_i||})$$

Wilson and Snavely, 2014.

What makes it possible?

- Linear perspective. Geometry.
- Rich feature extraction.
- Feature grouping, clustering and data structures.
- Graph algorithms for image clustering.
- Large scale non-linear optimization.
- Global motion averaging.
- **Divide and conquer?**

Divide and conquer: Efficient large scale SFM using graph paritioning (ACCV, 2014)

Use multiway **Normalised Cut** (Shi and Malik, 2000):

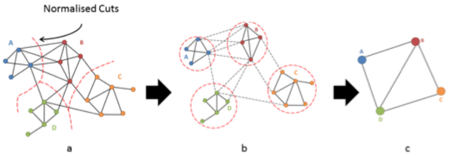
$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

where

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

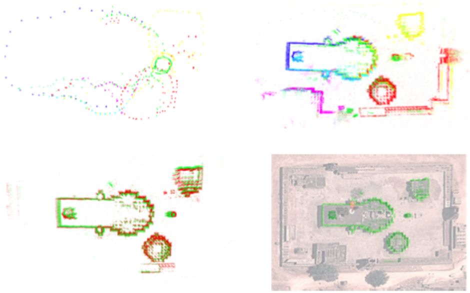
$$assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$$

Component-wise reconstruction and registration

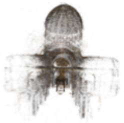


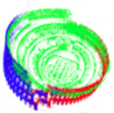
- Incremental bundle adjustment for reconstruction of each component.
- Estimation of epipolar geometry, rotation and translation of cut edges.
- Global motion averaging and registration.

Segmentation and reconstruction example



Reconstruction examples



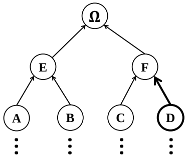


Alternative: agglomerative clustering

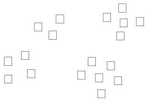
- Create a reduced match-graph from the vocabulary tree.
- Associate with each edge a score of robustness of epipolar computation.
- Grow a dendritic tree bottom up using

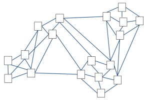
$$D_{AB} = \frac{1}{n_A n_B} \sum_{i=1}^{n_A} \sum_{j=1}^{n_B} w(x_{Ai}, x_{Bj})$$

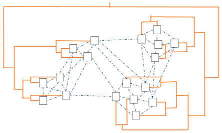
- Cluster by moving bottom up, stopping at internal nodes which satisfy a *size constraint*.

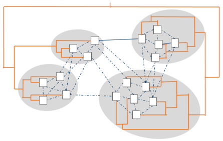


Hierarchical reconstruction pipeline

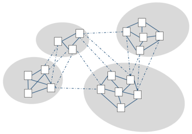

(a) Unorganised set of images

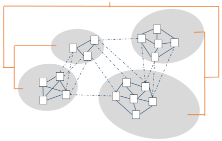

(b) Match graph created using vocabulary tree and epipolar geometry

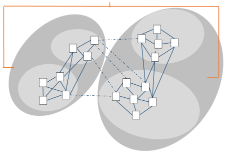

(c) A binary tree created using agglomerative clustering from the matching information

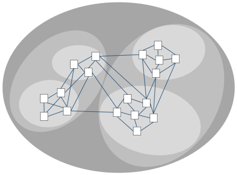

(d) Base clusters of 40 to 140 images identified from the agglomerative tree

Hierarchical reconstruction pipeline


(e) SfM is solved within each base cluster using motion averaging and triangulation


(f) Pairwise registration of reconstructed clusters using inter-cluster epipolar relationships


(g) Selective batch bundle adjustment of registered clusters


(h) Final solution

Some reconstructions

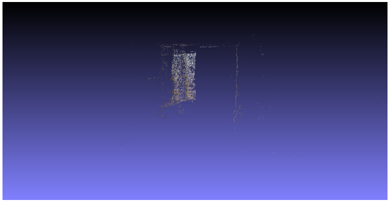
Vitthala Temple

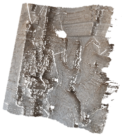
Piccadilly Circus

One order of magnitude speed-up

Dataset	Match graph creation using vocabulary tree, epipolar pruning and clustering (mins)	Total reconstruction and registration time for all clusters (mins)	BBA refinement using PBA and re-section (mins)	Total time by our method (mins)	Pairwise matching by by VSFM (mins)	Reconstruction by VSFM (mins)	Total time by VSFM (mins)
Central Rome	1358	312	35	**1705**	N/A	N/A	N/A
Vitthala Temple	592	80	5	**677**	9522	59	9581
St. Peter's Basilica	108	28	1	**137**	1385	10	1395
Art's Quad	874	156	11	**1041**	N/A	N/A	N/A
Colosseum	102	21	1	**124**	1231	29	1260
Piazza Bra [NEW]	30	2	1	**33**	110	5	115
San Giacomo	24	2	1	**27**	101	5	106

Dense reconstruction: Hampi pillar





Simultaneous Localisation And Mapping

LSD-SLAM: Large-Scale Direct Monocular SLAM
Jakob Engel, Thomas Schöps and Daniel Cremers
ECCV 2014

LSD-SLAM: Overview

- Monocular, video based.
- Works directly on intensities. Not feature based.
- Locally tracks the motion of the camera, but allows to build consistent, large-scale maps of the environments.
- Filtering-based estimation of semi-dense depth maps.
- The world is represented as a number of keyframes connected by pose-pose constraints.
- Pose graph optimization.
- Accounts for scale drifts.

LSD-SLAM: Pose representations

- 3D Rigid Body Transformations. $\mathbf{G} \in SE(3)$ is defined as

$$\mathbf{G} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \quad \text{with} \quad \mathbf{R} \in SO(3) \quad \text{and} \quad \mathbf{t} \in \mathbb{R}^3.$$

- $\xi \in \mathfrak{se}(3)$ is the corresponding element in the associated *Lie-algebra*. ($\xi \in \mathbb{R}^3$).
- Exponential map and inverse: $\mathbf{G} = exp_{\mathfrak{se}(3)}(\xi)$ and $\xi = log_{SE(3)}(\mathbf{G})$.
- Pose concatenation: $\circ : \mathfrak{se}(3) \times \mathfrak{se}(3) \rightarrow \mathfrak{se}(3)$ is given as:

$$\xi_{ki} := \xi_{kj} \circ \xi_{ji} := log_{SE(3)} \left(exp_{\mathfrak{se}(3)}(\xi_{kj}) \cdot exp_{\mathfrak{se}(3)}(\xi_{kj}) \right)$$

LSD-SLAM: Pose representations

- Define a 3D projective warp function ω which projects image point $\mathbf{p}$ with inverse depth d into a ξ transformed frame

$$\omega(\mathbf{p}, d, \xi) := \begin{pmatrix} x'/z' \\ y'/z' \\ 1 \end{pmatrix} \text{ with } \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} := exp_{\mathfrak{se}(3)}(\xi) \begin{pmatrix} p_x/d \\ p_y/d \\ 1/d \\ 1 \end{pmatrix}$$

- Finally, a 3D similarity transformation $\mathbf{S} \in Sim(3)$ is defined as

$$\mathbf{S} = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \quad \text{with} \quad \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \text{ and } \; s \in \mathbb{R}^+.$$

LSD-SLAM: Gauss-Newton Optimisation on Lie-Manifolds

- Two images are aligned by Gauss-Newton minimisation of the photometric error

$$E(\xi) = \sum_{\mathbf{i}} \underbrace{\left(I_{ref}(\mathbf{p_i}) - I(\omega(\mathbf{p_i}, D_{ref}(\mathbf{p_i}), \xi))\right)^2}_{=:\mathbf{r}_\mathbf{i}^2(\xi)}$$

- Starting with an initial estimate $\xi(0)$, in each iteration an increment $\delta\xi^{(n)}$ is computed by solving for the minimum of a Gauss-Newton second-order approximation of E

$$\delta\xi^{(n)} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}(\xi^{(n)}) \text{ with } \mathbf{J} = \left.\frac{\partial \mathbf{r}(\epsilon \circ \xi^{(n)})}{\partial \epsilon}\right|_{\epsilon=0}$$

- $\xi^{(n+1)} = \delta\xi^{(n)} \circ \xi^{(n)}$
- Iteratively re-weighted least-squares is possible and recommended for robustness.

LSD-SLAM: Overall scheme

- For every new image **the tracking component** estimates the new image pose $\xi \in \mathfrak{se}(3)$ with respect to the current key-frame, using the pose of the previous frame as initialization.
- The **depth map estimation component** estimates and refines the depth of the current key-frame over many per-pixel, small baseline stereo comparisons with interleaved spatial regularization.
- If the camera has moved too far, a new key-frame is initialized.
- Once a key-frame is replaced, it is incorporated into the global map by a **map optimization component**.
- To detect loop closures and scale-drift, a similarity transform $\xi \in \mathfrak{sim}(3)$ to close-by existing keyframes (including its direct predecessor) is estimated using scale-aware, direct $\mathfrak{sim}(3)$-image alignment.