Chemnitz University of Technology

Bachelor Thesis

Image-Based Photogrammetric Reconstruction of Geometrically Simple Objects for Mixed Reality Visualization

Author:
Xiangyu Tong

Supervisor:
Prof. Dr. Guido Brunnett
Second Supervisor:
Tom Uhlmann M.Sc.

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Computer Graphics and Visualization
September 22, 2021

Declaration of Authorship

I, Xiangyu Tong, declare that this thesis titled, "Image-Based Photogrammetric Reconstruction of Geometrically Simple Objects for Mixed Reality Visualization" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"*First of all, I want to thank Tom.Uhlman M.Sc., who gave me a lot of valuable opinions on my bachelor thesis, so that I have a goal and direction in developing and writing this thesis.* "

Xiangyu Tong

CHEMNITZ UNIVERSITY OF TECHNOLOGY

Abstract

Computer Graphics and Visualization
Computer Science

Bachelor of Science

Image-Based Photogrammetric Reconstruction of Geometrically Simple Objects for Mixed Reality Visualization

by Xiangyu Tong

By taking advantage of preliminary progress in photogrammetric reconstruction, a robust working pipeline of photogrammetric reconstruction for geometrically simple objects is proposed in this thesis. This thesis combines the strength of SfM (Structure of Motion) and MVS (Multi-View Stereo). By taking several disordered photos of a certain scene as input of the proposed working pipeline, it produces a high quality 3D model of the geometrically simply object from the scene. Further more, the output can be easily applied in mixed-reality scenes using capable device or it completes the whole process of reconstruction on a server and send the results to users.

Contents

List of Figures

Chapter 1

Motivation

For a long time, researchers have tried to give machines the same perception as human senses, which includes vision. Due to the existence of binocular vision, it is relatively easy to obtain depth information. As one key technology of environment perception, 3D reconstruction can be applied in auto pilot, mixed reality, motion detection, security monitoring, etc. Despite the blooming researches on image understanding, it is only the part of computer vision. Beyond the part of image understanding, the next step is to perceive the 3D environment. Hence, the 3D reconstruction is one of the most important areas in computer vision and computer graphics. As we live in the 3 dimensional space, for the interaction and perception, it is meaningful to reconstruct the world and then create the immersive experience. However, due to the loss of one dimension in the projection process, the estimation of the true 3D geometry is difficult and a so called ill-posed problem, because usually infinitely many different 3D surfaces may produce the same set of images. The related traditional works are mainly based on photogrammetric reconstruction, i.e. recovering the 3D information using the various geometric relations. One of the greatest challenges is to estimate structure and motion of camera (camera geometry), since the 2D images taken from a 3D scene lack the information of depth and camera parameters. Table 1.1 shows the certainty of pose of camera, triangulation and reconstruction from scene geometry or motion. Usually the camera position, internal parameters and the motion of triangulation are assumed to be known or can be estimated from the set of images. The 3D reconstruction is based on estimation, either from scene geometry or motion, with the help of image correspondences. Accordingly, the goal of multiview 3D reconstruction is to infer geometrical structure of a scene captured by a collection of images using various mathematical and geometrical techniques.

Table 1.1: 3D information estimation

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

Besides, the performance of photogrammetric reconstruction is limited by the quality of the images and the photography-conditions. The proposed method by the author is a refinement of the preliminary work of Schönberger [19] and Hernández [7] in photogrammetric reconstruction. In recent years, with the popularity of deep learning, thousands of papers related to generate 3D models using single image were

published. It seems that the strength and magic of deep learning has been proved again in a way, that is, doing almost the impossible. At the same time, the traditional geometric methods seem to be ignored to the public. However, after some strict tests of experiment [29], the quality of almost all those works based on deep learning or neural network is actually not so good comparing with the traditional methods such as nearest neighbor baselines. As Porf. Yi Ma said in his speech, *there are no such universal methods or algorithms, at least in the area of 3D reconstruction. It is not a scientific way, if one algorithms does not take advantage of the geometric relationships sufficiently. Theoretically, if 3D models can be recovered through learning from a single image, then most of the animals would have only one eye.*

Therefore, the study of photogrammetric reconstruction still make one of the major contribution to the research on 3D reconstruction. Moreover, from the author's aspect, it is of interest to know whether the general method of SfM (Structure from Motion) + MVS (Multi-View Stereo) still works well with geometrically simple object. Hence, the aim of the author here is to propose a strong version of photogrammetric reconstruction working-pipeline and package into an integrated project, which can be easily implemented and used by readers. By taking several disordered photos of a certain scene as input of the proposed working pipeline, it produces a high quality 3D model of the geometrically simple object from the scene with less ambiguity. The proposed method by the author should be robust to the quality, number of images and the pose of camera, which means that the result throughout the working pipeline is typically good, regardless of the way of photography. Specifically, for a box shaped object, the surface of the reconstructed 3d model is expected as flat as the real object and the geometric relations between edges is similar to the real geometric relation. Likewise, for a sphere shaped object, the surface is expected to be as smooth as possible. Furthermore, the reconstructed 3D models can be imported into mixed reality scene and the output of this pipeline is an App executed in portable devices such as iOS, Android, etc.
In addition, the author has implemented the whole working pipeline of reconstruction on a server so that it is convenient to the people outside to complete the reconstruction using only smartphone after photography. Given the above words and to summarize, what the author want to achieve in this proposed pipeline is the "3A", that is, reconstruction anywhere (no critical constraints on photography), in any develop environment (user friendly), and with any common mobile devices (easily applied).

Chapter 2

Preliminary Work

In order to facilitate readers understanding of the whole process of my working-pipeline and use the self-collected images to run the process, several classic 3D reconstruction technologies based on Structure from Motion (SfM) and Multi-View Stereo(MVS), that have been proposed in recent years, will be described in the next sections.

2.1 Photogrammetric Reconstruction

Structure from motion(SfM) is a photogrammetric range imaging technique for estimating three-dimensional structures from its projections into image sequences taken from different viewpoints. SfM strategy can be divided into incremental[32], hierarchical[39], and global approaches[10]. The incremental SfM is mostly applied in this working-pipeline (denoted as SfM in this thesis), which is a sequential process with an iterative reconstruction component (see Figure 2.1).

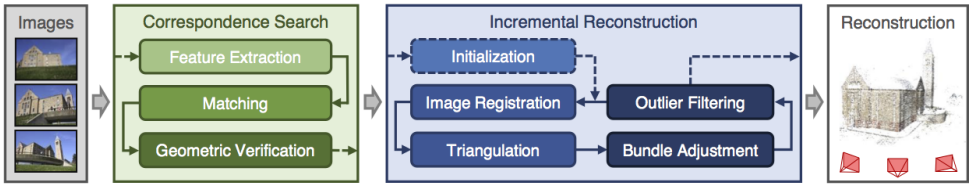


Fig. 2.1. the process of Incremental SfM consisting of correpondence search and incremental reconstruction, source: Colmap-SfM

Incremental-SfM begins generally with feature extraction and feature matching. The output is a set of image-pair-candidates $\mathcal{C} = \{\{I_a, I_b\} \mid I_a, I_b \in \mathcal{I}, \quad a < b\}$ and their associated feature correspondences $\mathcal{M}_{ab} \in \mathcal{F}_a \times \mathcal{F}_b$. Then comes the geometric verification, which estimates a homograhpy transformation $\mathbf{H}$ that maps feature points between images. The resulting scene graph as the input of incremental process supports the model to start the initial two-view reconstruction, before sequentially registering new images, triangulation, and refinement using bundle adjustment (BA).
Correspondence Search The goal of this stage is to find scene overlap from input images $\mathcal{I} = \{I_i \mid i = 1...N_I\}$ and identify projections of the same points in overlapping images. As shown in Figure 2.2 the output is Scene Graph and a set of geometrically verified image pairs.

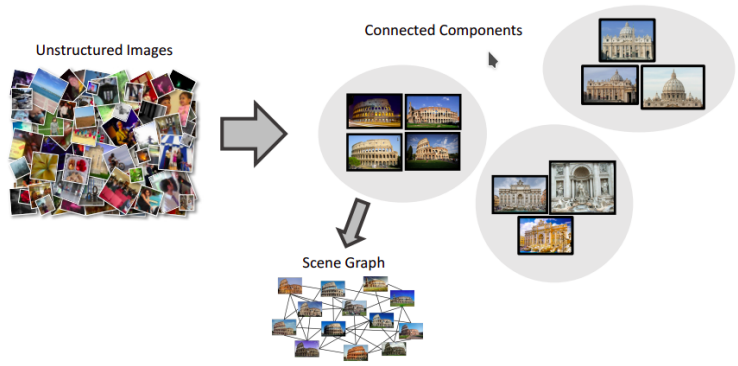


Fig. 2.2. after SfM, the relationships between image pair and the sharing content are founded. source: CVPR2017, turorial

Feature Extraction. The most well-know feature detection method is the SIFT (scale-invariant feature) algorithm.[25]. By using a suboctave DoG pyramid, the interesting patches at different scales is extracted (Figure 2.3).

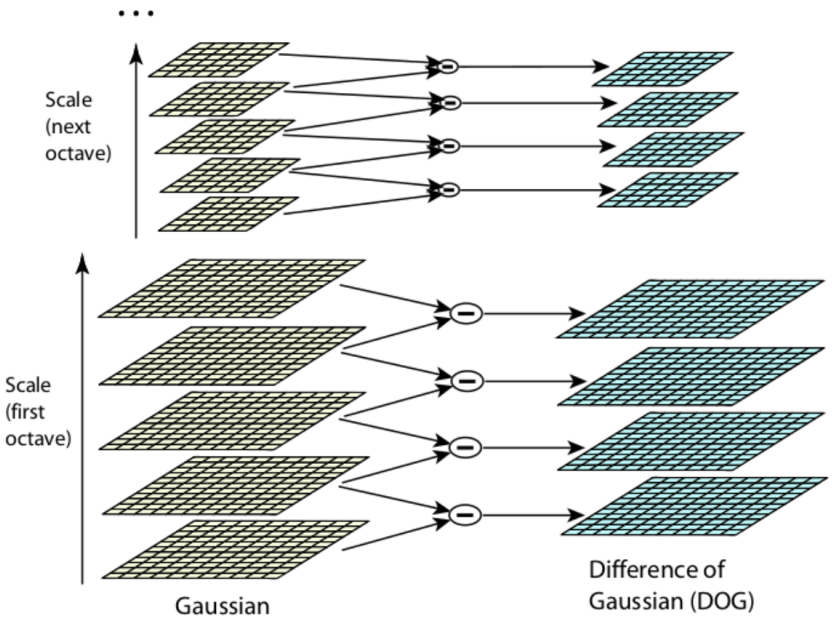


Fig. 2.3. proceed DoG of an image in different layers. source : David G. Lowe (2004). Distinctive image features from scale-invariant keypoints. IJCV

SIFT computes the gradient orientation for each pixel in the patch:

$$\theta = \arctan \frac{I_y}{I_x} \tag{2.1}$$

Regardless of its scale, each patch is split into NxN spatial bins (usually 4x4).
then, as shown in Figure 2.4, a histogram of M orientations for each bin is computed (M=8).

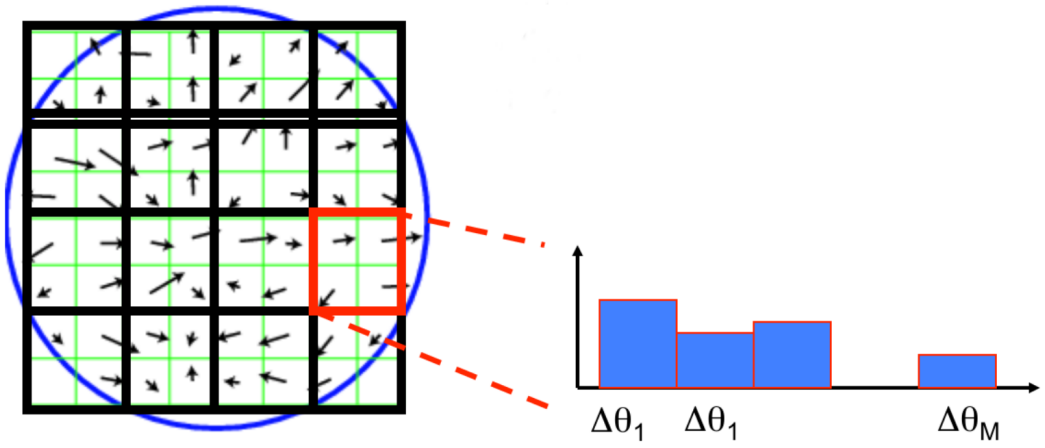
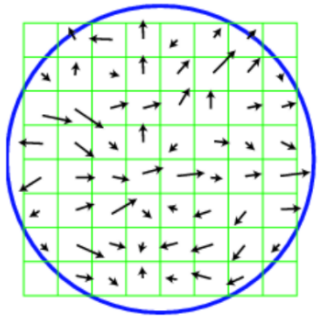


Fig. 2.4. Each patch is divided into N x N spatial bins and a histogram of M orientations for one of those bins. source: R. Szeliski

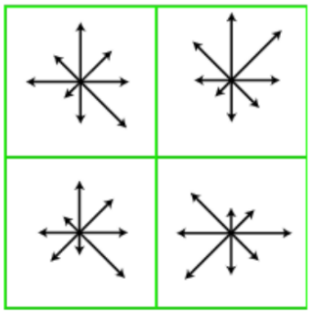
The orientation applied in the histograms are weighted by a Gaussian fall-off function, which gives more importance to orientations in the middle of the patch than on the edges. Since there are 4×4 histogram each with 8 orientation bins, this gives a vector W of 128 elements. By removing the mean and normalizing the variance, the vector is finally normalized:

$$W_n = \frac{W - \bar{W}}{\|W - \bar{W}\|} \tag{2.2}$$

This vector is a descriptor vector for each key-point such that the descriptor is highly distinctive and partially invariant to the remaining variations such as illumination, 3D viewpoint, etc.
SIFT algorithm uses gradients to remove the mean pixel intensity and the normalization brings invariance to the contrast. Thus, it is robust to illumination changes. Besides, it is also robust to small translation and scale, since histogram loses the spatial information and the DoG pyramid extracts features at their distinctive scale. Likewise, the SIFT features can be used for 3D object recognition and 3D modeling in augmented reality, where synthetic objects with accurate pose are superimposed on real images.
A general steps of SIFT is shown as below.



(a) image gradients



(b) keypoint descriptor

Fig. 2.5. (a) For each spatial bin compute the strength and orientation of the gradient at each pixel. (b) Compute a histogram with 8 orientation bins. Source: R. Szeliski

1. Extract interesting patches at different scales using a suboctave DoG pyramid.
2. Split each patch into a 4x4 grid of spatial bins.
3. For each spatial bin:
   - Compute the strength and orientation of the gradient at each pixel.
   - Weight the orientation by a Gaussian fall-off function.
   - Compute a histogram with 8 orientation bins.
4. Put all histograms in a vector with 128 elements and normalize it.

After detecting all features in images, the next step is to select image pairs for feature matching. **Image Matching** aims to find images that are looking to same areas of the scene. For that, we use the most common method to generate this image descriptor with vocabulary tree approach[36]. The main idea is image retrieval techniques, which to find images that share some of the contents without the cost of resolving all feature matches in details.

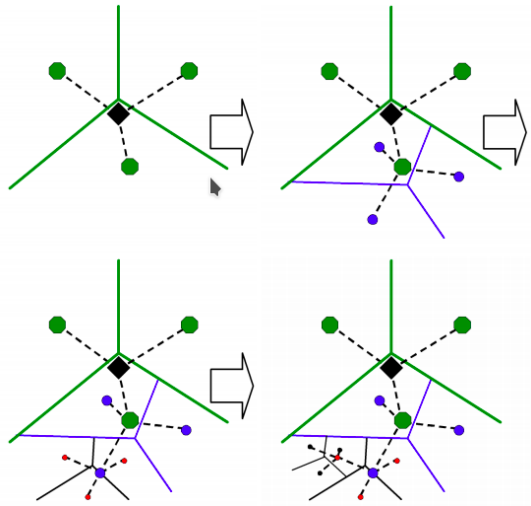


Fig. 2.6. The Voronoi regions are defined recursively by splitting its father Voronoi region. Source: Scalable Recognition with a Vocabulary Tree, David Nister and Henrik Stewenius, CVPR 2006

First of all, by defining k cluster centers the training data is then divided into k groups, where each group consists of the descriptor closest to its cluster center. Mathematically, this means that the above descriptors are divided according to the Voronoi diagram generated by theses mean points[27]. Recursively, the same process is proceeded to each group of descriptor vectors, defining Voronoi regions by splitting its father Voronoi region. The tree is determined level by level up to a particular number of levels seen in Figure 2.6. By passing all extracted features descriptors into it, it makes a classification by comparing their descriptors to the ones on each node of this tree. Each feature descriptor ends up in one leaf, which can be stored by a simple index: the index of this leaf in the tree. The image descriptor is then represented by this collection of used leaves indices.

The computational cost in the hierarchical approach is logarithmic in the number of leaf nodes. The memory usage is linear in the number of leaf nodes k^L [36] where L denotes a maximum number of levels. The total number of descriptor vectors that must be represented is $\sum_{i=1}^{L} k^i \approx k^L$. Given the D-dimensional descriptors, which can be represented as **char**, the size of the tree is approximately Dk^L bytes.
let:

- $X = \{x_i\}_{i=1}^{N}$ be the descriptor vectors
- $\mu = \{\mu_j\}_{j=1}^{K}$ be the cluster centers
- $c = \{c_i\}_{i=1}^{N}$ be the encoder

In the running phase, each descriptor vector is propagated down the tree at each level by comparing the descriptor vector to the k candidate cluster centers and choosing the closest one denoted as c_i in level L_j. Trivially, the path down the tree can be encoded by a single integer C concatenating from c_i in each level L_j. Encoder c can be obtained by C *mod* k in each level.
Back to the K-means algorithm, the objective is to minimize the distortion function (or total cluster variance):

$$\tau = \arg\min_{\mu,c} \sum_{i=1}^{N} \|x_i - \mu_{c_i}\|^2 \tag{2.3}$$

This class of algorithms is called an expectation-maximization (EM) algorithm. If k and d(the dimension) are fixed, the problem can be exactly solved in time $O(N^{d \cdot K+1} \cdot \log N)$, where N is the number of entities to be clustered. The problem is computationally difficult(NP-hard). However, a variety of heuristic algorithms such as Lloyd's algorithm, that can always converge quickly to be a local minimum [30], are generally used. The running time of Lloyd's algorithm is $O(NKdi)$ [17], where i is the i-th number of iterations needed until convergence.

Since K-means algorithm is applied in this section, it is possible to check if different images share the same content by comparing these image descriptors. Following is the implementation of K-means algorithm.

Algorithm 1 Building the hierarchical k-means tree

Input: feature dataset D, branch factor K, center selection algorithm to use C_{alg}
Output: k-means tree

```
if | D |< K then
  create leaf node with the points in D
else
  P ← select K points from D using the C_alg
  converged ← false
  iterations ← 0
  while not converged and iterations < I_max do
    C ← clusterthepointsinDaroundnearestcentersP
    P_new ← meansofclustersinC
    if P = P_new then
      converged ← true
    end if
    P ← P_new
    iterations ← iterations + 1
  end while
  for each cluster C_i ∈ C do
    create non − leaf node with center P_i
    recursively apply the algorithm to the points in C_i
  end for
end if
```

Feature Matching. The objective of this step is to match all features between image pair candidates. First, do photogrammetric matching between the set of descriptors from the 2 input images to find correspondences between them. The transformation (affine or projective) between the two sets of features must then be estimated. For each feature in image A, a list of candidate features in image B is obtained. The naive approach(brute force) checks every image pair for scene overlap by searching correspondences for all features of image A, which feature of image B is the closest in the feature space:

$$d(patch_1, patch_2) = \|SIFT(patch_1) - SIFT(patch_2)\|_2 \tag{2.4}$$

Find the 2 closest descriptors in the second image for each feature is computationally intensive with a brute force approach $O(N_I^2 N_{Fi}^2)$ and is prohibitive for large image collections. As the descriptor space is not a linear and well defined space, we cannot rely on absolute distance values to know if the match is valid or not (we can only have an absolute higher bound distance). To remove bad candidates, we assume that there is only one valid match in the other image. So for each feature descriptor on the first image, we look for the 2 closest descriptors and we use a relative threshold between them. This assumption will kill features on repetitive structure but has proved to be a robust criterion.[25]
There exists many optimized algorithms[47][21][50][49][18][20][19]. The most common

one is Approximate Nearest Neighbor (ANN)[31], which is suggested by the author and applied in the proposed working-pipeline.

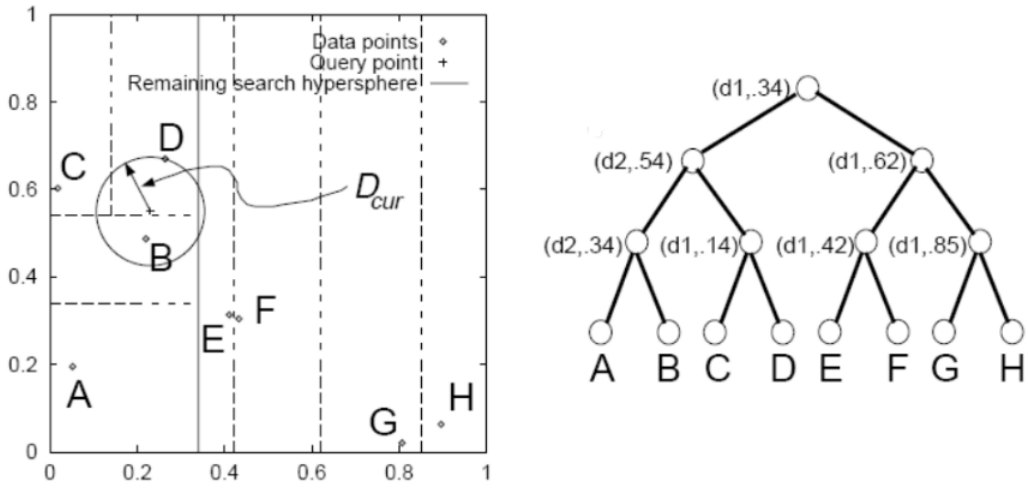


Fig. 2.7. Kd-Tree in 2 different representations, source: Beis and Lowe 1999

The ANN algorithm involves a space-partitioning data structure called Kd-Tree [5]. The Kd-tree is a binary tree where each leaf node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that cuts the space into two parts, known also as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points from the right of the hyperplane are represented by the right subtree. A particular hyperplane direction can be chosen as follow: every node in the tree is associated with one of the k dimensions, with the hyperplane vertical to that dimension's axis. In Figure 2.7(left), the spatial classification of the axis-aligned cutting plane is considered as dashed lines. Similarly , from Figure 2.7(right), the subdivision can be defined as a binary tree.

Feature alignment using least squares. Let' s suppose N matches between the feature points $\{x_i\}_{i\in[1,N]}$ and $\{x'_i\}_{i\in[1,N]}$ have been found in image pairs. Then perform searching for a planar transformation **p** which minimizes the **least squares error** between the two sets of points[38] :

$$\begin{aligned} E_{LS}(p) &= \sum_{i=1}^{N} \|r_i\|^2 = \sum_{i=1}^{N} \|W(x_i,p) - x'_i\|^2 \\ &= \sum_{i=1}^{N} \|J_p(x_i) \times p - \Delta x_i\|^2 \\ &= p^T \times \mathcal{H}_p \times p - 2p^T \times b_p + c \end{aligned} \tag{2.5}$$

where $W(x_i,p)$ is a warp function transforming the point x_i with the parameterized planar transformation p and r_i is the residual between the actual position of the feature in the second image x'_i and its predicted position by $W(x_i,p)$.
$J_p(x)$ is the Jacobian of a transformation between two vectors and is defined by:

$$J_p(x) = \left[\frac{\partial W(x,p)}{\partial p_1}, \frac{\partial W(x,p)}{\partial p_2}, ..., \frac{\partial W(x,p)}{\partial p_l}\right] \tag{2.6}$$

and $\mathcal{H}_p = \sum_{i=1}^{N} J_p(x_i)^T \times J_p(x_i)$ is the **Hessian matrix** of the transformation and

$b_p = \sum_{i=1}^{N} J_p(x_i) \times \Delta x_i$.The minimum p^* of this function is achieved when its derivative w.r.t p is zero. trivially, we get:

$$p^* = \mathcal{H}_p^{-1} \times b_p \tag{2.7}$$

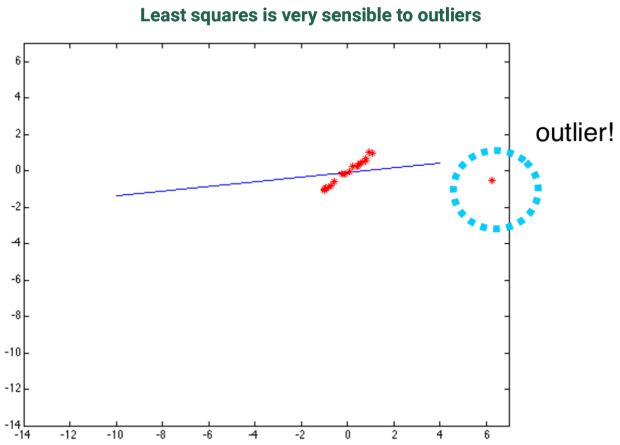


Fig. 2.8. One dimension least square is a kind of linear regression, which is very sensible to outliers. Source: S. Savarese

Figure 2.8 shows that the least square is sensible to outliers. Considering this, the feature correspondences are used in the images to make a geometric filtering by using epipolar geometry in an outlier detection framework called RANSAC (RANdom SAmple Consensus)[13]. RANSAC is an iterative method that decides to leave out the outliers from the calculation of the transformation. At each iteration, RANSAC randomly select a small subset of the feature correspondences and estimate the transformation using least squares and check the number of features that validates this model.The winning transformation is the one with the most inliers (or least outliers). RANSAC doesn't deal well with conditions that:

- The number of true outliers is small compared to the number of inliers.
- There are enough inliers to estimate the parameters.

Many variants of RANSAC exist:

- **Preemptive-RANSAC**[35] adds constraints on the transformation to rule out early most outliers
- PROSAC[8] selects the next k pairs from the previous set of inliers.

Then comes the **Incremental Reconstruction**.
The goal of this step is to understand the geometric relationship between each 2 of the observations provided by the input images, and infer the rigid scene structure (3D points) with the pose (position and orientation) and internal calibration of cameras Figure 2.9. The Incremental pipeline is a growing reconstruction process. It first computes an initial two-view reconstruction that is iteratively extended by adding new views.

Two-View Geometry.

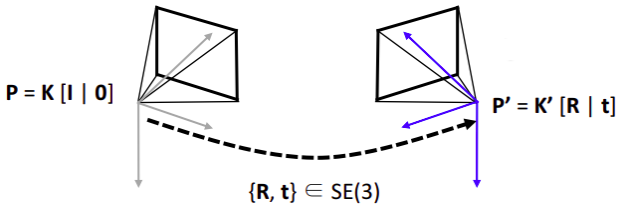


Fig. 2.9. Two view geometry is the relative geometry of two different perspective views of the same 3D scene. It is usually referred to as epipolar geometry. In epipolar geometry, a point x in the first view constrains the position of the corresponding point x' in the second view, which must lie on the corresponding epipolar line. Sourc: CVPR2017, Tutorial

For the two-view geometry, we have the fundamental matrix $F = [e']_\times H_\pi$, where H_π is a 2D homography mapping every x_i to x'_i.

Lemma 2.1.1. *The fundamental matrix satisfies the condition that for any pair of corresponding points $x \leftrightarrow x'$ in the two images [40]*

$$x'Fx = 0 \tag{2.8}$$

This lemma shows a way characterizing the fundamental matrix without reference to the camera matrices, i.e. only in terms of corresponding pair. Besides, according to the works from Hartley and Zisserman[41], we have a general form for computing F

$$F = K'^{-T}[t]_\times RK^{-1} \quad (Longuet - Higgins\ equation) \tag{2.9}$$

The 8-point algorithm is the simplest method of computing the fundamental matrix F, involving no more than the construction and (least-squares) solution of a set of linear equations[LonguetHiggins-81].
From the Proof[C.9] of Hartley and Zisserman, the camera matrices corresponding to a fundamental matrix F may be chosen as $P = [I \mid 0]$ and $P' = [[e']_\times F \mid e']$
Considering normalized coordinates, we have the essential matrix, which is the specialization of the fundamental matrix[42].

$$\hat{x}'^T E\hat{x} = 0 \tag{2.10}$$

in terms of the normalized image coordinates for corresponding points $x \leftrightarrow \hat{x}$.

The essential matrix $E = k'^T FK = [t]_x R$ has only five degrees of freedom. In principal, 5 pair of corresponding points are sufficient to decide E [34].
Decompose E **to** R, t. For a given $E = Udiag(1,1,0)V^T$ and the first camera matrix $P = [I \mid 0]$, there are four possible choices for the second camera P', namely

$$\begin{aligned} P' = [UWV^T \mid +u_3] \quad or \quad [UWV^T \mid -u_3] \quad or \\ [UW^TV^T \mid +u_3] \quad or \quad [UW^TV^T \mid -u_3] \end{aligned} \tag{2.11}$$

based on the two possible choices of R and two possible signs of t.

$$\begin{aligned} t &= \pm U(0,0,1)^T = u_3 \\ R &= UR_z(\frac{\pi}{2})V^T, UR_z^T(\frac{\pi}{2})V^T \end{aligned} \tag{2.12}$$

where $R_z(\frac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(2.12) shows 4 possible solutions shown in Figure 2.10.

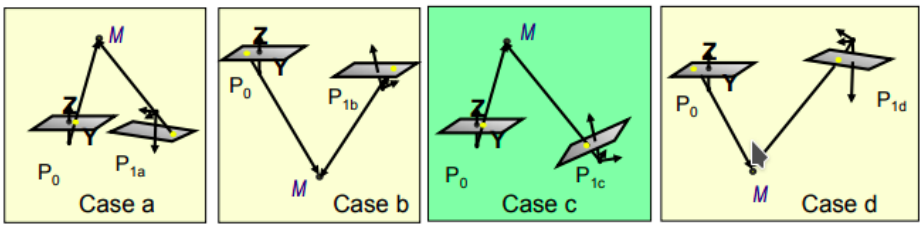


Fig. 2.10. The four solutions show that a reconstructed point X will be in front of both cameras in one of these four solutions only, Thus, testing with a single point to determine if it is in front of both cameras is sufficient to decide between the four different solutions for the camera matrix P'. surce: CVPR2017, Tutorial

Between the left and right sides there is a baseline reversal. Between the top and bottom rows camera B rotates 180° about the baseline. Therefore, only in case C is the reconstructed point in front of both cameras.

Triangulation.

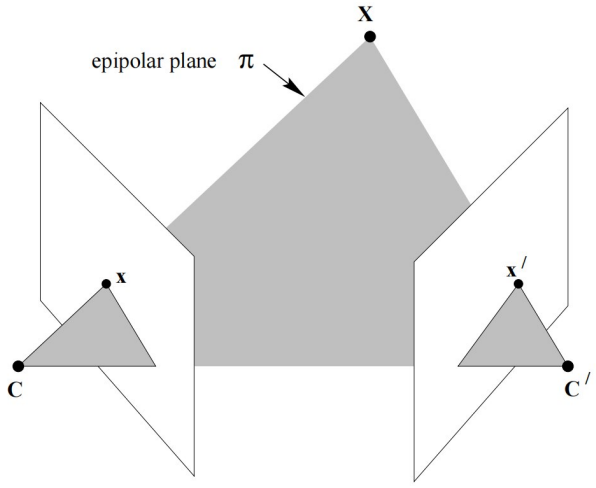


Fig. 2.11. Given the two cameras (P, P') and the corresponding image point pair $x_i \leftrightarrow x_i'$, find the 3D point X_i that projects to the given image points. Solving for X in this way is known as triangulation. Source: Multiple View Geometry in Computer Vision, page 240

Given the 2 cameras indicated by their centers C and C', the corresponding matrices P and P' and one image points correspondence $\{x_i, x_i'\}$, estimate the 3D point X. According to the epipolar geometry, we have $x \times (PX) = 0$ and writing this out gives[43]

$$\begin{aligned} x(p_{3T}X) - (p^{1T}X) &= 0 \\ y(p_{3T}X) - (p^{2T}X) &= 0 \\ x(p_{2T}X) - y(p^{1T}X) &= 0 \end{aligned} \tag{2.13}$$

Then an equation of the form $AX = 0$ can be derived, with

$$A = \begin{bmatrix} xp_{3T} - p^{1T} \\ yp_{3T} - p^{2T} \\ x'p'_{3T} - p'^{1T} \\ y'p'_{3T} - p'^{2T} \end{bmatrix} \tag{2.14}$$

This concatenates the 2D points from both images. This homogeneous linear system can be solved using SVD (singular value decomposition). However, due to the geometric error the rays might be skewed, which means they don't intersect. Figure 2.12 shows this situation

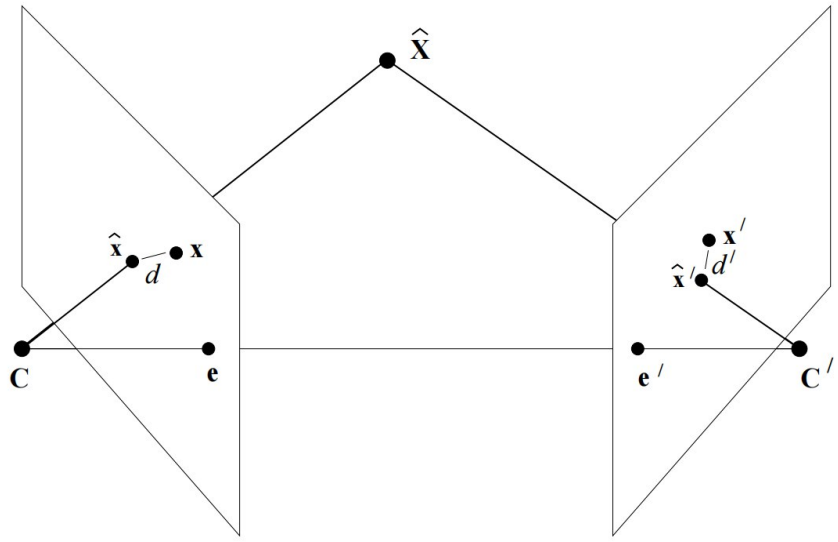


Fig. 2.12. Triangulation: Issues. The estimated 3-space point $\hat{X}$ projects to the two images at $\hat{x}, \hat{x}'$. The corresponding image points $\hat{x}, \hat{x}'$ satisfy the epipolar constraint, unlike the mesured points x and x'. The point $\hat{X}$ is chosen so that the reprojection error $d^2 + d'^2$ is minimized. Source: Multiple View Geometry, page: 314

Since the point correspondences $x \leftrightarrow x'$ do not in general satisfy the epipolar geometry constraint, let $\hat{X}$ satisfy the camera geometry so that $\hat{x} = P\hat{X}$ and $\hat{x}' = P\hat{X}'$ and minimize the following function:

$$C(x, x') = d(x, \hat{x})^2 + d(x', \hat{x}')^2 \quad subject\ to\ \hat{x}'F\hat{x} = 0 \tag{2.15}$$

where $d(*, *)$ is the Euclidean distance between the points.
Reformulation of the minimization problem The above minimization problem can be formulated differently as follows, that is, to minimize

$$d(x, l)^2 + d(x', l')^2 \tag{2.16}$$

where l and l' range over all choices of corresponding epipolar lines. Hence the point $\hat{x}$ is the closest point on the line l to the point x and so is $\hat{x}'$. The main idea is shown bellow[44]

- Parametrize the pencil of epipolar lines in the first image by a parameter t. Thus an epipolar line in the first image may be written as $l(t)$, which shown in Figure 2.13 .

- Using the fundamental matrix F, compute the corresponding epipolar line $l(t)$ in the second image.
- Express the distance function $d(x, l(t))^2 + d(x', l'(t))^2$ explicitly as a function of t.
- Find the value of t that minimizes this function.

Finally the problem is reduced to solving the minimum of a function in one variable.

$$\mathcal{C} = \arg\min_{\hat{X}} d(x, \hat{x})^2 + d(x', \hat{x}')^2 \rightarrow \mathcal{C} = \arg\min_{t} d(x, l(t))^2 + d(x', l'(t))^2 \tag{2.17}$$

Fig. 2.13. the projections $\hat{x}$ and $\hat{x}'$ of an estimated 3D point $\hat{X}$ lie on a pair of corresponding lines. The optimal $\hat{x}$ and $\hat{x}'$ will lie at the foot of the perpendiculars from the measured points x and x'. Parametrizing the corresponding epipolar lines as a one parameter family, the optimal estimation of $\hat{X}$ is reduced to a one-parameter search for corresponding epipolar lines so as to minimize the squared sum of perpendicular distances $d^2 + d'^2$. source: Multiple View Geometry in Computer Vision, page: 315

Figure 2.13 shows that the projection $\hat{x}$ and $\hat{x}'$ of an estimated 3D point $\hat{X}$ lie on a pair of corresponding epipolar lines. Then by parametrizing the corresponding epipolar lines as a one-parameter family, the optimal estimation of $\hat{X}$ is then reduced to a one-parameter search procedure.

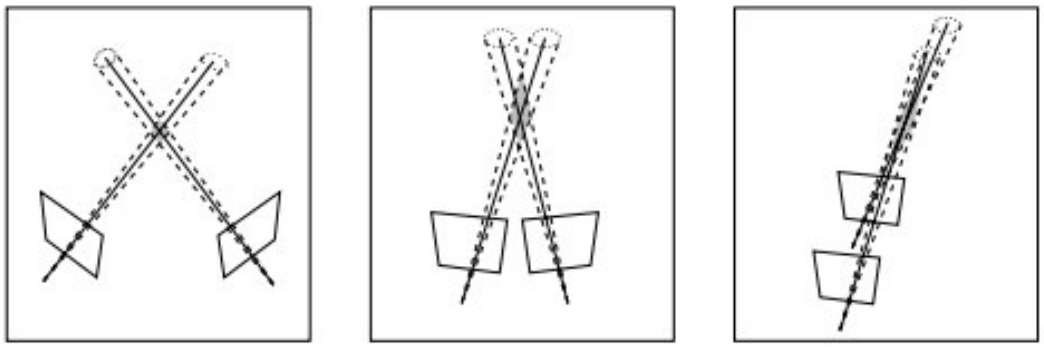


Fig. 2.14. The shaded area in each case shows the shape of the uncertainty region, which depends on the angle between rays. Points are less precisely localized as the rays become more parallel. Source: Multiple View Geometry, page: 321

The selection of angle between two view points is critical. When the angle became bigger, the fewer feature correspondences matched but less uncertainty of triangulation exist. The detail about how to set the angle between two view rays will be discussed in Chapter 4

Process of Incremental Reconstruction. A scene graph is produced from the stages above and as the input of incremental- reconstruction model. The output of this stage is the reconstructed scene structure as a set of points $\mathcal{X} = \{X_k \in \mathbb{R}^3 \mid k = 1...N_x\}$. First, we put all feature matches between image pairs into tracks. Each track is considered as the representation of a point in place, which is visible from multiple views(camera) (see Figure 2.15).

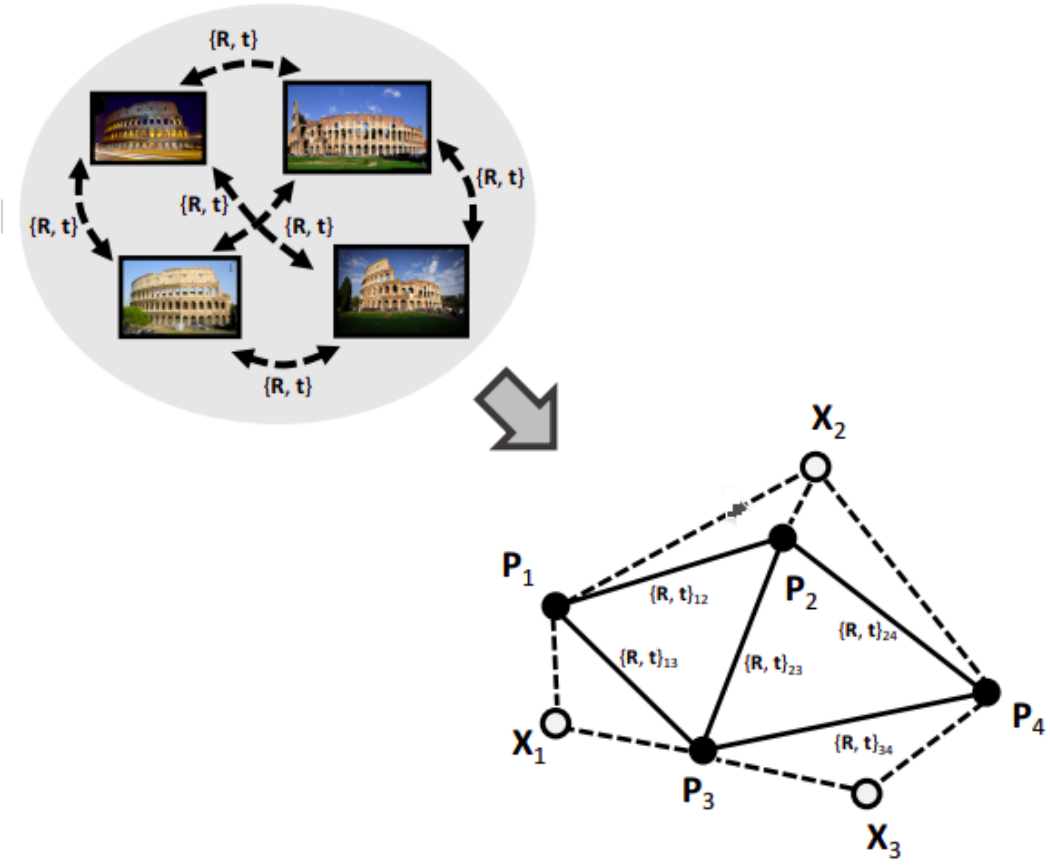


Fig. 2.15. Scene graph consists of views and the sharing content visible by different views. source: CVPR2017, Tutorial-Sparse Modeling

Initialization. SfM has to choose the two-view reconstruction as the best initial image pair[4]. Choosing a good initial image pair is critical for the quality of the final reconstruction, since the reconstruction might bot be recovered from an unsuitable initialization. Furthermore, it also requires robust matches and contains more true geometric information. Therefore, the initial image pair should maximize the number of matches and the repartition of the corresponding features in images. However, the angle between two view points has to be large enough to get the reliable geometric information. For that, the choice of the seed location has a positive influence on robustness, correctness, and performance of the incremental process.

Considering that the first image is the origin of the coordinate system, SfM computes the fundamental matrix between these two images. Since the pose of two cameras is known already, we can triangulate the image correspondences and reproject it to 3D space.

Image Registration. All the images from the rest, that have enough associations with the features that are already reconstructed in 3D space, are selected (shown in Figure 2.16 (a)). This algorithm is called next best views selection[24]. In metric reconstruction, the new selected image can be registered to the current model by solving the Perspective-n-Point(PnP) problem[14] with the help of feature correspondences to reconstructed points in already registered images(2D-3D correspondences). An example is shown in Figure 2.16.

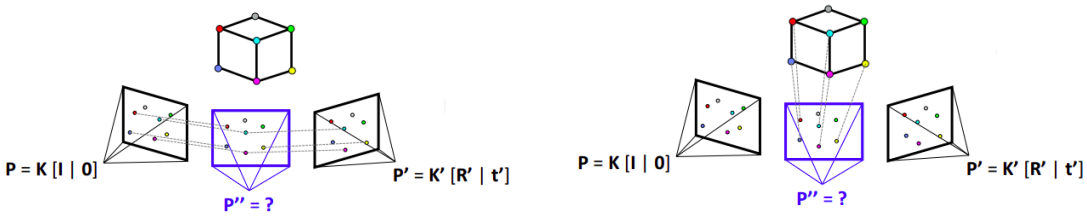


Fig. 2.16. (a) Find the 2D-2D correspondences between the new added image and the already rigistered image. (b) Find 2D-3D correspondences in the new added view and then compute the camera matrix p''. source: CVPR2017-Tutorial-Sparse-Modeling

Solving the PnP problem is to estimate the pose P_c and, if it was uncalibrated camera, then its intrinsic parameters. The set $\mathcal{P}$ is consequently expanded by P_c of the newly selected image. Due to the outlier in 2D-3D correspondences, the pose for calibrated camera is estimated in a RANSAC framework and a non-linear minimization is performed to refine the pose[14] [48]. From the work of Bujnak, Kukelova[26], the problem for the condition of uncalibrated camera can be solved with various minimal solvers.

After that, as shown in Figure 2.17, some tracks become visible by 2 or more registered cameras from these new camera's poses. Then it triangulates them and new points X_k can be added into $\mathcal{X}$. There exist a large number of methods for multi-view triangulation. In the author's opinion, a robust and efficient triangulation is applied[22] in the proposed working-pipeline, which is a kind of sampling-based triangulation method that can robustly estimate all points within an outlier-contaminated feature track by concatenating two-view correspondences.

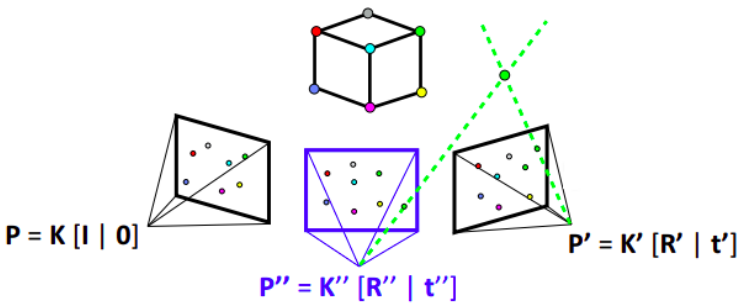


Fig. 2.17. From the 2D-2D correspondences and the estimated camera matrix P'', then triangulate them to obtain new points and add them into $\mathcal{X}$. source: CVPR2017, Tutorial-Sparse Modeling

Bundle Adjustment. If the image measurements are noisy then the equation $x_i^j = P^i X_j$ will not be satisfied exactly. In SfM pipeline, despite that image registration and triangulation are separate procedure, their results are highly simultaneously correlated. Without further refinement, SfM may easily shift to a non-recoverable state after a certain number of iterations. For this case, the author suggests to use the Maximum Likelihood(ML) solution[46] assuming that the measurement noise is Gaussian: estimate projection matrices $\hat{P}_c$ and 3D point $\hat{X}_k$ which project exactly to image point $\hat{x}_j$ as $\hat{x}_j = \hat{P}_c \hat{X}_k$ and also minimizes the reprojection error

$$E = \sum_j \rho(\|\pi(P_c, X_k) - x_j\|) \tag{2.18}$$

This estimation is known as BA. It involves adjusting the bundle of rays between each camera's center and the set of 3D points. Since the MVS in the next step is very sensitive to reprojection errors, BA is necessary for MVS to sub-pixel reprojection errors. A reconstruction with n points over m views requires minimization over $3n + 11m$ parameters.The Levenberg-Marquardt algorithm is one of the method solving BA problems[3].

Filter. As new points are estimated by triangulation, more image candidates are chosen for next best views selection. The process iterates like that, adding cameras and triangulating new 2D features into 3D points and removing invalid 3D points that with large reprojection errors[33] until new views can't be localized.

Re-Triangulation. The purpose of this stage is to improve the completeness of the reconstruction by keeping the tracks of points that previously failed to triangulate. However we don't increase the triangulation thresholds but only continue tracks with observations whose errors are below the filtering thresholds.

Multi View Stereo: Dense Reconstruction
This part will focus on reconstruction of the dense point cloud and remeshing the object. For that, the first step is to develop the concept of multi-view photometric consistency then generate depth map with associate view. From depth maps we can obtain the 3D information of each scene. **photo consistency measures**
Given a set $\mathcal{I}$ of N input images and a 3D point X viewed in all the images. Then we can define the photo-consistency of X:

$$\mathcal{C}_{ij} = \rho(I_i(\Omega(\pi_i(X))), I_j(\Omega(\pi_j(X)))), \tag{2.19}$$

where $\rho(a,b)$ is a similarity measure that compares two vectors, $\pi_i(X)$ projects point X into the i-th image, $\Omega(c)$ denotes a support domain around point X, and $I_i(x)$ defines the image intensities sampled with the domain. However some photo-consistency measures do not need the support domain Ω. By using SBM we can find the pair correspondences x and x' in two images which are projections of the same 3D point X. The main idea of correlation based SBM is that, covering image into many windows and comparing the similarity of windows(blocks) between two images. Each window is a vector in an m^2 vector space. Due to epipolar constraint the searching space based on raster scan order is reduced to one dimension. The horizontal shift between correspondences is called disparity.
Similarity Measure

- Sum of Absolute Differences: $C_{SAD}(d) = \sum_{(i,j)\in W} \mid \hat{I}_L(i,j) - \hat{I}_L(i-d,j) \mid$
- Sum of Squared Differences: $C_{SSD}(d) = \sum_{(i,j)\in W} (\hat{I}_L(i,j) - \hat{I}_L(i-d,j))^2$
- Normalized Correlation: $C_{NC} = \sum_{(i,j)\in W} \hat{I}_L(i,j)\hat{I}_L(i-d,j)$

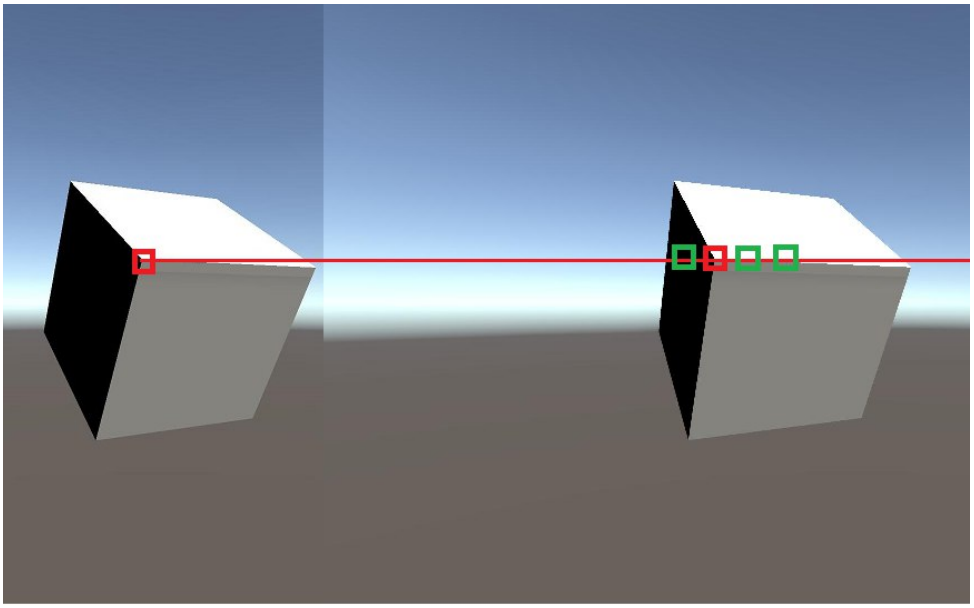


Fig. 2.18. An example of Stereo Block Matching for a cube. For the left chosen block, it will match the corresponding block in the right image sequentially and evaluate the match costs

Winner-Takes-All Strategy. As shown in Figure 2.19, a simple depth map reconstruction algorithm like NCC or SBM is to evaluate photo-consistency values

throughout the depth range (associates the epipolar line in the right image) and pick the depth value with highest photo-consistency score for each pixel independently.

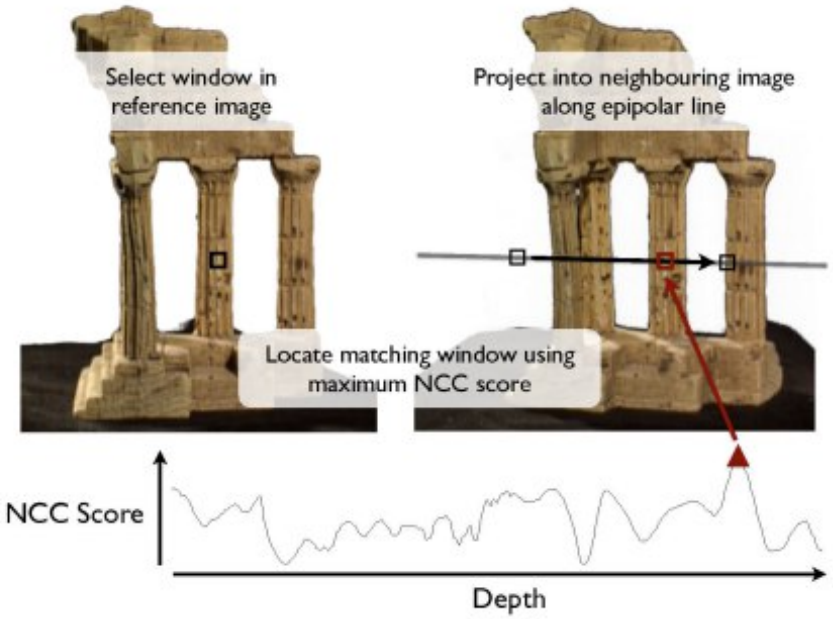


Fig. 2.19. Winner-takes-all strategy for depthmap reconstruction that evaluates photo-consistency values throughout the depth range (associates the epipolar line in the right image) and pick the depth value with highest photo-consistency score for each pixel independently. source: Multi-View Stereo: A Tutorial, page: 47

As the peak of a photo-consistency curve may not correspond to the true depth in challenging case such as textureless areas, repeated patterns, and specularities, an advanced algorithm is proposed to improve Stereo Block Matching so that disparity values change slowly.

Depth Map in form of MRF. A standard solution for these problem is to enforce spatial consistency under the assumption that neighbor pixels have similar depth values, where Markov Random Field (MRF)[23] is a very famous and successful formulation for this task and is a popular formulation for many other Computer Vision/-Graphic Problems as well. Since this problem is considered as MRF formulation, it can be solved as a combinatorial optimization problem, where the input depth range is discretized into a finite set of depth values. Through assignment for each pixel p a depth label k_p from depth label set, then minimize the following energy function:

$$E(k_p) = \sum_p \Phi(k_p) + \sum_{(p.q)\in\mathcal{N}} \Psi(k_p, k_q) \tag{2.20}$$

The first summation is range over all the pixels in the images and the second summation is over all the pairs of neighbor pixels denoted as $\mathcal{N}$.
From the main idea of MRF, in this pipeline the SGBM(Semi-Global-Block-Matching) algorithm is applied. The following 2 criteria are suggested by the author for a good correspondences:

1. Match Quality: similarity measure between windows
2. Smoothness: adjacent pixels should usually shift about the same amount of unit

Then define an energy function which is similar to MRF-formulation:

$$E(d) = E_d(d) + \lambda E_s(d) \tag{2.21}$$

where $E_d(d)$ is SSD distance between windows centered at (i,j) and $(i+d(i,j),j)$

$$E_d(d) = \sum_{(i,j)\in W} C(i,j,d(i,j)) \tag{2.22}$$

while $E_s(d)$ measures extent to which d is not piecewise smooth

$$E_s(d) = \sum_{(p,q)\in\varepsilon} V(d_p.d_q) \tag{2.23}$$

$$V(d_p,d_q) = \begin{cases} 0 & d_p = d_q, \\ 1 & d_p \neq d_q \ . \end{cases}$$

This optimization is in general NP-hard. However, there are still several ways to get an approximate solution typically. Here we using dynamic programming approximations. For that we define $DP(x,y,d)$ as the minimum cost of solution such that

$$DP(x,y,d) = C(x,y,d) + \min_{d'}\{DP(x-1,y,d') + \lambda \mid d-d' \mid\} \tag{2.24}$$

Energy functions of this form $E(d) = E_d(d) + \lambda E_s(d)$ can also be minimized using graph cuts.[50]
In particular this energy function form is applied in SGBM(Semi-Global Block Matching) Algorithm from OpenCV. [39]

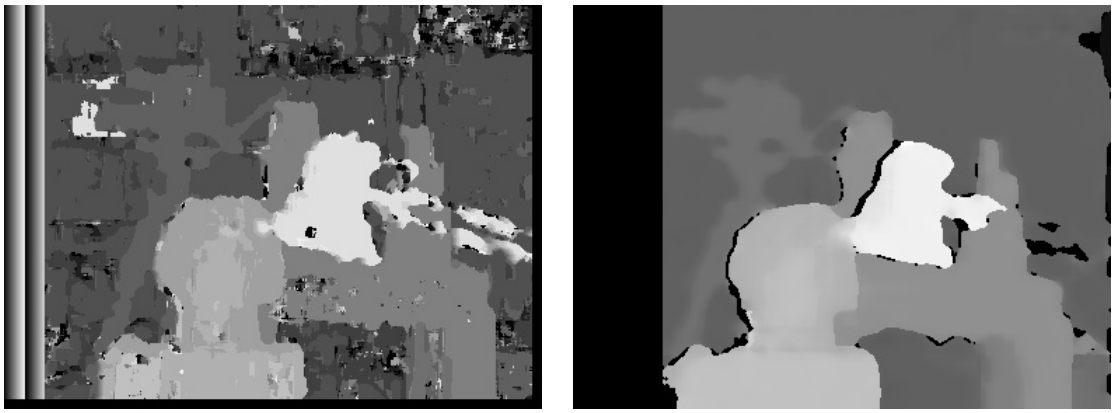
$$E(D) = \sum_p (C(P,D_p)) + \sum_{q\in N_p} P_1 I[\mid D_p - D_q \mid = 1] + \sum_{q\in N_p} P_2 I[\mid D_p - D_q \mid > 1] \tag{2.25}$$

where p and q are pixels, N_p are the neighbor pixels

General step of SGBM

1. Preprocessing
2. Cost Calculation
3. Dynamic Programming
4. Postprocessing

The following Figure 2.20 shows the performance using SAD and SGBM. Besides, the run time of SAD is usually faster than SGBM but with large number of noises.



(a) (b)

Fig. 2.20. (a) Depthmap using SAD with amount of noises (b) Depthmap using SGBM with less noises

As shown in Figure 2.21, the performance of SAD and SGBM is not so good in cube with such flat surfaces. The special treatment for such objects will be discussed in next chapter.

(a) (b)

Fig. 2.21. (a) Sparse point cloud using SAD (b) Sparse point cloud using SGBM

Mesh Reconstruction: Volumetric data fusion

Finally the 3D information of scene is obtained from the above section. Now the goal of this stage is to create a high quality mesh. For that, the volumetric surface extraction is applied to solve this challenging task. Seminal work from Curless and Levoy [9] proposes an algorithm to accumulate surface evidence into a voxel grid by using signed distance functions, in which the final mesh is extracted as the zero iso-surface of the aggregated signed distance functions. However, the Delaunay Triangulation is a good start for this section.

Delaunay Triangulation The technology of triangulation for a set of discrete points is significant in computer graphics.

Definition 2.1.1 (Triangulation)**.** A **triangulation**[11] of a finite point set $P \subset \mathbb{R}^d$ is a collection S of d-simplices, such that

(1) $conv(P) = \bigcup_{T \in S} T$; (Union Property)

(2) Any pair of these simplices intersects in a common edge; (Intersection Property)

Especially, due to the exceptional geometric properties, the Delaunay triangulation is recongnized as the best triangulation.

Definition 2.1.2. A triangulation of a finite point set $P \subset \mathbb{R}^d$ is called a **Delaunay triangulation**[12], if the circumcircle of every triangle is empty, that is, there is no point from P in its interior.

A three-dimensional Delaunay triangulation, denoted by DT, is a tetrahedralization that divides the convex hull of sample points into a collection of tetrahedrons. There are 3 common algorithms which can implement Delaunay triangulation. Here in this chapter involves only the Delaunay algorithm based on region growing space [7] proposed by Brassel and Reif in 1979. It is an algorithm to generate Voronoi diagramms for a set of n points defined in the plane.

A similar method in PCL library called triangulation greedy algorithm is suggested by the author, which is a projection of the original point cloud fast triangulation algorithm. The algorithm assumes a smooth surface, a uniform density of point cloud, and smoothing the surface can not be fixed in the holes while triangulation. This method is based on Delaunay algorithm on region growing space shown in Figure 2.22:

- A point V and its vicinity are projected onto a target plane through the normal, denoted as the point set $\{S\}$
- From $\{S\}$ forms all the edges between each two points and then sorted in ascend order of distance
- Add the shortest edge at each stage and remove it from the memory. If the edge satisfies the criteria of Delaunay-Triangulation, then it is added to the triangulation, otherwiese, it is removed.
- obtaining a triangular mesh model according to the topological connection relations of points in the plane

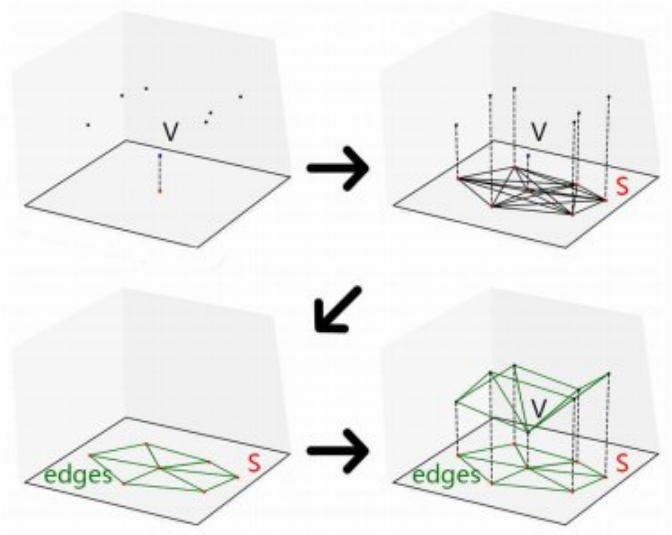


Fig. 2.22. Greedy projection triangulation on the 3-space [7]

Volumetric Graph-Cuts on a Voxel Grid. A bounding box that contains the solution surface is selected and so the space is partitioned with a voxel grid. Generally the input 3D information such as a set of depthmaps can be analyzed to determine a bounding box. Then a label is assigned to each voxel, as interior or exterior, where the label boundary can be extracted as a surface model. The objective is to find the formulated optimal label assignment to all the voxel that minimizes the following cost function[7]

$$E(\{k_v\}) = \sum_v \Phi(k_v) + \sum_{(v,w)\in\mathcal{N}} \Psi(k_v, k_w). \tag{2.26}$$

where k_v is a variable from one of these two labels for a voxel, $\Phi(k_v)$ denotes the cost of assigning a label to voxel v. The first term defines the summation of per voxel cost over the entire domain. The second term encodes the summation over all pairs of adjacent voxels denoted as $\mathcal{N}$. From the above section we know it is similar to the Markov Random Field formulation for depthmap reconstruction. Φ depends on only one variable, while Ψ is a pairwise interaction term.

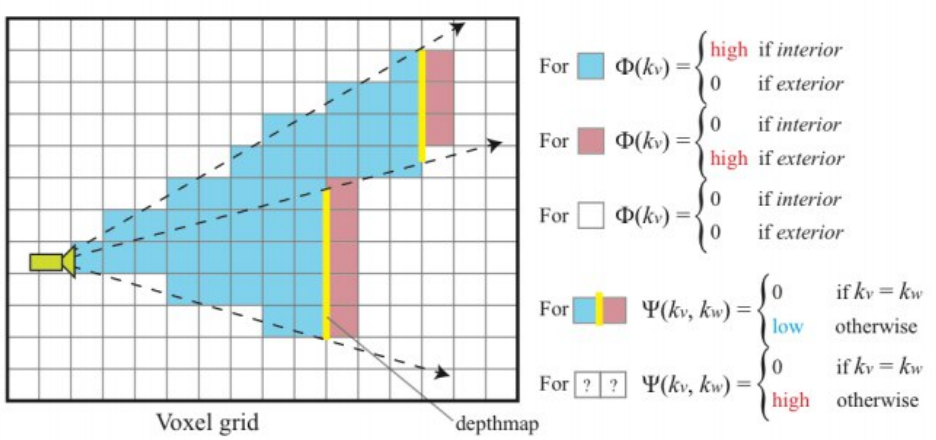


Figure 3.21: An example of how 3D MRF cost function should be set from a single depthmap.

Fig. 2.23. source: Multi-View Stereo: A Tutorial Color

Suppose the simple condition of a single depthmap. $\Phi(k_v)$ should be set so that the penalty of assigning becomes high for an unsuitable label. In addition, $\Psi(k_v, k_w)$ can be set as a smoothness adaptive control, which means that it is assigned a penalty when $k_v \neq k_w$.

Optimization. The Problem can be solved exactly and efficiently with a graph-cuts algorithm, if for each neighbour their pairwise term $\Psi(k_v, k_w)$ is submodular:

$$\begin{aligned}&\Psi(interior, interior) + \Psi(exterior, exterior) \leq \\ &\Psi(interior, exterior) + \Psi(interior, exterior)\end{aligned} \tag{2.27}$$

Pairwise terms satisfy the above condition in most case of photogrammetric reconstruction. Figure 2.24 shows one of the earliest volume fusion techniques based on the volumetric graph-cuts, which uses the constant ballooning term and takes the photo-consistency volume as the input. However, one problem is that the memory allocation quickly becomes very expensive during computing. Hence, a more effective

method will be used in the proposed working-pipeline, which will be introduced in next section.

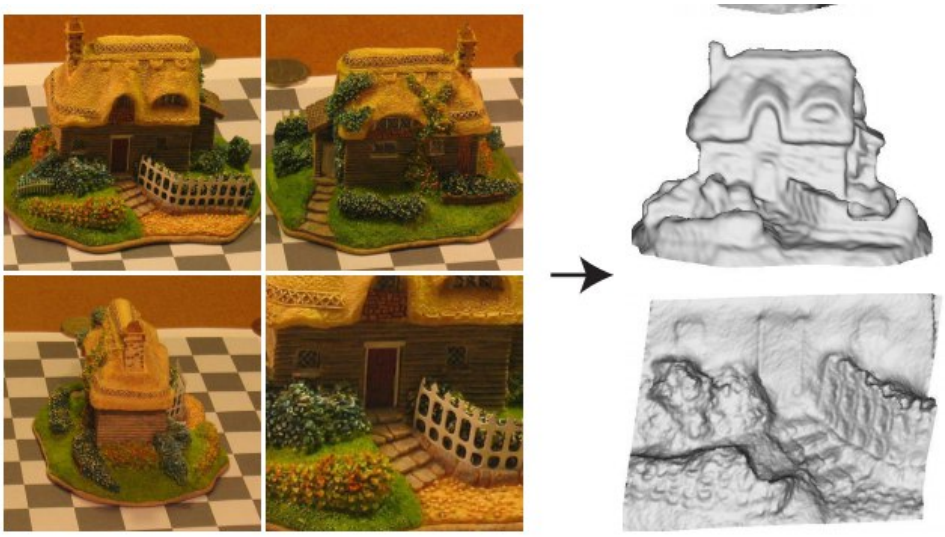


Fig. 2.24. An example of volume fusion. source: Vogiatzis, Torr and Cipolla(2005)

Volumetric Graph-Cuts on Delaunay Tetrahedralization

The main idea of this method is to first reconstruct a sparse 3D point cloud of a scene, then apply the 3D points as the nodes of the space partition grid. Moreover, this space discretization is adaptive to the current 3D information. A space with denser point is partitioned more densely, on the contrast, a space with sparse point will be discretized sparsely.

According to that, Labatut, Pons, and Keriven presented an algorithm that first matches image features based on the SIFT descriptor to reconstruct sparse 3D point cloud (see previous section), then performs the 3D Delaunay triangulation of the points to partition the space[37]. Figure 2.25 shows the 2D illustration of Delaunay triangulation.

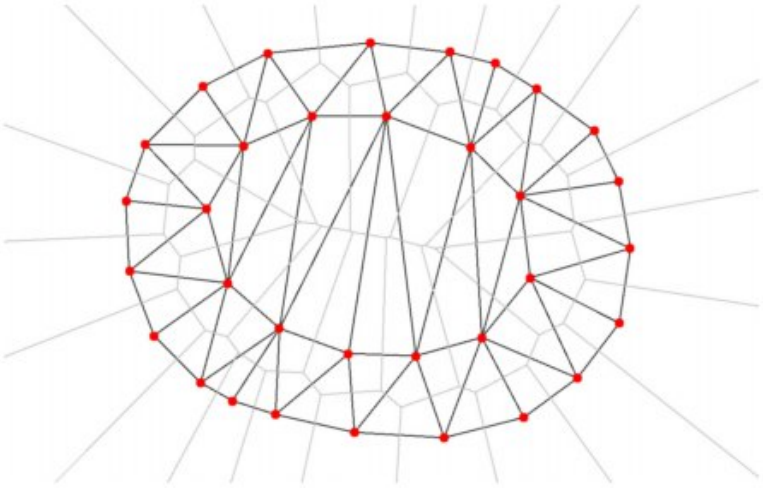


Fig. 2.25. Delaunay tetrahedralization on 3D point cloud, where red points define the input point samples and divide the reconstruction space into a set of triangles. (Figure courtesy of Labatut et al. Multiple View Stereo: Tutorial, page: 83)

Furthermore, given a 3D Delaunay triangulation, a volumetric MRF problem can be solved that is related to the one in previous section of extract a surface model. The volumetric MRF consists of the unary and pairwise terms as usually, where the unary term is defined for each tetrahedron while the pairwise term is defined for each face shared by two tetrahedra. In the work of Labatut, for each ray connecting a reconstructed point and one of the camera which can see the point, the setting of label is as follow:

$$\Phi(p_0) = \begin{cases} \infty & \text{if } \textit{interior}, \\ 0 & \text{if } \textit{exterior} \ . \end{cases}$$

$$\Phi(q_2) = \begin{cases} 0 & \text{if } \textit{interior}, \\ \alpha(\textit{constant}) & \text{if } \textit{exterior} \ . \end{cases}$$

where p_0 defines the triangle that contains the cameracenter and q_2 denotes the triangle that behind the 3D point. The term in q_2 adds a bias for the cell in the scene interior space. For each pair of triangles(p_1, q_1) whose common face intersects with the ray, where p_1 is notably on the camera side.

$$\Psi(p_1, q_1) = \begin{cases} \beta & \text{if } p_1 = \textit{interior}, q_1 = \textit{exterior}, \\ 0 & \text{otherwise} \ . \end{cases}$$

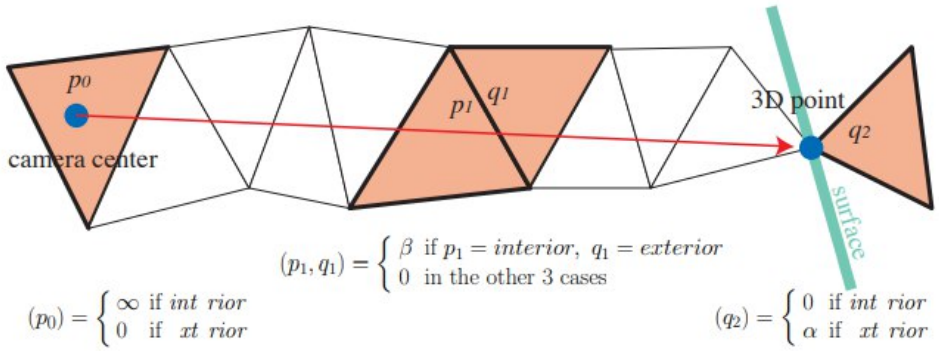


Fig. 2.26. The setting of labels for each ray connecting a reconstructed point and one of the camera which can see the point. Source: Multi-View Stereo: A Tutorial, page: 83

In the process of computing during every ray between a 3D point and its visible camera, the above two different terms are accumulated as opposed to overwriting. In light of the pairwise term construction, the above energy is guaranteed to be sub-modular. Hence, the graph-cuts algorithm can be applied to find the optimal label assignment for triangular cells, where the cell labeled as boundary is extracted as a surface model.

Figure 2.27 shows some reconstruction examples of the Delaunay tetrahedralization technique proposed by Labatut, Pons, and Keriven [37]. Despite that the reconstructed surfaces may look slightly noisy, this is their mesh initialization step, which is the starting step for the refinement step introduced in the next section.

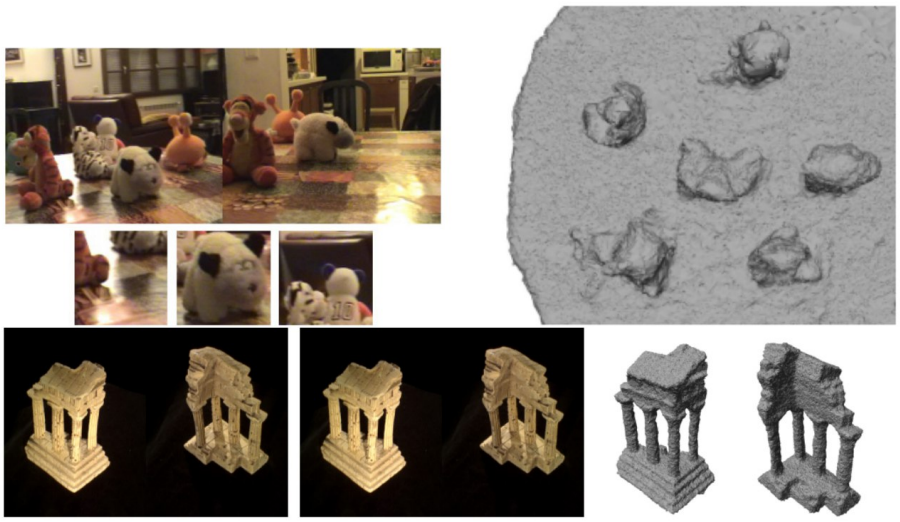


Fig. 2.27. Reconstruction results of a Delaunay tetrahedralization based volumetric graph-cuts approach. Source: Multi-View Stereo: A Tutorial, page: 83

Mesh Refinement As the last step of MVS, in the refinement step, all the images are used once again for evaluation of the photometric consistency score over the surface, in which the vertex position are iteratively optimized to increase the score. Mesh refinement algorithms move the position of each vertex v_i step by step or simultaneously, while minimizing an objective function $E(\{v_i\})$, which is defined over the mesh model.

$$E(\{v_i\}) = E_p(\{v_i\}) + E_r(\{v_i\}) + [E_s(\{v_i\})] \tag{2.28}$$

where $E_p(\{v_i\})$ is related to a photometric consistency measure and $E_r(\{v_i\})$ defines a regularization term which measures the smoothness of mesh. What's more, if the input of images involves the information of silhouettes, then a silhouette consistency measure E_s is suggested to be added to make the mesh consistent with the image silhouettes.
In this situation, gradient descent is usually considered as the method of optimization, in which the movement of each vertex at each iteration is determined by the gradient of the objective function. Let (x_i, y_i, z_i) be the coordinates of vertex v_i, then the gradient of the energy for each v_i is

$$\left(\frac{\partial E}{\partial x_i}, \frac{\partial E}{\partial y_i}, \frac{\partial E}{\partial z_i}\right)$$

Photometric Consistency Term. According to the work of Lastly, Hiep and Keriven[28] defined the term as a sum of photometric consistency scores over all the faces of a discrete triangulated mesh model:

$$E_p(\{v_i\}) = \sum_{f \in \mathbb{F}} \sum_{(I_i, I_j) \in \mathbb{V}_f} \mathcal{C}(f, I_i, I_j) \tag{2.29}$$

where $\mathbb{F}$ is the set of faces in the mesh model, $\mathbb{V}$ denotes a set of image pair in which face f is visible and $\mathcal{C}(f, I_i, I_j)$ defines the photometric consistency score measured on face f based on the two image I_i and I_j.

Regularization Term. Despite that there exists various definition of regularization term, the Mesh Laplacian and Mesh Bi-Laplacian are commonly used to define

the regularization force[6]. Mesh Laplacian operation $\delta(v_i)$ produces a 3D vector at v_i

$$\Delta(v_i) = \frac{\sum_{v_j \in \mathcal{N}_i} v_j}{|\mathcal{N}_i|} - v_i \tag{2.30}$$

$\Delta(v_i)$ naively computes the difference vector form v_i to the center of its neighbor. It is forceful in vertical to make vertices coplanar and also has strong force in lateral direction to make vertices uniformly distributing. Bi-Laplacian operation is similarly defined as follow

$$\Delta^2(v_i) = \frac{\sum_{v_j \in \mathcal{N}_i} \Delta(v_j)}{|\mathcal{N}_i|} - \Delta(v_i) \tag{2.31}$$

which can be considered applying the Mesh Laplacian operation 2 times.
The Firgure 2.28 shows examples for 1D surfaces. Laplacian becomes zero in the situation of planar surface, while Bi-laplacian becomes zero with constant or uniform curvature of the surface.

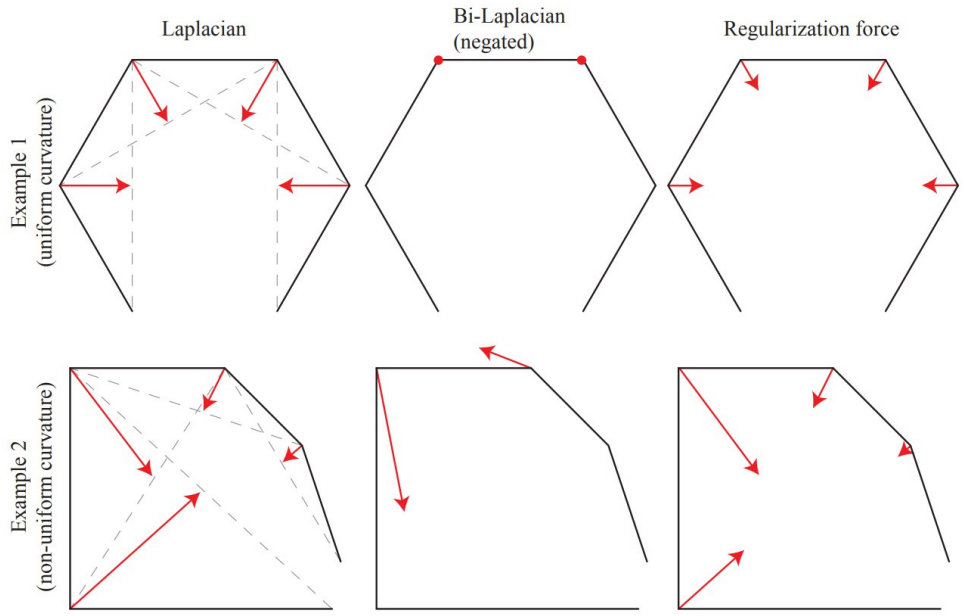


Fig. 2.28. Laplacian becomes zero in the situation of planar surface, while Bi-laplacian becomes zero with constant or uniform curvature of the surface. Source: Multi-View Stereo: A Tutorial

Since the Bi-Laplacian force is much weaker than the Laplacian force, the linear combination of the two operations is commonly defined to be the regularization force at each vertex v_i

$$\alpha\Delta(v_i) - \beta\Delta^2(v_i) \tag{2.32}$$

where α and β are weights factor of linear combination that can be typically set around 0.5.

Silhouette Consistency Term The silhouete consistency simply enforces that an image silhouette of the reconstructed mesh matches the input image silhouette. According to the work of Hernández and Schmitt, the silhouette consistancy force is considered as a soft constaints for each vertex v_i along the surface normal direction n_i

$$(d_S(v_i) \cdot \alpha(v_i))n_i \tag{2.33}$$

$d_S(v_i)$ is given as minimum distance between an image projection of a vertex v_i to the visual hull surface on the image domain over all the input image, where the distance is negative if the projection is outside the silhouette and positive in the contrast condition:

$$d_S(v_i) = \min_{I_j} d_S(S_j, v_i) \tag{2.34}$$

where S_j defines an image silhouette related to image I_j(see Figure 2.29).

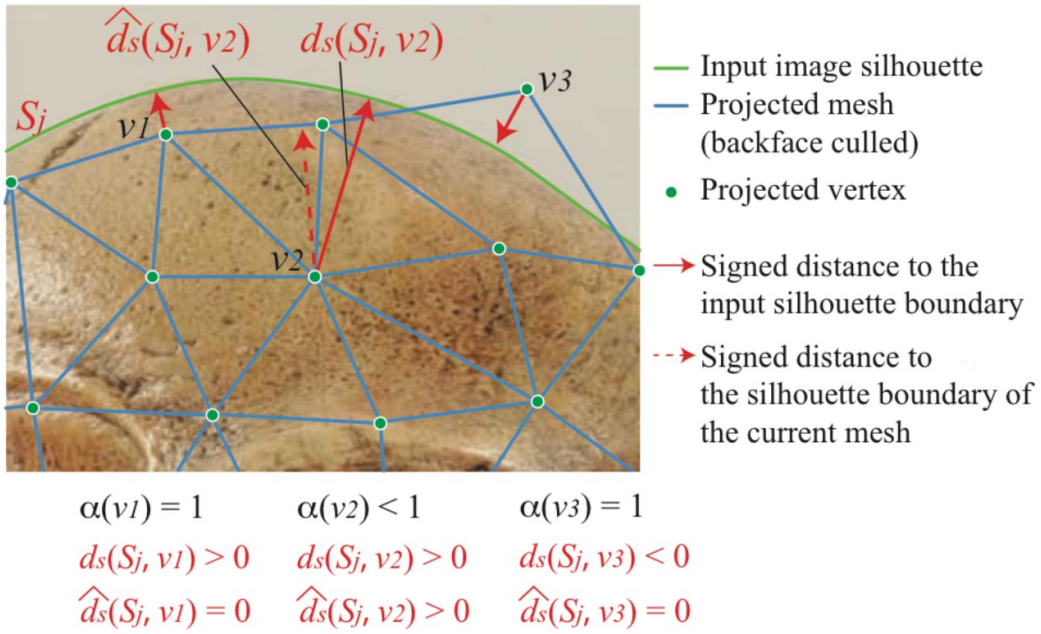


Fig. 2.29. Meshrefinement: minimum distance between an image projection of a vertex to the visual hull surface on the image domains over all the input image, where the distance is negative if the projection is silhouette. source: Multi-View Stereo: A Tutorial,page 102

Never the less, the first term drives the mesh to visual hull model, while we do not need all the vertices to be on the boundary of image silhouettes, and so $\alpha(v_i)$ is defined to be

$$\alpha(v_i) = \begin{cases} 1 & \text{if } d_S(v_i) \leq 0, \\ \frac{1}{(1+\hat{d}_S(v_i))^n} & \text{if } d_S(v_i) > 0. \end{cases}$$

There are many choices to make when building a mesh refinement algorithm. The latest and most successful state-of-art by Vu, Labatut and Pons [16] in 2009 is used in the proposed pipeline by the author. The photometric consistency term is defined over the polygonal mesh model and their system can deal with arbitrary datasets. The regularization term only exploits the bi-Laplacian operation and α is set to 0. The input mesh model to the final step of refinement is generated by reconstructing a sparse point cloud, then using a volummetric graph-cuts technique with Delaunay tetrahedralization, which is already introduced in the previous section.

Affine To Metric Reconstruction

Considering the motion of the cameras in our case, we obtain an affine reconstruction from above method. However, we still need to progress our step to metric reconstruction. Since the key to metric reconstruction is the identification of the absolute conic ω_∞, the next step will be to apply the affine transform to the affine reconstruction so that the identified absolute conic in the standard Euclidean frame. The resulting reconstruction is then related to the true reconstruction by a projective transformation which fixes the absolute conic.
In practice we consider the image of the absolute conic in one of the images. Subsequently we will show how the image of the absolute conic ω can be used to define the homography H which transforms the affine reconstruction to a metric reconstruction.

Lemma 2.1.2. *Suppose that the image of the absolute conic is known in some image to be ω, and one has an affine reconstruction in which the corresponding camera matrix is given by $P = [M \mid m]$. Then the affine reconstruction may be transformed to a metric reconstruction by applying a 3D transformation of the form*

$$H = \begin{bmatrix} A & 0 \\ 0^{(3)} & 1 \end{bmatrix} \tag{2.35}$$

where A is obtained by Cholesky factorization from the equation $AA^T = (M^T \omega M)^{-1}$

Based on Hartley and Zisserman's result, we also have

Let v_1 and v_2 be the vanishing points of two lines in an image, and let ω be the image of the absolute conic in the image. If θ is the angle between the two line directions, then

$$\cos\theta = \frac{v_1^T \omega v_2}{\sqrt{v_1^T \omega v_1}\sqrt{v_2^T \omega v_2}} \tag{2.36}$$

Indeed, this manner to metric reconstruction depends on identifying the image of the absolute conic. Following are 2 ways of doing this, which are mainly applied in our case

1. **Constraints arising from scene orthogonality** Vanishing point pairs v_1 and v_2 , which results from orthogonal scene lines, places a single linear constraint on ω:

$$v_1^T \omega v_2 = 0$$

2. **Constraints arising from the same cameras in all images** One of the properties of the absolute conic is that its projection into an image relies not on the position or orientation of the camera, but only the calibration matrix. Since in our case, all cameras have the same calibration matrix one has that $\omega = \omega'$, which means that the image of the absolute conic is the same in both images.Due to the absolute conic lies on the plane at infinity, the image may be transferred from one view to the other through the infinite homography. Hence,

$$\omega' = H_\infty^{-T} \omega H_\infty^{-1} \tag{2.37}$$

The prerequisite of forming this equation is that we already obtain an affine reconstruction.

(2.36) implies a set of linear equations for the entries of ω. Generally it places four constraints on ω, while ω has five degrees of freedom. Since ω is not completely determined, we need combine these linear equation with those above provided scene orthogonality.

Texture Mapping For the reconstruction of texture based on mapping, the first step is input. Obviously a reconstructed mesh can be obtained from the above steps, which contains vertices and faces, and each vertex is determined by a set of 3 dimensional coordinates, each face is composed of 3 vertex index number; in addition, we also have a set of images used to reconstruct the grid and the estimated internal/external camera parameters corresponding to each image in SfM of the first step. Then comes the output. In addition to vertex and face, a renderable model must at least contain the texture coordinates corresponding to the texture map(atlas). During rendering of the model, the color of each face will be determined, by finding the pixels in atlas corresponding to the 3 vertices.Then the area determined by the 3 pixels are deducted and paste it on the face.

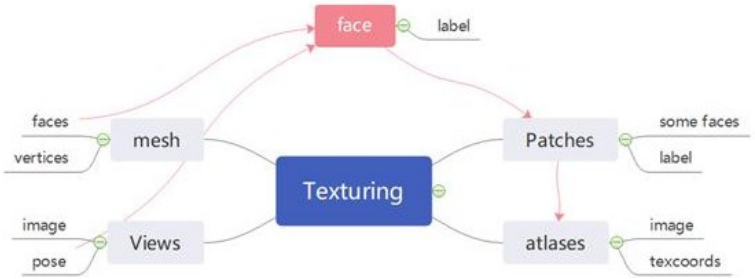


Fig. 2.30. Reconstructed mesh containing vertices and faces. Each vertex is determined by a set of 3 dimensional coordinates, each face is composed of 3 vertex index number. The output is a texture map (atlas)

the selection of views. The objective of this step is to select an appropriate view for each face to determine the color expression of the face. Intuitively, the label should satisfy the following conditions:

- the face is visible from this view
- the front of the face can be seen from this view
- the normal direction of the face is close to the direction parallel to the sight
- the image range of the view includes the area where the face is located

Based on the above conditions, intuitively, there are various criteria from different algorithms: the triangle area mapped by the face from view as the criterion; the angle between the line of ray and the face normal. In addition, considering that on the basis of satisfying the above criterion, the corresponding perspective should be able to express enough content as much as possible without ignoring the texture details on surface. By using MRF-formulation we have an energy function to optimize

$$E(l) = E_d + \lambda E_s \tag{2.38}$$

E_d is the main cost

$$E_d = -\sum_{i}^{m}\sum_{f}^{n} Q(f_i, l, j) \tag{2.39}$$

where f_i defines face and l_j is the label corresponding to view j. Q denotes the evaluation to quantize the criterion. On the other hand, considering the boundary problem, adjacent faces may choose the same view, so we have another penalty term:

$$E_s = \sum_{f_i, f_j \in adjacencies} S(f_i, f_j, l_i, l_j) \tag{2.40}$$

where S denotes the penalty function.
The entire mesh can be simplified to an undirected graph where faces are considered as nodes and the relationships between each two faces denotes as edges. The above problem can be described as the optimization problem of MRF, for that there exists various algorithms to solve MRF problem[23] [9]. **Generate Texture**. After selecting the view for each face, the faces share same view can be considered as a patch for better performance and continuity of the color. Then all patches are sequentially assigned an area in atlas and map each vertex in mesh to corresponding pixel in atlas. Specifically, for each patch a related area will be extracted from the corresponding view to generate a sub-image, then compute the coordinates in this sub-image for all the vertices of the patch. After that, every sub-images related to each patch will be put into the initial atlas. This process can be considered as a bin packing problem. OpenMVS used the MVE's [12] method to adjust the texture patch color and packed the image information corresponding to the texture patch to out put the atlas. **Texture Patch Boundary Smoothing**. The texture patch generated by graph-cut would lead to a jagged boundary. A smoothing method needs to be adopted to solve the problem. Since the seam comes from that two adjacent patch are mapped to 2 different views, where the light and other nature influence are different. Hence the main idea is to modify the color of pixels corresponding to two patches, so that the discontinuity can be mitigated.
The principle of Lempitsky[11]. To make the color of two patches more closer, the following target function must be optimized with new labeled color for vertex v in seams:

$$J = \min_g \sum_{v \in seams} (f_{v_l} + g_{v_l} - (f_{v_r} + g_{v_r}))^2 \tag{2.41}$$

where f is the color of v in different patches, while g is the modified color of v. However, considering the big difference of modified color between two adjacent vertices, an abnormal color occurs inside the patch. Therefore we add a constraint for adjacent vertices v_i and $v_j$in a same patch:

$$J = \min_g \sum_{v_i, v_j \in patch} (g_{v_i} - g_{v_j})^2 \tag{2.42}$$

In general, after combining those 2 part, the final optimization will be :

$$J = \min_g [\sum_{v \in seams} (f_{v_l} + g_{v_l} - (f_{v_r} + g_{v_r}))^2 + \frac{1}{\lambda} \sum_{v_i, v_j \in patch} (g_{v_i} - g_{v_j})^2] \tag{2.43}$$

which can be simplified to matrix form:

$$J = \|Ag - f\|_2^2 + \|\Gamma g\|_2^2 \quad = g^T(A^T A + \Gamma^T \Gamma)g - 2f^T Ag + f^T f \tag{2.44}$$

where f denotes the vector implicating all the color differences of seam vertices, g defines the vector contains the element of modified color for all the vertices, A and Γ are the sparse matrices to select the seam vertex in different patch or adjacent vertices

in a same patch. The conjugate gradient method is recommend to solve the above function.

2.2 Mixed Reality

From the stages above, it is clear how an object is recovered from multiple view. Then it is meaningful to view and check the reconstructed virtual object in the real 3-space environment comparing with the original object. Since the parts that how a scene is projected into 2D-images and the 2D-2D, 2D-3D correspondences matching, are already discussed in the previous chapter, it is trivial to use these technology to present a virtual reconstructed object in a 3 dimensional space and project into a screen according to the motion of viewpoint per frame. Moreover, moving a reconstructed virtual object somewhere in the real 3-space also changes the presentation in screen. Hence, a term called mixed reality is applied in the proposed pipeline, which presents the virtual objects in a virtual environment or in a real 3 dimensional space, so that the users can easily check their results from the stages above and reuse the technology in a further field.

Mixed Reality is a blend of physical and digital worlds, unlocking the links between human, computer, and environment interaction. This new reality is based on advancements in computer vision, graphical processing power, display technology, and input systems. Figure 2.31 shows the interactions between computers, humans, and environments.

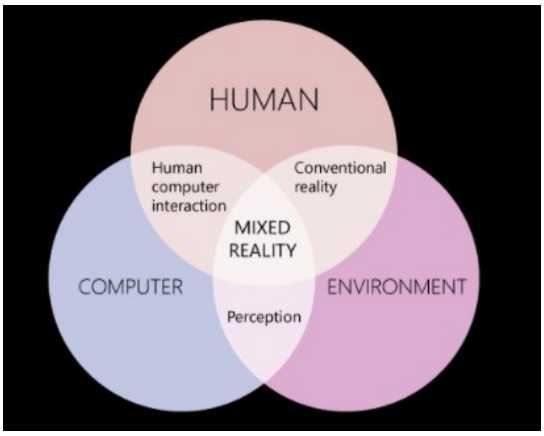


Fig. 2.31. Mixed Reality: the interactions between computers, humans, and environments. Soruce: Mixed Reality Docs

All the 3 parts together are necessary for creating true Mixed Reality experiences. Motion through the physical world translates to movement in the digital world.
The relationship between Augmented Reality, Virtual Reality and Mixed Reality are shown in Figure 2.32. In general, AR can be considered as a subset of Mixed Reality.

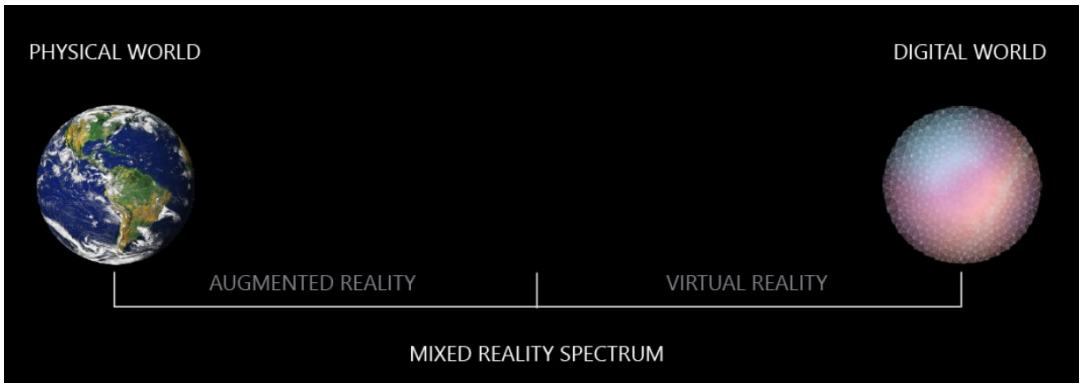


Fig. 2.32. Mixed Reality and the relationship between Augmented Reality, Virtual Reality and Mixed Reality, soruce: Mixed Reality Docs

Virtual reality (VR) places user inside a completely computer-generated environment, while augmented reality (AR) aims to visual information that is directly registered to the physical environment. The research on AR lasts at least decades. According to the survey paper of Azuma [2], AR must contain the following 3 characteristics:

- Combines real and virtual
- Interactive in real time
- Registered in 3D

For that, real time control and spatial registration, real time alignment of corresponding virtual and real information indeed, are required. Furthermore, the user can exercise some sort of interactive viewpoint control such as gaze, gesture using an AR-display. The computer-generated augmentation-components in the display will remain registered to the associated objects in the environment.
The qualification of real time performance relies actually on the task or application and the interaction means that the human-computer interface runs in a highly coupled feedback loop by continuously navigating the AR scene and controlling the AR experience. Conversely, the AR-system picks up the user's input by tracking the user's viewpoint or pose, then the pose will be registered in the real world with the virtual content, and presents to the user a situated visualization. For this purpose, the SIFT algorithm , which has been introduced in the previous section, is applied for 3D object recognition and tracking in context of augmented reality, where accurate pose are superimposed on real images [15]. SIFT matching is done for a number of 2D images of a scene or object taken from different angles. It builds a sparse point cloud of the viewed scene and simultaneously recover camera poses and calibration parameters using BA initialized from an essential matrix. Then the position, size, orientation of the virtual object are defined relative to the coordinate frame of the recovered model. In the condition of online match moving, SIFT features again are extracted from the current frame and matched to the features that are already computed for the world mode, resulting in a set of 2D-to-3D correspondences. These correspondences are then used to compute the current camera pose for the virtual projection by solving PnP problem (see previous page) and final rendering.
Figure 2.33 shows that, a complete AR system must contains 3 components: a tracking component, a registration component, and a visualization component. In addition, a forth component called spatial model, stores the information of the real world and

the virtual world. The real-world model is considered as serving as a reference for the tracking component, which determine the user's location in the real world. On the other hand, the virtual-world model consists of the content that are used for the augmentation.

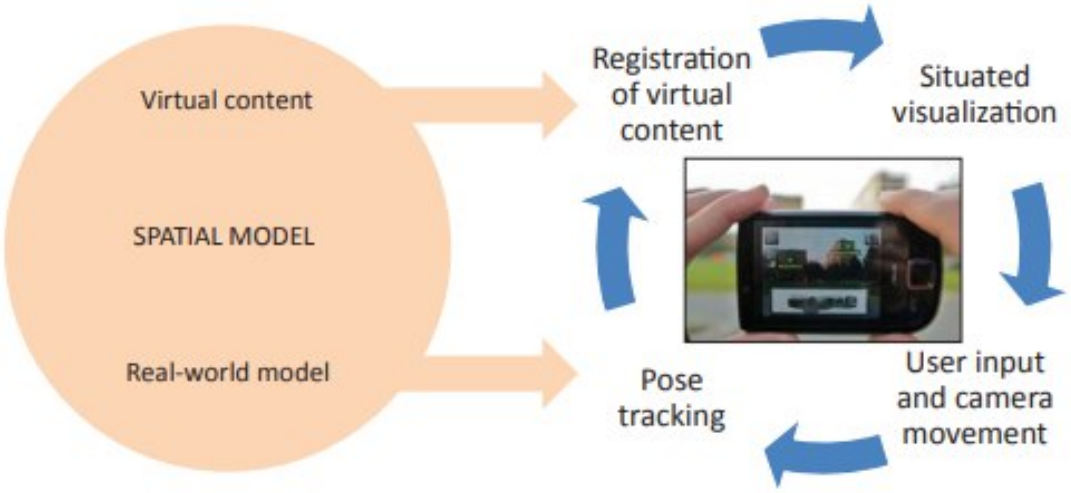


Fig. 2.33. A feedback loop between human and computer system, soruce: A survey of agmented reality

Chapter 3

Working pipeline of photogrammetric reconstruction

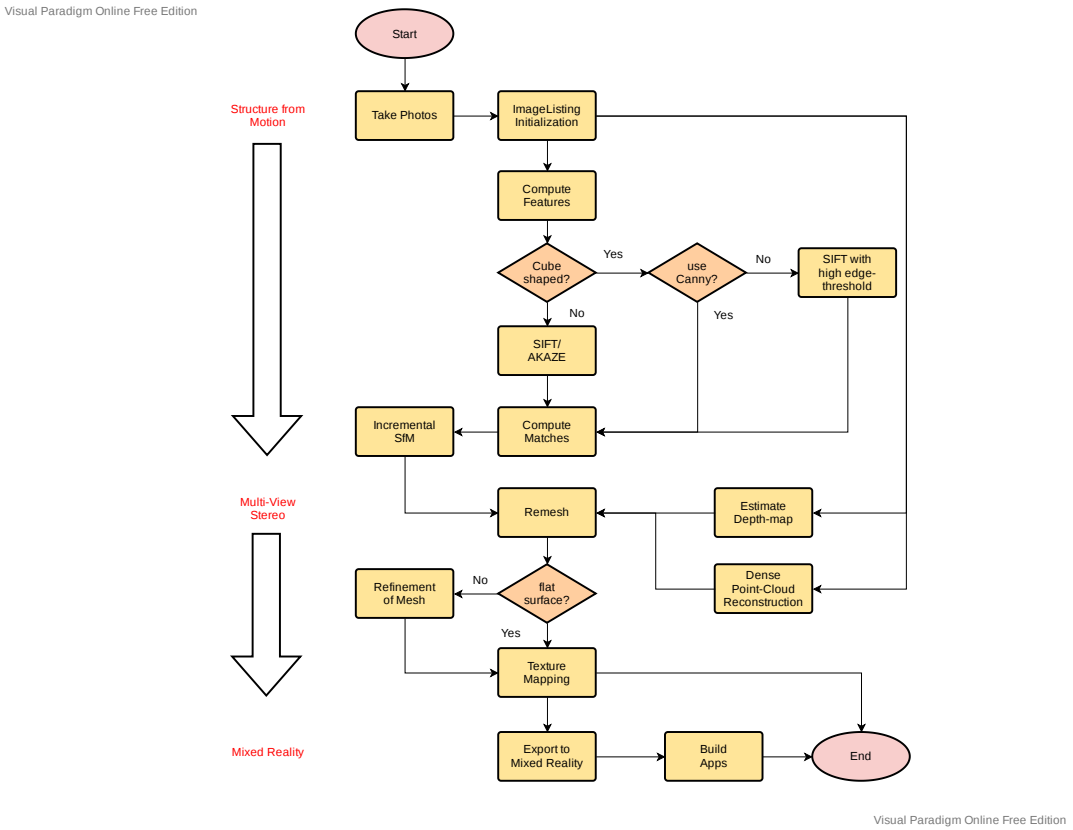


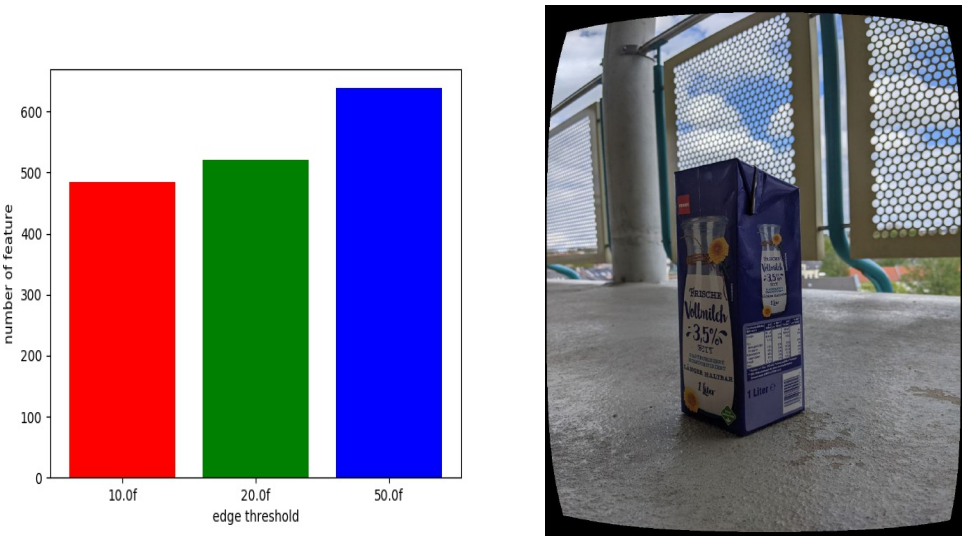
Fig. 3.1. The proposed pipeline is based on SfM, MVS and Mixed Reality

The general theories involve 3D photogrammetric reconstruction are already introduced in Chapter 2. In this chapter the key is building the reconstruction working pipeline for geometrically simply objects and then visualize it in mixed reality.
Figure 3.1 shows the structure of the proposed pipeline. Given a set of images as input, after the whole processes of pipeline, a 3D model with high quality is obtained. This section will focus on the challenges of building pipeline and some special tricks and treatment to solve those problems. Besides, some details of some steps in this pipeline will be also discussed.

The first stage in this pipeline (see Figure 3.1) is to take photos of a certain scene at different camera positions. To be clear that the number of photos and the relationships between each two photos influence the quality of final output. From the author's experiences in test cases, the angle between two rays from the camera center to the middle of the object is suggested to be 15°, which remains most of the share contents and the depth tolerance is acceptable. Then build an image dataset and store in a .json file. This structured file lists for each images an object called a view, which stores information and lists including the image name, size, and the internal camera calibration information (if has one).
Then comes the features computing. The objective of this step is to computes image description for a given .json file, that is, for each view it compute the image description and stores them on disk. The general principles of this step are already talked about in Chapter 2. Nevertheless, most of the general pipeline uses the SIFT algorithm and according to the work of Lowe [25], the parameter of edge-threshold is recommended to set as 10.0f. As a result, it will eliminate the edge effect and filter many of the point on edge. That is not suitable for our case such as reconstruction a cube, because it is reasonable to save edge points as many as possible for a cube, so that the feature matching will proceed with enough correspondences.

Here the author gives 2 solutions:

- set the edge-threshold higher than 10.0f
- use canny detection to save most of the edge points



(a) (b)

Fig. 3.2. (a) Results of SIFT using different values of edge-threshold from an image of milk (b) Original image about a box of milk

Figure 3.2 shows that the edge threshold is higher, we get more feature points. However, a high edge threshold might lead to a noise problem. The parameter depends on different situation and has to be optimized manually.

Canny Edge Detection is a multi-stage algorithm. Since edge detection is sensible to noise in the image, the first step is to remove the noise in the image with a

5×5 Gaussian filter. Then the smoothened image is filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction G_x and vertical direction G_y. Furthermore, we can find the edge gradient and direction for each pixel as follows:

$$\begin{aligned} G &= \sqrt{G_x^2 + G_y^2} \\ \theta &= \tan^{-1}\left(\frac{G_y}{G_x}\right) \end{aligned} \tag{3.1}$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions. After this, a full check of image is done to remove any unwanted pixels which may not constitute the edge. For this purpose, each pixel is checked if it is a local maximum in its neighborhood in the direction of gradient (see Figure 3.3).

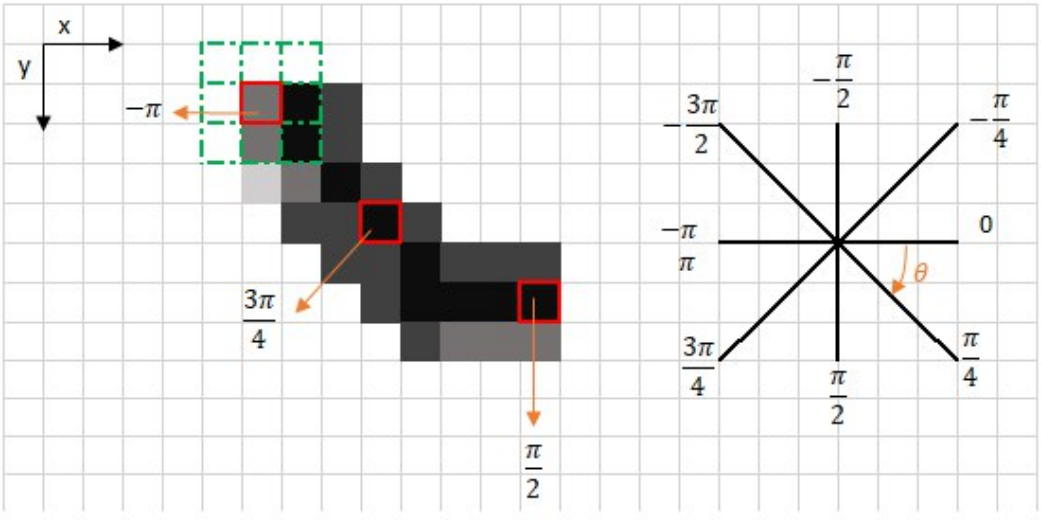


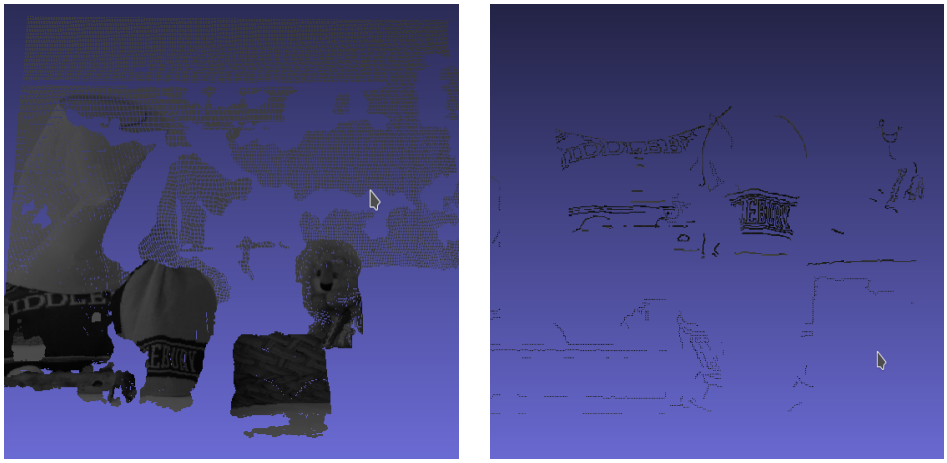
Fig. 3.3. The algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions. The orange arrows denote the gradient direction of intensity. source:towards data science

The upper left red box in the above image represents an intensity pixel of the gradient intensity matrix being processed. The orange arrow with an angle denotes the corresponding edge direction. For this pixel, its left neighbor is more intense, then only the more intense one is kept for next stage, and the rest is suppressed (put to zero). Next we set two thresholds minVal and maxVal. Any edges with the intensity gradient more than maxVal are sure to be edges and those below minVal are considered as non-edge. For those who lie between the two thresholds, we check if they are connected to the "sure-edge" pixels in the neighborhood. If yes, they are considered as the part of edges. Figure 3.4 shows a result of canny edge detection.

(a) (b)

Fig. 3.4. (a) Original image with clear contour (b) Only the points on the edges remain using canny detection.

Figure 3.5 (b) shows the reconstruction of point cloud only for the point which detected by Canny as an edge, while (a) is the result of normal reconstruction using SIFT detector.



(a) (b)

Fig. 3.5. (a) Point cloud using normal feature detection (b) Point cloud remaining only points on edges. all most all the edges have been detected and in addition, due to the light some boundaries with large intensity gap is considered to be edge according to canny

If an object is sphere shaped, then the AKAZE algorithm is in our pipeline recommended instead of SIFT. According to the work by Oskar Andersson and Steffany Marquez[1], they compared SIFT, KAZE, AKAZE and other popular methods on different criterion. One of the results is shown as follow:

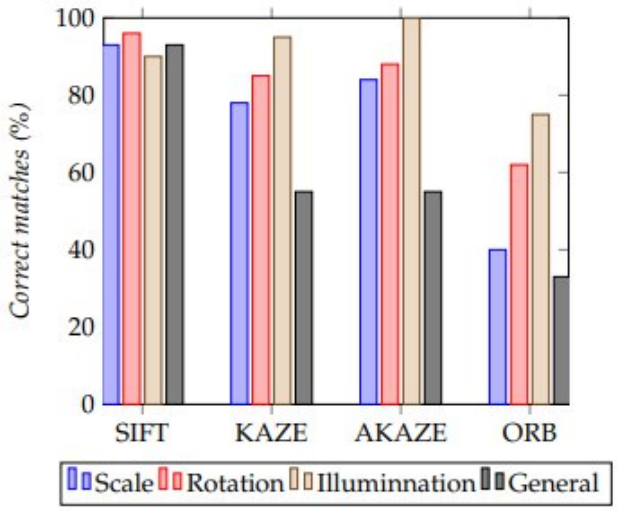


Fig. 3.6. source: A comparison of object detection algorithms using unmanipulated testing images, Oskar Andersson

Figure 3.6 shows that the AKAZE algorithm has a better performance than other techniques regarding varying illumininations situations. Since the surface of sphere is sensitive to illuminnation, AKAZE is applied in our pipeline for sphere shaped object.
Using the image descriptions computed by the last step, then the goal in 3rd step is to establish the corresponding putative photometric matches and filter the resulting correspondences using robust geometric filters (fundamental matrix, essential matrix, or homography matrix filtering).
After that, the incremental SfM starts. It begins from an initial two-view reconstruction that is iteratively extended by adding new views and 3D points using pose estimation and triangulation. Since choosing a good initial image pair is critical for the quality of the final product, here we recommend to choose the image pair with numerous correspondences while keeping a wide enough baseline.
After all procedures of SfM-phase done, the output is a scene graph and a sparse point cloud that stored in a .bin file. Then comes the phase of Multi-View Stereo. First, MVS obtains all the information from the .bin file and then starts to estimate the depth map for each view. After that, the reconstruction of the corresponding dense point cloud will proceed using those depth map above. Further, the step of remeshing is to reconstruct the mesh of the object using Delaunay Tetrahedralization. This part has been discussed in Chapter 2. After the above step done, now step of refinement of the mesh depends on the shape of the object. If it is cube shaped, the author suggest to avoid remeshing, since the refinement of mesh may lead to a uneven surface. Otherwise, the refinement is recommended to proceed after raw remeshing, as long as the surface of object is curved, such as sphere shaped object.
After texturing, the reconstructed object is then imported into Unity or Unreal Engine for further development in mixed reality. Finally, the output is an executable project by users, such as App in Android devices or Microsoft HoloLens.

Due to the fact that Nvidia has dominated the scientific research area in AI and Computer Graphics, most of the common libraries support only CUDA + Nvidia graphics cards. Nevertheless, we still want our pipeline to be general for normal users. For that, the open source libraries OpenMVG and OpenMVS are applied in

our pipeline and the whole project runs only with CPU.

Chapter 4

Implementation

This chapter shows how to implement the whole pipeline and some important steps will be discussed so that readers can follow the step without previous knowledges in every phase of the implementation.

4.1 Run the whole project using Docker

Since the pipeline involves various libraries and software modules, then comes the problem of coupling. Some of them are high coupling and the others might be independent from each other. Even more, the requirement of development environment of different libraries could also be distinct. As a consequence, the setup of the whole project can be complex and difficult. Hence, we have to find a way to build a integrated development environment and libraries inside it with low coupling, so that users can copy the whole project of pipeline and easily run it everywhere without concern about the dependencies. With this in mind, docker is a good solution for our purpose.

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable filesystem such as aufs and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines Wiki.
In other words, docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Docker is similar to virtual machine. Figure 4.1 shows the different between virtual machines and docker. Each virtualized application includes not only the application and the necessary libraries, but also an entire guest operating system, while Docker allows applications to use the same Linux kernel as the system that they are running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

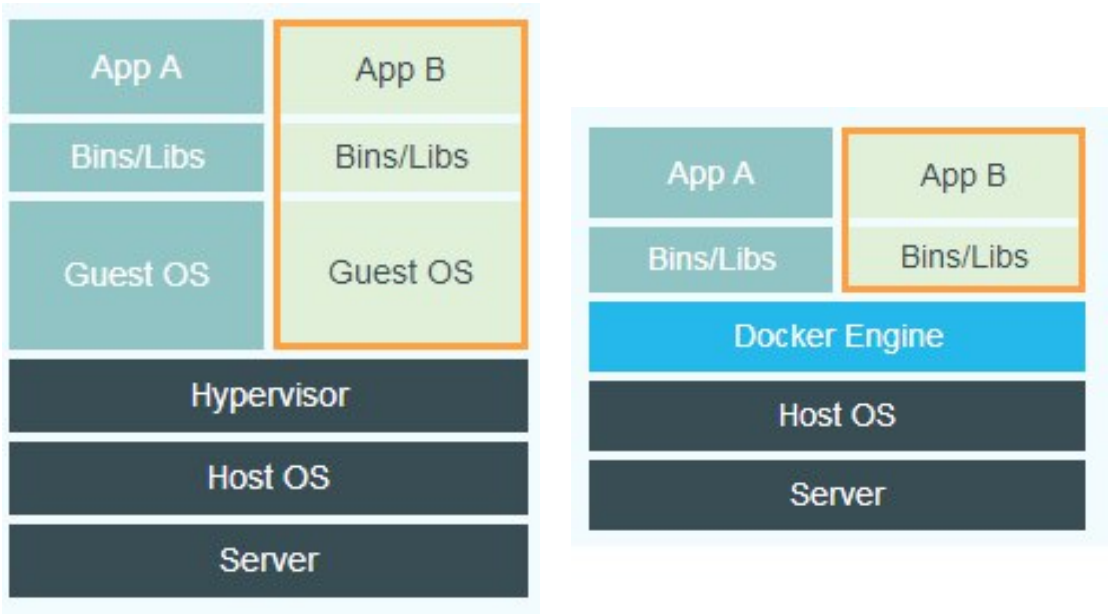


Fig. 4.1. (a) Virtual machines consisting of Applications, libraries and guest OS (b) Applications in Docker share the same linux kernel as the system that they are running on. source: Docker Documents

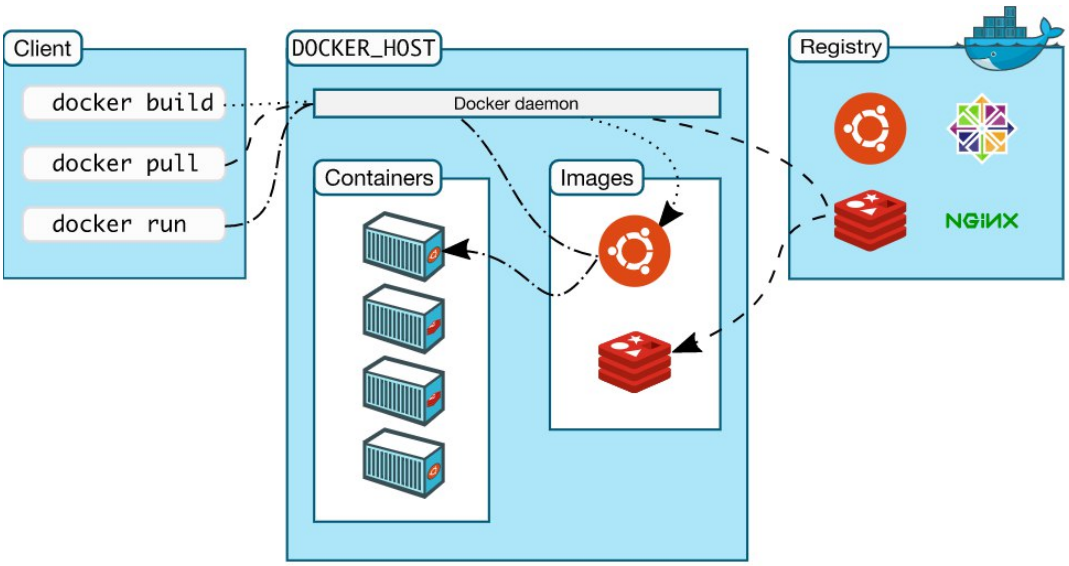


Fig. 4.2. Docker architecture. Client use the images by building, pulling and running. The container is considered as an instance of an docker image.

The architecture of Docker is shown in Figure 4.2. An image in Docker is a template with instructions for creating a Docker container. To build our own image, a Dockerfile with a simple syntax for defining the steps is needed. Each instruction in Dockerfile generates a layer in the image, so that if the Dockerfile was changed, only those layers which have changed are rebuilt. This is one of the reason why images is more lightweight, small, and fast compare to other virtualization technologies.
A container is considered as a runnable instance of an image. Creation, starting, stopping, moving, or deleting a container is available. We can even connect a container

to one or more networks and other containers. Hence, it is critical to decide which contents should be built in a same container and which not, especially in our case, this is directly related to the performance of the whole project/application.
After building an image wisely, upload the image to Docker Registry or a cloud such as azure, so that it can run in other device or server.

Installing Docker in associate platform. Since Docker support Mac OS, Windows, and the most distributions of Linux, it won't be difficult to install Docker (see the Documents of Docker). To check if Docker is successfully installed, tap **docker –version**

4.2 Corresponding libraries in the proposed pipeline

Because OpenMVG and OpenMVS both provide a standard Dockerfile, just copy it to local and run the docker command to build images for OpenMVG and OpenMVS in the folder where the Dokcerfile exists. Since the two libraries support the same version of Ubuntu and the result from OpenMVG can be easily exported to OpenMVS, we can simply merge two Dockerfile into one and build it in a single image. After building the image and running a container of this image, we can start the reconstruction using the methods provided by OpenMVG and OpenMVS.

Image matching/retrieval

Tasks	Description
Init the scene `openMVG_main_SfMInit_ImageListing`	Initialize a `sfm_data` scene with Views referencing the input images and reference to camera model(s). - Each View is referencing an intrinsic Id and a pose Id. *Note that this binary allows choosing with camera model your images are using.*
Describe the images `openMVG_main_ComputeFeatures`	Compute the requested features (`Regions`) for all Views of a `sfm_data` file.
Image Matching `openMVG_main_ComputeMatches`	Compute pairwise matches.

Structure-from-Motion specific methods

Tasks	Description
`openMVG_main_IncrementalSfM`	Start the reconstruction from 2-view and sequentially adding the remaining view.
`openMVG_main_IncrementalSfM2`	Start the reconstruction from 2 or n-view and sequentially adding the remaining view. This pipeline is faster than `openMVG_main_IncrementalSfM` but less accurate.
`openMVG_main_GlobalSfM`	Compute relative motion between pair or images and then use motion averaging to solve all the camera poses at the same time.
`openMVG_main_ComputeStructureFromKnownPoses`	Retrieve the structure of the scene (point cloud) where the camera locations are known by either triangulating the matches or looking for corresponding points along the epipolar lines.

Fig. 4.3. Various of instructions are used in different situations. source: OpenMVG

Figure 4.3 shows some general instruction of OpenMVG. Note that OpenMVG will try to retrieve the input image's pixel focal length. If an error with the log "No Intrinsic found" came, it is necessary to submit an approximate focal length by adding

a optional parameter -f X * 1.2, where $X = \max(image < width, height >)$. This formula works for most of the smart phone cameras.
After finishing SfM process, OpenMVG will generate a .bin file containing camera poses and the sparse point cloud. Figure 4.4 shows the sparse point cloud of a box of milk and the green points are the positions of camera.

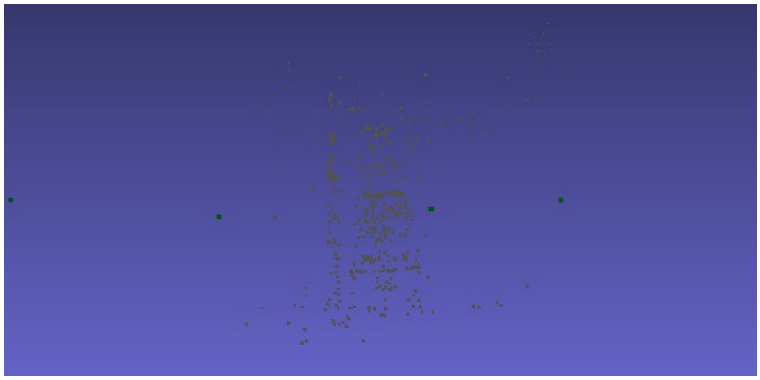
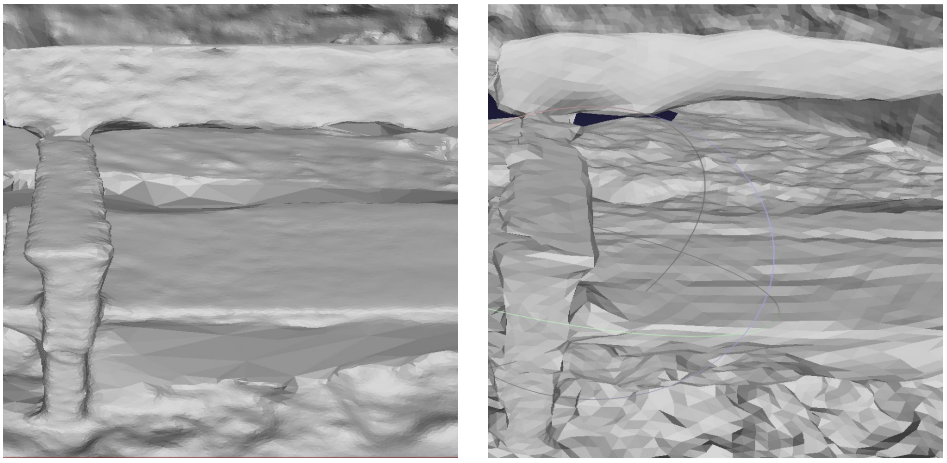


Fig. 4.4. Recovery of the camera positions and the sparse point cloud

Using the exporter tool by OpenMVG, it's easy to convert .bin file to the OpenMVS project as .mvs format. Then OpenMVS compute depth-map for every processed image and save as .dmap format and a corresponding dense point cloud. By adding an option -v 4, it generates additionally a .ply file for viewer. The dense point cloud is used as the input of the mesh reconstruction using Volumetric Graph-Cuts on Delaunay Tetrahedralization.



(a) (b)

Fig. 4.5. (a) Surface of a reconstructed object (bench) before mesh-refinement (b) Result of the left object after mesh-refinement.

Figure 4.5 shows a result of mesh reconstruction and mesh refinement of a bench. In each iteration of mesh refinement, the phase called adaptMesh removes the planar vertices that have small gradient, that is, their gradient-size and smooth of grad score are under a default threshold. Since the step of adaptMesh influences the quality of reconstructed surface with curvature, the threshold is not suggested to set to a high value. As a result, (b) after mesh refinement has fewer faces comparing with (a).

Since this method of mesh-refinement in the proposed pipeline works not good in this situation, it is not necessary for box shaped object, which has flat faces.
After remeshing, the last step is Texture Mapping. OpenMVS generates an associate atlas containing texture coordinates (see Figure 4.6).

Fig. 4.6. Texture Atlas of the reconstructed object (bench)

4.3 Postprocessing of 3D models

From the previous section, a reconstructed scene is obtained. Figure 4.7 is a example of a reconstructed scene. It can be applied as a virtual scene in virtual reality. However, in some cases only a specific object from the reconstructed scene is needed for further applications. Hence, the Postprocessing is necessary for further applications. Here a 3D model editor called Blender is suggested by the author to deal with this case. It is user friendly and contains various powerful functions of editing a 3D model.

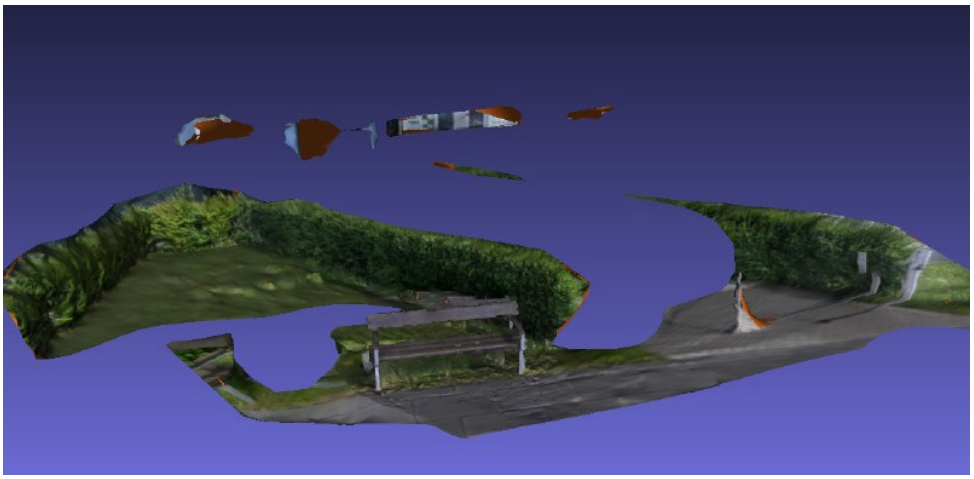


Fig. 4.7. An example of a reconstructed scene, which contains the target object (bench), unnecessary parts and groups of close vertices

After cutting off the unnecessary details using blender, then only a clean 3D model of a bench remains (see Figure 4.8). So it can be used in the mixed reality, which will be discussed in the next section.

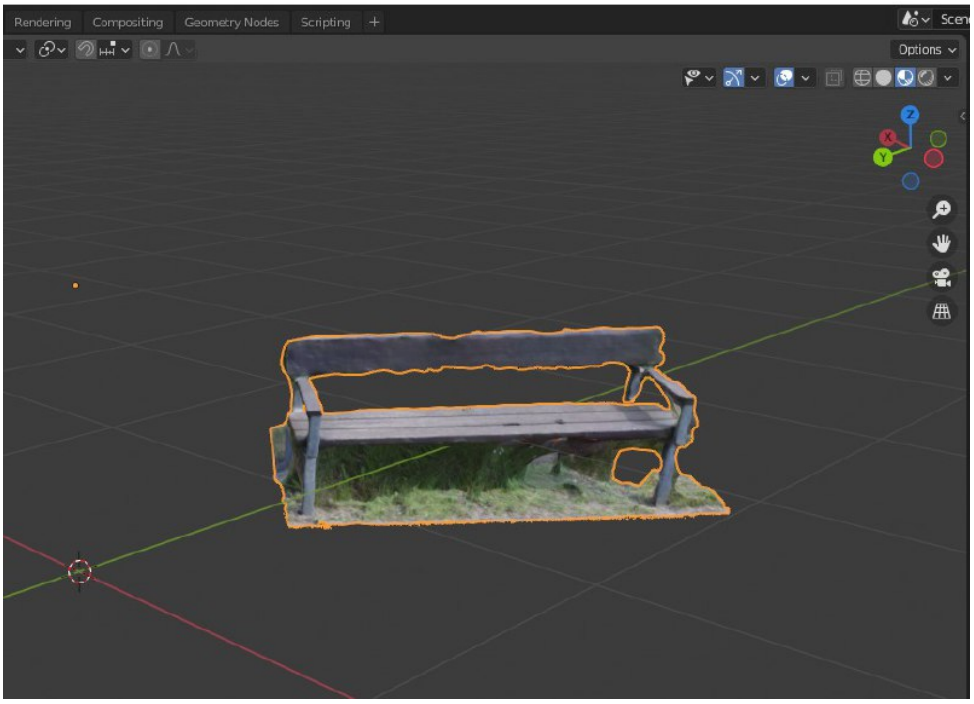


Fig. 4.8. Blender is a powerful tool for postprocessing of 3D model

In addition, Meshlab provide various powerful functions relating to repair and clean the mesh of the reconstructed 3D object. One of the mainly used functions for the proposed pipeline is merging close vertices, which merge together all the vertices that are nearer than the specified threshold, like a unify duplicated vertices but with some tolerance. As a result, it reduces the total number of vertices and the size of object-file. The fuctions of smoothing such as Laplace-smooth depends on the situation of further usage.

4.4 Application in Mixed Reality

After processing of the previous steps, a 3D model has been reconstructed. It can be used in further applications. Here is an example in Unity to creating an AR-App for Android device. Following are the general steps:

1. clone the repository of AR-Foundation in local
2. make sure all the requirements are satisfied, such as JDK
3. switch the platform to Android
4. choose your favorite scene for different situation
5. import 3D models into the scene
6. build project and save as .apk file

If users aim to import reconstructed object into advanced mixed reality scene like in Microsoft Hololens, it is suggested to use Mixed Reality Toolkit (MRTK). It has various functions and supports common mixed-reality devices.

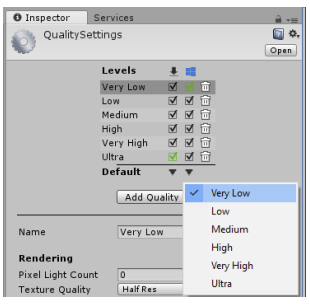
To get started with the Mixed Reality Toolkit, the following requirements must be satisfied:

- Visual Studio 2019 or higher
- Unity 2018.4.x, Unity 2019
  MRTK supports both IL2CPP and .NET scripting backends on Unity 2018
- Windows SDK 18362+
  This is necessary if building a UWP app for WMR, Hololens 1&2
- Hololens Emulator
  The Hololens Emulator lets user test holographic applications on PC without a physical HoloLens

Make sure to get the latest MRTK Unity packages and import MRTK packages into the corresponding Unity project.To create a HoloLens application, switch the Unity project to the Universal Windows Platform (UWP). Last, add MRTK to a new scene or new project.
The configurations of developing a HoloLens-App is shown as below:

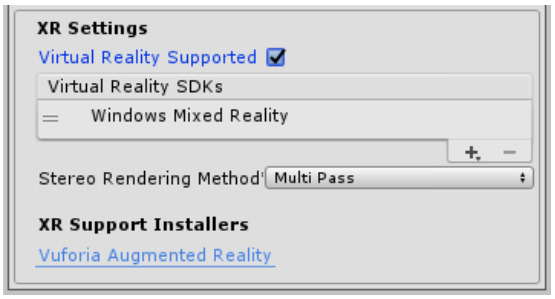
Quality settings for HoloLens

1. Select Edit > Project Settings > Quality
2. Select the dropdown under the Windows Store logo and select Very Low.

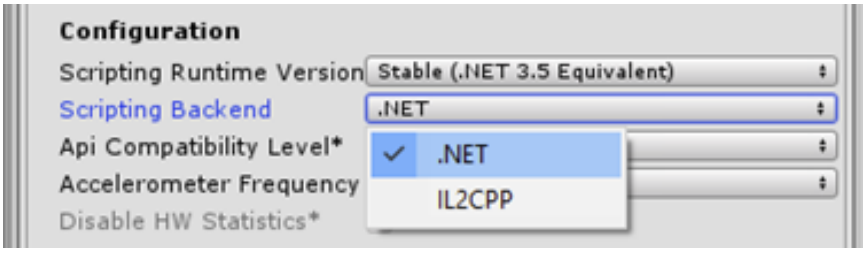


Target Windows 10 SDK

1. Edit > Project Settings > Player.
2. select the Windows Store icon.
3. Expand the XR Settings group.
4. In the Rendering section, check the Virtual Reality Supported checkbox to add a new Virtual Reality SDKs list.
5. Verify that Windows Mixed Reality appears in the list.



Verify .NET Configuration



make sure that Scripting Backend is set to .NET
Notice that in Unity 2019 the .NET backend was deprecated and only IL2CPP remains

4.5 Building pipeline on server

The whole project of the pipeline except the mixed reality part can run on a server, so that users can use the functions of reconstruction outside with a smart phone.

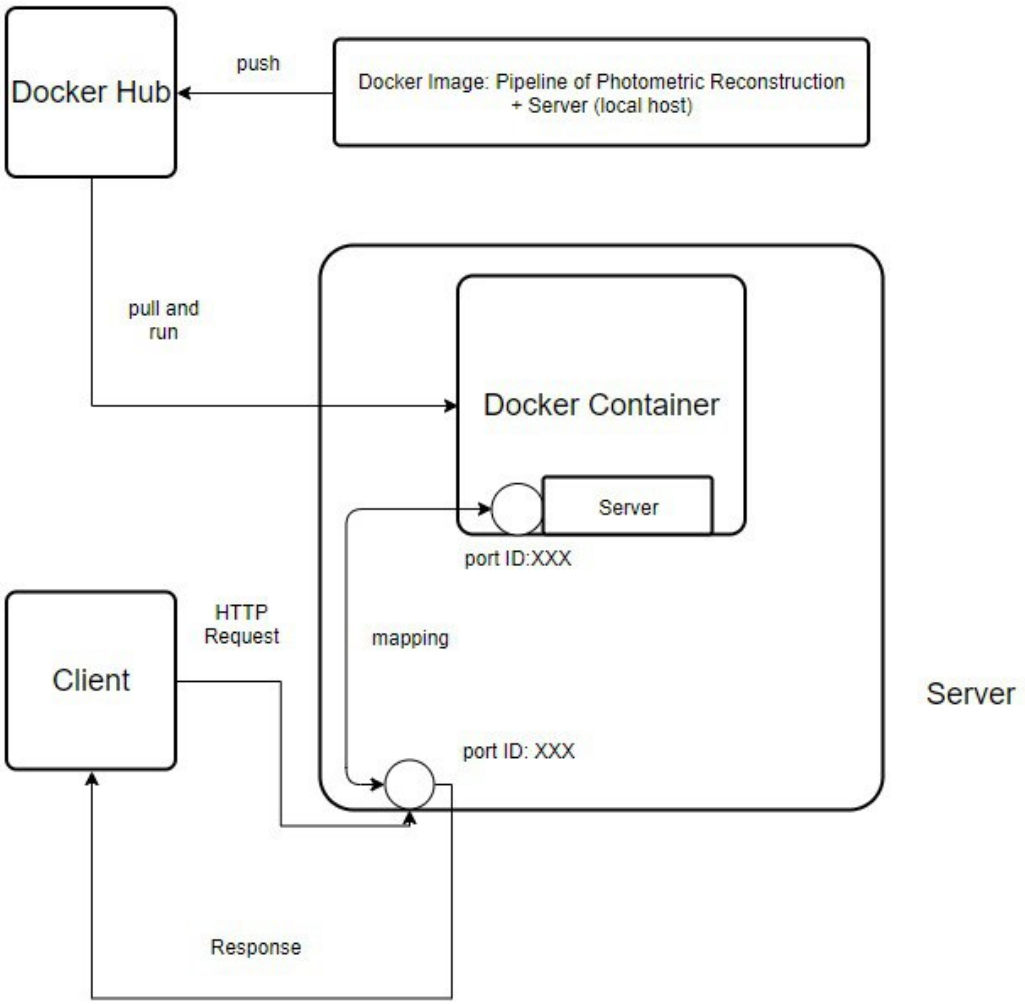


Fig. 4.9. Architecture of a server supporting docker and a simple web server starts, offering uploading of images and downloading of the returned result of reconstruction for clients

Figure 4.9 shows how a server supporting docker works. Since the proposed working pipeline can run in the docker container, it is easy to push the whole container as a docker image to Docker Hub, which stores various images such as OS based image, server based image, etc. When the server supports docker engine, it can pull the image from Docker Hub into local and run the image without any other dependencies. Then, inside the container, a simple web server starts, offering uploading of images and downloading of the returned result of reconstruction for clients. To be noticed that this web server should be bound in a certain port of local host, which is mapping to one port of server, so that visiting the port of the server is equivalent to visit the port inside container. Generally speaking, if port A inside container was mapping to port B of the server, then the IP adress is IP (server): ID (port B) visited by clients

```
PORTS
0.0.0.0:2020->2020/tcp, 5000/tcp
```

Fig. 4.10. Port 2020 of an image is mapped to the port 2020 of the server

As shown in Figure 4.10, the port 2020 of an image is mapped to the port 2020 of the server. since the server is logged as root, it is necessary to check the connection information of ports. Once the web service inside the container starts, it can be visited from outside. Visitors can click the button to send a request to the server and wait the response. Then it is possible to upload images, which shown in Figure 4.11

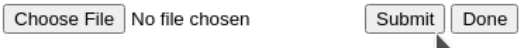


Fig. 4.11. Website for uploading images

After uploading all the required images, it starts the photogrammetric reconstruction in backend (see Figure 4.12).

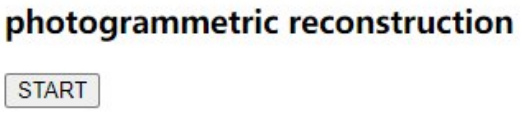


Fig. 4.12. Website for starting photogrammetric reconstruction

In the backend of the server, all the log of information about reconstruction-progress will be listed and seen by the administrators (see Figure 4.13)

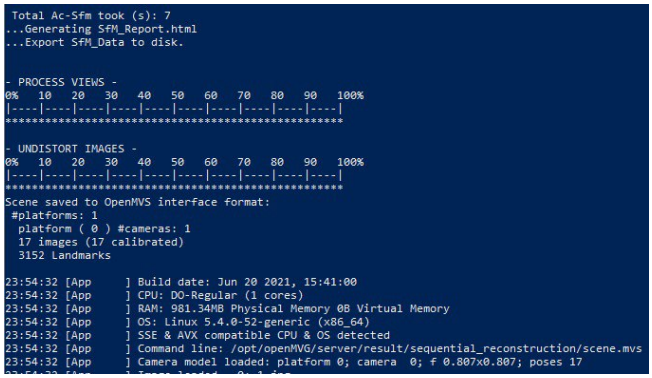


Fig. 4.13. Log information of reconstruction is listed and seen by the administrators of server.

Fig. 4.14. After computing, it turns to download website.

After a period of time in reconstruction (the evaluation of run time will be discussed in the next chapter), it turns to a download website, which offers download option for the visitors to download the file of results.
The 2 libraries, Flask and Werkzeug, are used for building server, that support a static web service. Flask is a micro web frame work written in Python. It is classified as a microframework because it does not require paticular tools or libraries [45]. It contains no database abstraction layer comparing the common web framework, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Hence, it is enough for the purpose of this thesis, that is, creating a simple but quick web service for the photogrammetric reconstruction. Since there are various web application applied in the web, it is suggested to use Werkzeug, which ensures that the web application can speak with the webserver and these web applications can work nicely together.

Chapter 5

Evaluation

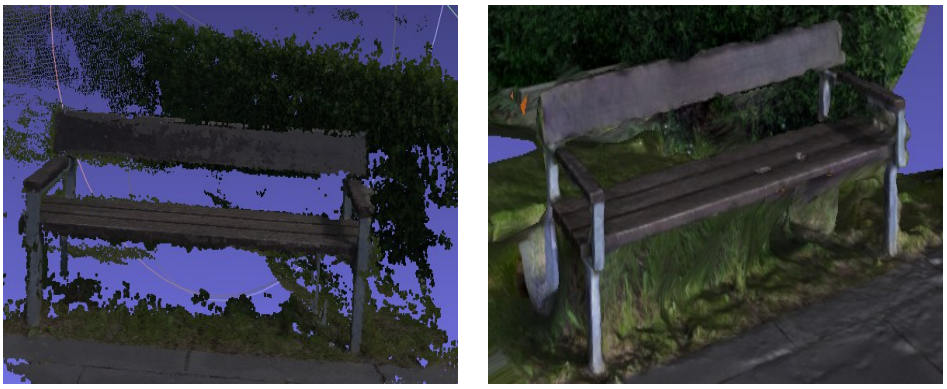
5.1 Test on a box shaped object

It is convenient to take photos and run experiments on various datasets to evaluate the performance of the proposed working-pipeline in different criterion such as performance in run time, number of image and the resolution of image. Following is an example of reconstructing a bench with 17 photos as input (1280 $\times$ 960). The reason why choose bench as the object to be evaluated is that bench is a common representative of boxed shaped object but not so simple as box. It consists of numbers of flat surface, which are a good samples to evaluate the quality of surface. Besides, the evaluation of affine ambiguity is also possible, as bench has many parallel lines.

(a) (b)

Fig. 5.1. (a) One of the view focus on bench (b) The corresponding sparse point cloud and the camera poses. Most of the point on edges remain

From figure 5.1 we can see, most of the points (over 1000) on edges remain. After testing on numbers of box-liking samples, the results shows that the proposed pipeline has a good performance on these box shaped objects in the step of sparse point cloud reconstruction



(a) (b)

Fig. 5.2. (a) Dense point cloud of the bench (b) Mesh of the bench with texture

As the angle between each 2 adjacent views is suggested to be 15°, Figure 5.2 (a) shows that the dense point cloud of bench with few outliars and a robust sequential SfM. Besides, in (b) we can see the high quality of 3D model of the bench. By using SGBM algorithm in the proposed working pipeline, even the grass is lifelike.
After cutting off the redundant parts using Blender, the 3D-model of bench is then imported into Unity for developing an AR application. Figure 5.3 shows a use case of the bench in augmented reality. The color of the object is approached to the original one (light brown wood and silver base). Since the project runs in the framework of AR-Foundation, it is possible to drag, scale and rotate the object. Furthermore, it can be used in other applications (see more in AR-Foundation).

Fig. 5.3. An use case of the reconstructed bench in an AR scene

Figure 5.4 shows the run time of SfM in each step and the step of feature matching is the most time consuming in SfM.

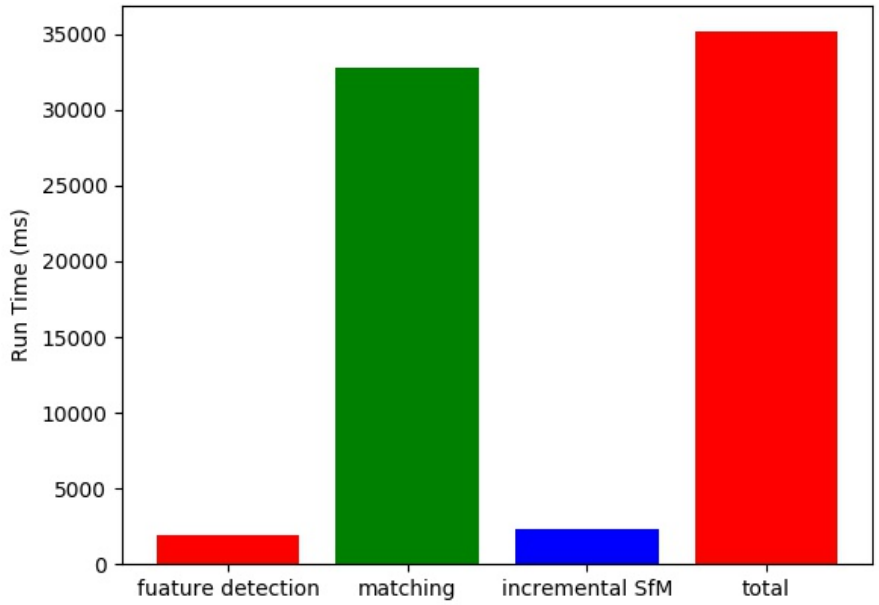
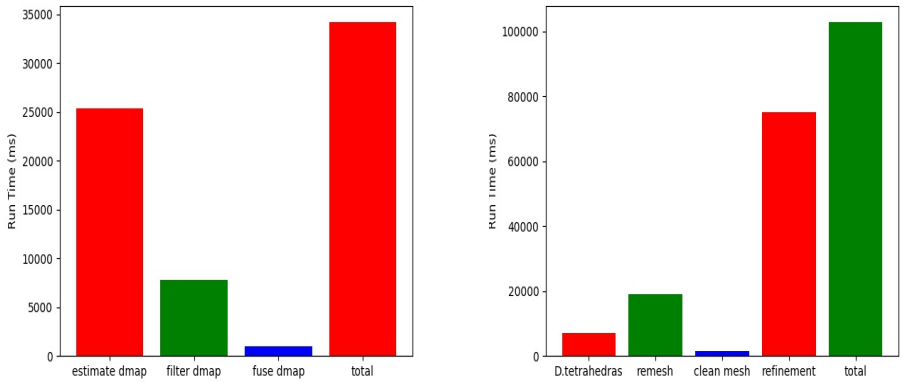


Fig. 5.4. Run time (ms) of SfM with 17 (1280 × 960) images of bench

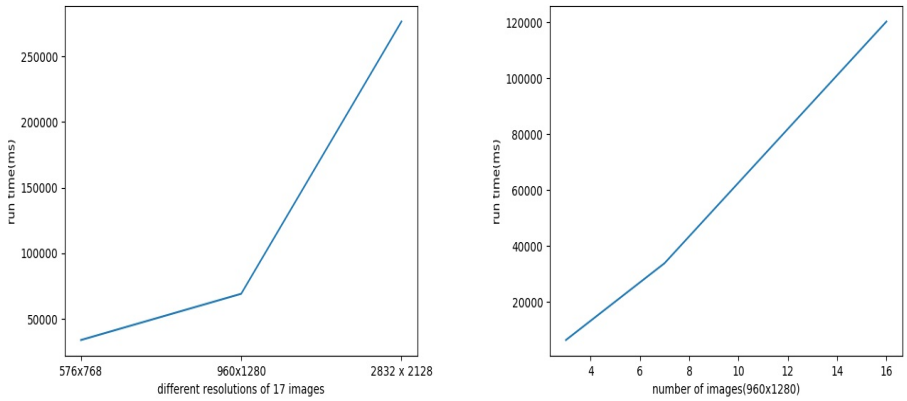
Figure 5.5 shows the run time of remeshing and refinement. The phase of estimation of depthmap takes much of the time, because the algorithm of semi-global-matching, discussed in Chapter 2, runs in pseudo-polynomial time. So the same is the phase of refinement. In general, most of the algorithm based on MRF-formulation are weakly NP-complete and run in pseudo-polynomial time.



(a) (b)

Fig. 5.5. (a) Run time (ms) of remeshing with 17 (1280 × 960) images of bench (b) Run time (ms) of refinement of mesh with 17 (1280 × 960) images of bench

Figure 5.6 (a) shows that the total run time increases rapidly with the growth of image resolution, and so is the number of images in Figure 5.6 (b)



(a) (b)

Fig. 5.6. (a) Run time (ms) in different resolutions with 17 images of bench (b) Run time (ms) of different number of image (1280 × 960) of bench

5.2 Test on a cylinder shaped object

From the last section, it has been shown that the proposed pipeline is robust to the box shaped object with flat surfaces. Now, it is of interest to test the performance of the pipeline on an object with curved surface. Here the author chooses a cylinder-like object with 34 images (960 × 1280) as samples. Since the AKAZE-algorithm is applied in the proposed pipeline for the object with curved surface, the point cloud of the pole of lamp in Figure 5.7 shows robustness to illumination. Besides, most of the details on surfaces remain and almost no obvious hole on surfaces exists.

(a) (b)

Fig. 5.7. (a) One of the 34 views on cylinder-like object (960 × 1280) (b) The corresponding dense point cloud contains massive points.

As 34 images has been used in this test, the high cost of run time is inevitable. Figure 5.8 shows that the depth-map estimation costs nearly 5 minutes, despite that

this method has been reduced to a linear time (dynamic programming, see Chapter 2). The situation is the same in the phase of mesh-refinement (see Figure 5.9), it costs more than 3 minutes. As the proposed suggestions above, the angle between each 2 adjacent views can be selected from 15° to 30°, so theoretically 12-24 images are enough to save information as much as possible.

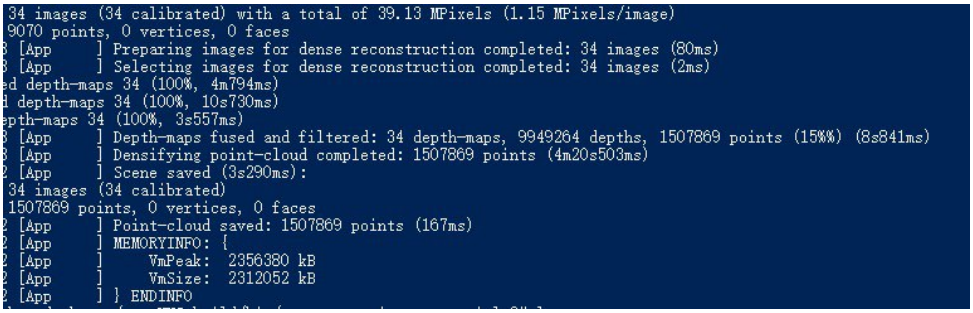


Fig. 5.8. Log information of the depth-map phase and dense point cloud, where the step of depth-map-estimation costs nearly 5 minutes.

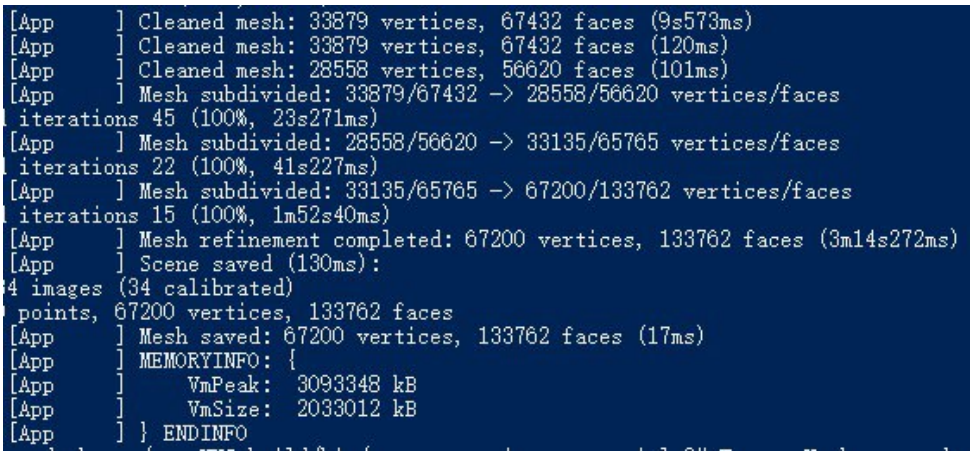


Fig. 5.9. Log information of mesh-refinement shows that the phase of mesh-dividing is time consuming.

Figure 5.10 shows the details on the surface of the reconstructed cylinder-like object. The surface is smooth and well curved, which meets the author's expectations.

Fig. 5.10. The proposed pipeline shows a good performance on rebuilding the surface of the cylinder-like object with high quality.

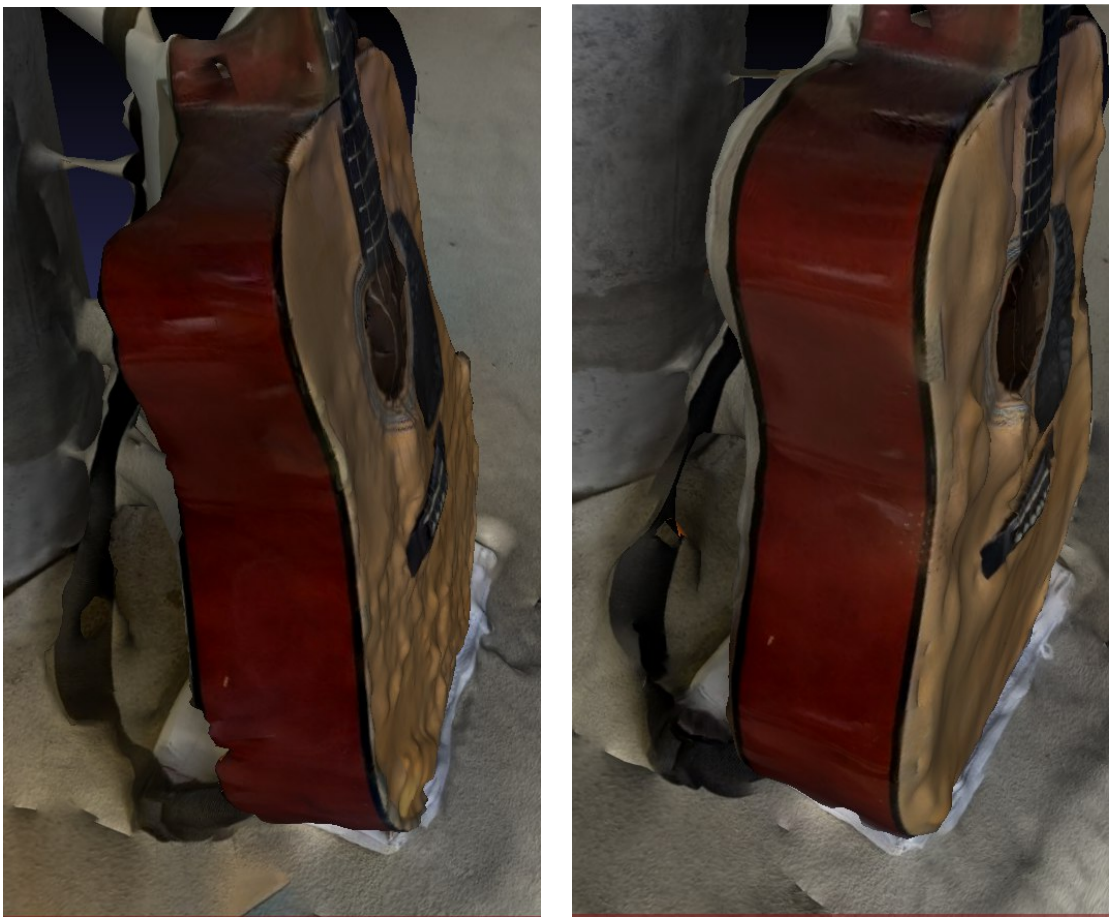
5.3 Test on a complex object

As the author's expectation, the proposed pipeline works not only on the simple object above, but also has a good performance on the complex object. Hence, a guitar here is chosen as a complex object for the evaluation of the pipeline. It contains various geometries, such as straight lines and curved surface on both sides. Moreover, it is worth noting that the front surface consists of the exact same texture, which is a good sample to test the robustness of feature-matching. Since it is a complex object, it is critical to choose the most suitable feature-algorithm. Figure 5.11 (18 images with resolution of 1280×960) shows a example of reconstruction using AKAZE algorithm due to the nice performance on curved surface with a high albedo.

(a) (b)

Fig. 5.11. (a) One of the 18 views (b) The reconstruction result shows that AKAZE has a good performance on these kind of objects

Figure 5.12 shows the reconstructed guitar with and without the phase of mesh-refinement. As a result, it is suggested to use mesh-refinement for these objects containing curved surface. However, as the expense of using mesh-refinement, the front surface undulates largely than the left, which should be expected as flat as possible.



(a) (b)

Fig. 5.12. (a) Final result without refinement, which shows that the surface on both sides is uneven (b) After refinement of the mesh, the surface is more smooth and well curved comparing with (a)

Chapter 6

Summary

This thesis aimed to build a working-pipeline of photogrammetric reconstruction focusing on geometrically simple objects and analyze the performance of the pipeline on different kind of objects. Chapter 2 involves general theories of photogrammetric reconstruction and the preliminary works, which can help the readers better understand the area of reconstruction. Chpater 3 is the refinement of the general pipeline, so that it is more suitable for the geometrically simple objects. Chapter 4 is the implementation of the proposed pipeline, which makes a easy way to build and run. As the evaluation in Chapter 5, the results almost meets the author's expectations. However, it is not perfect and universal for all cases. For example, for the objects that contains both flat and curved surfaces, the quality are not all good on both of them at the same time. As someone said, "*You can't have your cake, and eat it*". In the near future, the author will apply his energy to solve this problem and make it more robust to the complex objects. Furthermore, if a scan-device with high accuracy and some libraries supporting GPU are applied in this proposed pipeline instead of using smart phone and non-GPU libraries, this pipeline could be further applied in industry.

Bibliography

[1] Oskar Andersson and Steffany Marquez. "A comparison of object detection algorithms using unmanipulated testing images". In: *KTH, Stockholm, Sverige* (2016).

[2] Ronald T. Azuma. "A survey of agmented reality". In: *Teleoperators and Virtual Environments* 6 (1997), pp. 355–385.

[3] R. I. Hartley B. Triggs P. F. McLauchlan and A. Fitzgibbon. "Bundle adjustment a modern synthesis". In: (2000).

[4] C. Beder and R. Steffen. "Determining an initial image pair for fixing the scale of a 3d reconstruction from an image sequence". In: *Pattern Recognition* (2006).

[5] J. L. Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM. 18 (9)* (1975), pp. 509–517.

[6] Mario Botsch and Olga Sorkine. "On linear variational surface deformation methods". In: *IEEE Transactions on Visualization and Computer* ().

[7] Reif Brassel. "A Procedure to Generate Thiessen Polygons". In: *Georaphical Analysis* (1979), pp. 289–303.

[8] Cum and Matas. "PROgressive SAmple Consensus". In: *Machine Vision and Applications* (2005).

[9] Brian Curless and Marc Levoy. "A volumetric method for building com plex models from range images". In: *In ACM SIGGRAPH* (1996).

[10] N. Snavely D. Crandall A. Owens and D. P. Huttenlocher. "Discrete-Continuous Optimization for Large-Scale Structure from Motion". In: *CVPR* (2011).

[11] Jesús A. De Loera. "Triangulations, Structures for Algorithms and Applications". In: *Algorithms and Computation in Mathematics. 25. Springer* (2010).

[12] Boris Delaunay. "Sur la sphère vide". In: *Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelle* (1934).

[13] Fischler and Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Comm. of the ACM. 24 (6)* (1981).

[14] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *ACM* (1981).

[15] Iryna Gordon and David G.lowe. "What and where: 3D object recognition with accurate pose". In: *Springer-Verlag* (2006), pp. 67–82.

[16] R. Keriven H-H. Vu P. Labatut and J.-P Pons. "High accuracy and visibility-consistent dense multi-view stereo". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2012), pp. 889–901.

[17] M. A. Hartigan J. A.; Wong. "Algorithm AS 136: A k-Means Clustering Algorithm JSTOR 2346830". In: *Journal of the Royal Statistical Society, Series C* (), pp. 100–108.

[18] M. Havlena and K. Schindler. "Vocmatch: Efficient multiview correspondence for structure from motion". In: *ECCV* (2014).

[19] E. Dunn J. Heinly J. L. Schonberger and J.-M. Frahm. "Reconstructing the World in Six Days(As Captured by the Yahoo 100 Million Image Dataset)". In: *CVPR* (2015).

[20] A. C. Berg J. L. Schonberger and J.-M. Frahm. "PAIGE: PAirwise Image Geometry Encoding for Improved Efficiency in Structure-from-Motion". In: *CVPR* (2015).

[21] D. Gallup T. Johnson R. Raguram C. Wu Y.-H. Jen E. Dunn B. Clipp S. Lazebnik J.-M. Frahm P. Fite-Georgel and M. Pollefeys. "Building Rome on a Cloudless Day". In: *ECCV* (2010).

[22] Jan-Michael Frahm1 Johannes L. Schönberger. "Structure-from-Motion Revisited". In: (2016).

[23] V. Kolmogorov and R. Zabih. "Computing visual correspondence with occlusions using graph cuts". In: *In IEEE International Conference on Com puter Vision* 2 (2001), pp. 508–515.

[24] Christophe Dumont Laurana M. Wong and Mongi A. Abidi. "Next-best-view algorithm for object reconstruction". In: *Proc. SPIE 3523, Sensor Fusion and Decentralized Control in Robotic Systems* (1998).

[25] David G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* (2004).

[26] Z. Kukelova M. Bujnak and T. Pajdla. "A general solution to the P4P problem for camera with unknown focal length". In: *CVPR* (2008).

[27] J. MacQueen. "Some Methods for classification and Analysis of Multivariate Observations". In: *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability* (2009).

[28] Tommi S. Jaakkola Martin J. Wainwright and Alan S. Willsky. "Map estimation via agreement on trees: message-passing and linear programming". In: *IEEE Transactions on Information Theory,51* (2005).

[29] René Ranftl Zhuwen Li Vladlen Koltun Thomas Brox Maxim Tatarchenko Stephan R. Richter. "What Do Single-view 3D Reconstruction Networks Learn?" In: *arxiv* (2019).

[30] H.Imai M.Inaba N.Katoh. "Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering". In: *Proceedings of the tenth annual symposium on Computational geometry* (1994).

[31] Marius Muja and David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". In: *VISAPP(1)* (2009).

[32] S. Seitz N. Snavely and R. Szeliski. "Photo tourism: exploring photo collections in 3d". In: *ACM TOG* (2006).

[33] S. Seitz N. Snavely and R. Szeliski. "Photo tourism: exploring photo collections in 3D". In: *ACM TOG* (2006).

[34] David Nister. "An Efficient Solution to the Five-Point Relative Pose Problem". In: *CN5300, Princeton, NJ08530* (2004).

[35] David Nister. "Preemptive RANSAC for live structure and motion estimation". In: *Machine Vision and Applications* 16 (), pp. 321–329.

[36] David Nister and Henrik Stewenius. “Scalable Recognition with a Vocabulary Tree”. In: *International journal of computer vision* (2006).

[37] Jean-Philippe Pons Patrick Labatut and Renaud Keriven. “Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts”. In: *IEEE International Conference on Computer Vision* (2007).

[38] Simon Prince. “Computer vision: models, learning and inference”. In: *Cambridge University Press* (2012).

[39] M. Farenzena R. Gherardi and A. Fusiello. “Improving the efficiency of hierarchical structure-and-motion”. In: *CVPR* (2010).

[40] Andrew Zisserman Richard Hartley. “Multiple View Geometry in Computer Vision”. In: *Cambridge University Press* (2004), pp. 243–245.

[41] Andrew Zisserman Richard Hartley. “Multiple View Geometry in Computer Vision”. In: *Cambridge University Press* (2004), pp. 246–247.

[42] Andrew Zisserman Richard Hartley. “Multiple View Geometry in Computer Vision”. In: *Cambridge University Press* (2004), pp. 257–258.

[43] Andrew Zisserman Richard Hartley. “Multiple View Geometry in Computer Vision”. In: *Cambridge University Press* (2004), pp. 321–313.

[44] Andrew Zisserman Richard Hartley. “Multiple View Geometry in Computer Vision”. In: *Cambridge University Press* (2004), pp. 315–316.

[45] Armin Ronacher. “Flask Foreword”. In: *Archived from the original* (2017).

[46] Richard J. Rossi. “Mathematical Statistics : An Introduction to Likelihood Based Inference”. In: *New York: John Wiley & Sons. p. 227. ISBN 978-1-118-77104-4* (2018).

[47] N. Snavely I. Simon B. Curless S. Seitz S. Agarwal Y. Furukawa and R. Szeliski. “Building rome in a day”. In: *ICCV* (2009).

[48] F. Moreno-Noguer V. Lepetit and P. Fua. “EPnP: An accurate O(n) solution to the PnP problem”. In: *IJCV* (2009).

[49] C. Wu. “Towards linear-time incremental structure from motion”. In: *3DV* (2013).

[50] N. Snavely Y. Lou and J. Gehrke. “MatchMiner: Efficient Spanning Structure Mining in Large Image Collections”. In: *ECCV* (2012).