Improving the Efficiency of SfM and its Applications

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (By Research)
in
Computer Science and Engineering

by

Siddharth Choudhary
200601088
siddharth.choudhary@research.iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2012

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Improving the Efficiency of SfM and its Applications" by Siddharth Choudhary, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. P J Narayanan

To my family and friends

Acknowledgments

This dissertation would not have been possible without the help, continuous encouragement and support from many people. I would like to thank Dr. P. J. Narayanan for his support and guidance during the past few years. I would like to acknowledge Noah Snavely for the Datasets and Torsten Sattler for insightful discussions over the email.

I am also grateful to my fellow lab mates at the CVIT, IIIT Hyderabad for their stimulating company during the last couple of years. Above all I am thankful to my family and friends for their unconditional support and unbounded patience

Abstract

Large scale reconstructions of camera matrices and point clouds have been created using structure from motion from community photo collections. Such a dataset is rich in information; we can interpret it as a sampling of the geometry and appearance of the underlying space. In this dissertation, we encode the visibility information between and among points and cameras as *visibility probabilities*. The conditional visibility probability of a set of points on a point (or a set of cameras on a camera) can be used to select points (or cameras) based on their dependence or independence. We use it to efficiently solve the problems of image localization and feature triangulation. We show how the conditional probability can be combined with other measures to prioritize a set of points (or cameras) for matching and use it for fast guided search of points for the image localization problem. We define the problem of feature triangulation as the estimation of 3D coordinate of a given 2D feature using the SfM data. Our approach can guide the search to quickly identify a subset of cameras in which the feature is visible.

Other than image localization and feature triangulation, bundle adjustment is a key component of the reconstruction pipeline and often its slowest and the most computational resource intensive. It hasn't been parallelized effectively so far. We also a present a hybrid implementation of sparse bundle adjustment on the GPU using CUDA, with the CPU working in parallel. The algorithm is decomposed into smaller steps, each of which is scheduled on the GPU or the CPU. We develop efficient kernels for the steps and make use of existing libraries for several steps. Our implementation outperforms the CPU implementation significantly, achieving a speedup of 30-40 times over the standard CPU implementation for datasets with upto 500 images on an Nvidia Tesla C2050 GPU.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advent of internet based photo sharing sites like Flickr and Picasa has lead to the creation of internet scale photo collections commonly known as *community photo collection*. More than 80 million photos are uploaded to the web everyday [19]. A Flickr search on "St. Peters' Basilica" results in thousands of images from many view points. Figure 1.1 shows a screenshot of a result page. Community photo collection (CPC) is a rich source of visual information about the world. It contains large numbers of photographs of popular locations which are interesting from a tourist's point of view. It implicitly models a region as a collection of interesting landmarks. There is high density of images in and around popular locations like Taj Mahal or Eiffel Tower and low density of images in less popular locations like a street in Delhi. Each image may contain additional information in the form of metadata like date/time of the capture, textual tags describing the image location, the object being photographed. In some of the cases, GPS information is also available (typically 10-15% of the images [19]). Images are also captured under different weather and illumination conditions using wide variety of cameras. Snavely [84] termed this dataset as the *Distributed Camera*, because millions of images are being uploaded at each moment capturing most of the places during any time of the day and it acts as a visual record of our world [60]. Such large collection of varied images captured all over the world has lead to the development of many data driven algorithms. CPCs have created new opportunities and challenges to the computer vision community. It has created new applications in the area of crowd sourced modelling [4, 19], augmented reality [6] and photo tourism [77]. Researchers are able to exploit the internet photo collection for estimating GPS coordinates of a location given its image [31] and to in-paint a scene using millions of photographs [30].

3D reconstruction using structure from motion (SfM) is one of the key challenges in computer vision. SfM is used to create 3D models using images of it. Rich community photo collections have created new opportunities in the area of large scale reconstruction and crowd sourced modelling and its applications in 3D visualization and localization. Photosynth is one such rich application which uses the 3D output reconstructed using CPCs to provide virtual and immersive photo-tour of a location [77]. Recently there has been impressive progress in the area of large scale reconstruction using community photo collections [4, 19]. It has lead to the creation of city scale reconstructions of places like Rome, Dubrovnik and

Vienna. The reconstructed output consists of many cameras and points. For example, the reconstructed output of Trevi Fountain in Rome contains over 2000 cameras and 656000 points. The 3D coordinates of reconstructed points as well as translation and rotation parameters of reconstructed cameras are known in a common coordinate frame. The set of reconstructed points is generally called as a *point cloud*. The point cloud representation can be seen as a compact summary of the scene consisting of addtional geometric and visibility information. For each point, we know the cameras it is seen in and for each camera, all the points that it sees.

We use the following terms to describe the entities involved unambiguously. An *image* is a photograph of the space which contains several *features*. A feature is a 2D interest point with an associated descriptor. The SIFT descriptor is commonly used by the SfM community [46]. The image gets upgraded to a *camera* when its projection matrix is known. Camera is an image whose projection matrix (intrinsic and extrinsic parameters) is known. Similarly, a feature gets upgraded to a *point* when its 3D coordinates are known. (Li et al. introduced the terms feature and point with this meaning [42].) We use the term point to refer to a 3D point and feature-point to refer to a 2D feature. An SfM dataset thus contains a number of cameras, points, and their mappings. Additional images and features of the same space are easily available from the internet and other sources. Image localization problem aims to estimate the *camera* from an *image*. Similarly, the feature triangulation problem aims to estimate the *point* from a *feature*.

In this thesis, we improve several aspects related to the process of structure from motion from community photo collection. We exploit the geometric and visibility information available in SfM reconstructions to solve the problem of image localization and feature triangulation. We also propose a hybrid implementation using CPU and GPU to solve the problem of bundle adjustment which acts as a major bottleneck in the large scale reconstruction process. In the following section, we discuss each of the major aspects involved in detail.

1.1 Structure from Motion

In computer vision, Structure from Motion (SfM) refers to the process of computing the three-dimensional structure using the motion information available in images or videos captured from different view points [86]. Traditionally, 3D points are triangulated given the camera poses of the input images. Conversly, if we have known 3D point coordinates, the camera pose of the given image can be estimated using pose estimation. However, in structure from motion neither camera pose nor point location is available. This is an example of a circularly related problem. SfM estimates the camera poses and the scene structure simultaneously without requiring either to be known a priori, from the information about common points visible in the images.

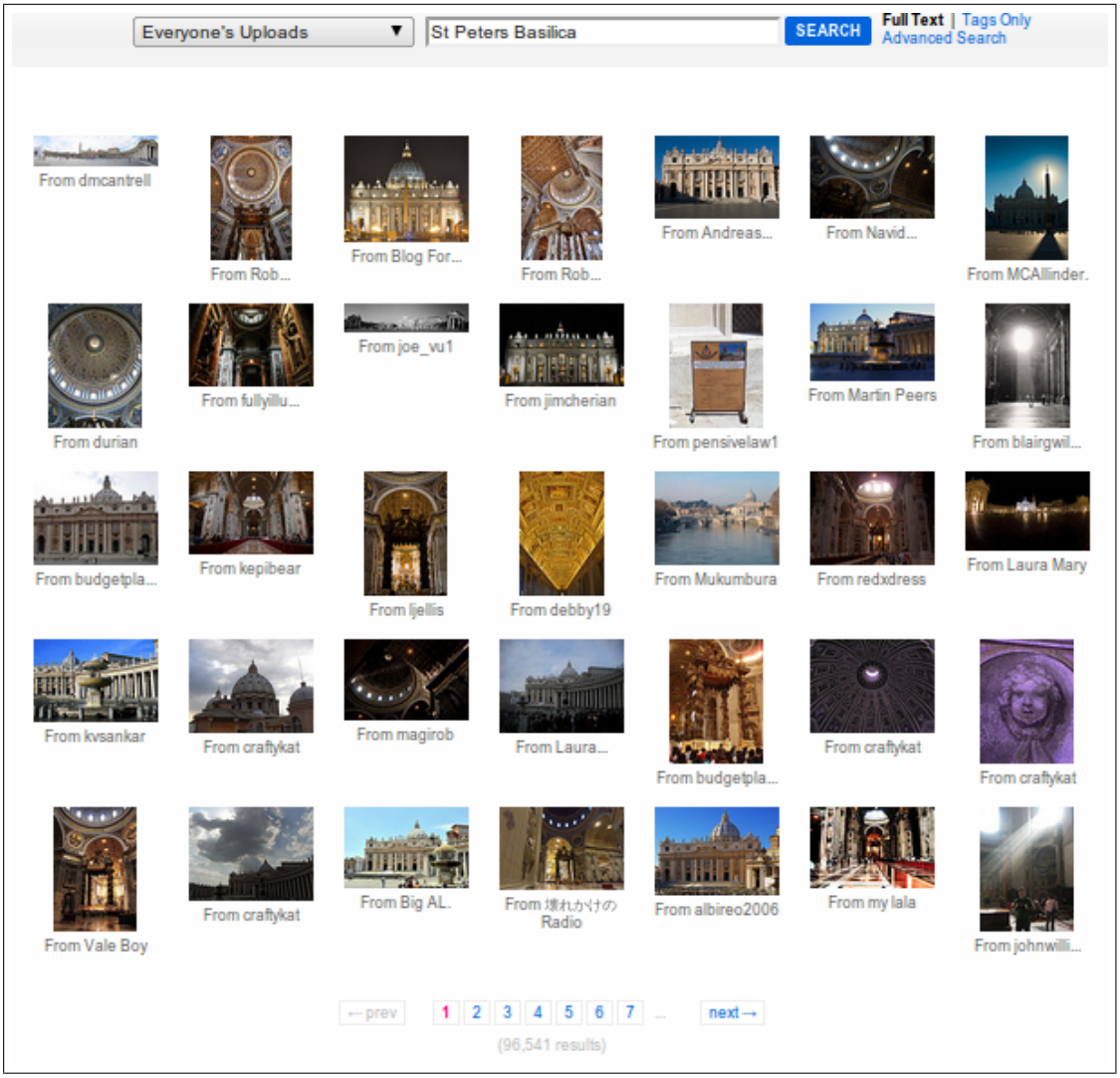


Figure 1.1: Result page for a Flickr search of "St. Peters Basilica"

1.1.1 Two View Structure from Motion

Two view SfM is closely related to the depth perception in the binocular human vision. Using images formed by an object in two eyes, we are able to roughly triangulate its distance in the real world. To do this, our brain implicitly computes the correspondences between the two images, so that we know which point in the two images correspond to the same 3D location. Our brain uses the distance between the two eyes (similar to relative camera pose) to triangulate the 3D location of the object.

The same technique can be applied to solve the problem of two-view SfM. Given two images of the same location, we find matching pixels in the two images and use those matches to estimate the pose of one image relative to the other. Using the relative pose, we estimate the 3D location of a point corresponding to each pair of matching pixels. To do so, we shoot a 3D ray from each camera location

through their respective matched pixels and intersect the rays to get a 3D point location. Figure 1.2 depicts the triangulation of a point using two views. The basic algorithm to estimate 3D point for two-views consists of these three steps,

- Estimate the relative camera pose between the two views (pose estimation).
- Identify common points in the two images.
- Project rays from the common points. Its intersection is the desired point (triangulation).

The SfM process can be divided into two basic categories of calibrated and un-calibrated condition.

1.1.1.1 Calibrated Case

Conventionally, 3D world is reconstructed using cameras calibrated through an offline process. Calibration is performed by observing a pattern of points whose position in 3D world is known and it can easily be used to estimate 3D-2D correspondences. Using the correspondences, we can estimate the camera pose with respect to the calibration pattern [88]. For the case of binocular stereo, realitve camera pose between the two views is estimated using the calibration pattern. Given the relative camera pose, epipolar constraint is used to estimate 2D-2D correspondences in the two images and the points are estimated using triangulation [29]. Two-view SfM in the case of calibrated cameras can be extended to multiple views. It is known as multi-view stereo. Multi-view stereo is used to reconstruct 3D model from images taken from previously known view points under calibrated conditions.

1.1.1.2 Un-calibrated Case

Structure from motion using uncalibrated views is relatively a tougher problem than SfM using calibrated views. Recently there has been a growing interest in calibrating cameras using the information available in the images itself without using any external calibration pattern. In the case of two-view SfM, pose of an image relative to the other image is estimated by finding 2D-2D correspondences between the two images. Given the correspondences, five-point algorithm is used to figure out the relative camera pose of two views [40, 55]. The 2D-2D correspondences are triangulated to estimate 3D points. Similar technique can be extended to multiple uncalibrated images which is commonly available in community photo collections. Large scale reconstruction using un-calibrated cameras is a challenging problem.

In the below section, we discuss the large scale reconstruction pipeline used by Photosynth [77] and many modern SfM methods to create point cloud models.

1.1.2 The SfM Reconstruction Pipeline

Large scale sparse 3D reconstruction from community photo collections using the structure from motion pipeline is an active research area today. The process poses many algorithmic challenges due to

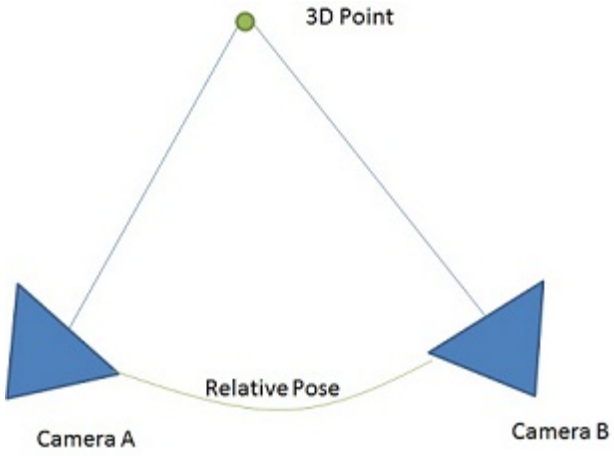


Figure 1.2: Two View Structure from Motion

large variations in the weather, illumination conditions, camera type and the lack of geographic information. Reconstructing multiple images using structure from motion can be divided into two distinct stages: (a) feature matching or correspondence estimation and (b) incremental structure from motion. We now explain each one of them in brief:

- *Image Matching*: Local features have to be computed in each image which are matched between every two image. In order to take into account the large variations, SIFT keypoint detector and descriptor [46] is used to find features in each image. SIFT descriptors are invariant to scale changes and are quite robust to other variations. Therefore, it is well suited to find correspondence across images. SIFT ratio test based matching is used to find correspondence between two images [46]. The pair-wise image matches are geometrically verified using epipolar constraints in RANSAC loop [18]. It removes the outliers which are not geometrically consistent and gives a realtive pose within all pair-wise configurations. Once pairwise matching and geometric verification is done, all the matching features corresponding to the same 3D point are organized into a track.

- *Incremental structure from motion*: Given the tracks of matching features, we estimate the 3D point and camera pose using incremental SfM. Initially, we find the best matching pair of cameras which are optimally seperated and reconstruct it using the two-frame reconstruction method. Rest of the images are added incrementally and reconstructed. Figure 1.3 visualizes the result of reconstruction after every set of images are added. After every increment, the points and cameras are optimized using a non-linear optimization called bundle adjustment which minimizes the sum of reprojection error across all registered images. We discuss bundle adjustment in detail in the later part of this chapter. Figure 1.4 shows the flowchart of the reconstruction pipeline as explained.

One of the primary issues with large scale structure from motion problem is its sclability to large image collections. Correspondence estimation and global optimization using bundle adjutment are one of the major computational bottleneck in the whole process. In the recent years there has been tremen-

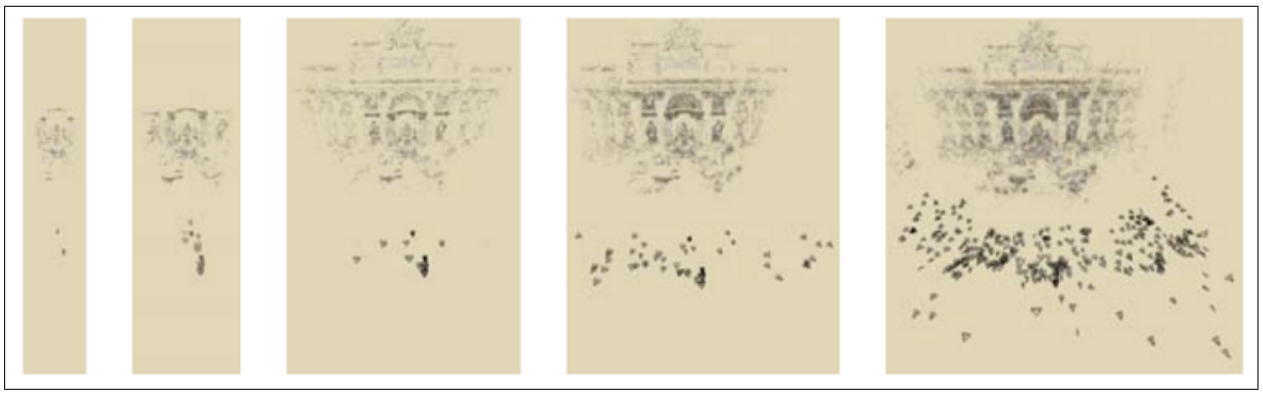


Figure 1.3: From Left to Right: Images of Trevi Fountain being reconstructed using incremental structure from motion(from Snavely[75])

dous improvement on this front. Starting from registering 3000 photos on 24 PCs using two weeks of computation as a part of Photo Tourism project, researchers are now able to reconstruct 3 million images on a single PC within 24 hours.

Structure from motion from community collections has created sparse point clouds and images of many large monuments and other spaces. A typical SfM dataset contains many points and several camera images. A few millions of points and a few thousand images are common today. The visibility of each point in its track of images is recorded, along with the SIFT descriptor in each. The camera matrices and point coordinates are known in a common frame. This represents a rich sampling of the geometry and appearance of the space. In this thesis, we explore the probabilistic structure of visibility induced by the points and images of an SfM dataset. We define the *visibility probability* of points and images and the *joint visibility probability* among points and among cameras, using which we estimate the conditional visibility probability of a set of points/cameras on other points/cameras. We define the probabilities in detail in Chapter 3. We use visibility probabilities and joint visiblity probabilities to efficiently solve the problem of localization and triangulation.

An SfM dataset contains a number of cameras, points, and their mappings. Additional images and features of the same space are easily available from the internet and other sources. Image localization aims to estimate the camera from an image. Similarly, the feature triangulation problem aims to estimate the point from a feature. We now describe each one of these in detail.

1.2 Image Localization

Given a query image, image localization is used to answer question “Where am I?”. The answer to this question can be provided either at a semantic level describing the location where the image is captured or in a geometric sense where the image location is estimated relative to other images or the model in a common frame. It is an important problem in the computer vision and robotics communities. Image Localization is defined as estimating the camera pose or parameters given several matching points

with respect to the world. The points being matched either belong to already localized 2D image or to 3D points belonging to a point cloud model. It is also known as camera resectioning. Given an image of a 3D scene, image localization is used to estimate the camera's position and orientation relative to the 3D scene. In this thesis, we specifically tackle the problem of localizing an image with respect to the SfM model. Localizing a new image to an existing SfM data set has attracted some recent attention [21, 32, 42, 64]. Image localization has numerous applications in the area of computer and robotic vision. It can be used during initial localization step in the process of incremental structure from motion. Instead of matching an image relative to other images, the new query image can be matched relative to the reconstructed model. Image localization can be done using these two major steps:

- Feature Matching: This is an important step of the localization pipeline. The 2D features in an image are matched to 3D points in a model to estimate the *absolute* orientation with respect to the model. In the reconstruction pipeline, pair wise image matches are done to estimate the *relative* pose between images. Estimating pair-wise matches is one of the major computational bottleneck in the process of large scale structure from motion. 2D points in the case of relative pose and 3D points in the case of absolute pose act as control points using which we estimate the camera pose by minimizing the re-projection error. Ideally, it is important to match several 2D or 3D points in a computationally efficient manner to attain optimal camera pose. An improvement in the area of correspondence estimation can have a large effect on the scalibitliy of the reconstruction pipeline. In this thesis, we use only 2D-3D correspondences to estimate absolute pose with respect to the SfM model. We use the visibility and geometric information available in the SfM dataset to efficiently guide the search for quickly estimating 2D-3D correspondences.

- Camera Pose Estimation: Given the correspondences, direct linear transform (DLT) is used to estimate the camera projection matrix [29]. The projection equation is represented by $\mathbf{x} \simeq \mathbf{PX}$ where $\mathbf{X}$ is a vector of 3D points, $\mathbf{x}$ is the corresponding vector of 2D features and $\mathbf{P}$ is the estimated projection matrix mapping 3D points to the corresponding 2D features. $\mathbf{P}$ is estimated by solving the equation $\mathbf{x} \times \mathbf{PX} = \mathbf{0}$ using singular value decomposition. It requires a minimum of six correspondences to estimate the projection matrix using DLT. We discuss the DLT method in detail in chapter 4. CPC is inherently noisy and can often result in false correspondences. DLT method is noise sensitive and false correspondences can introduce error in the resultant camera pose. Therefore, DLT it applied in a RANSAC loop to make it more robust to outliers [18]. RANSAC tries to remove outliers by assuming that the inilers can be explained using a model where as outliers are independent of each other and do not follow any model. RANSAC iterates by randomly selecting 6 points which are used to fit a mathematical model. In each iteration, errors of all the other points are estimated relative to the fitted model and classified into inliers or outliers based on some threshold. After some iteration, the model which had maximum number of inliers is considered as the desired model and is re-evaluated using all the inliers.

1.3 Feature Triangulation

The problem of feature triangulation is defined as estimating the 3D coordinates of a given 2D feature. This problem is a dual of the image localization problem. Each camera sees multiple points and each point is seen by multiple cameras. Each image that is localized is matched to many 3D points and similarly, each feature that is triangulated has to be matched to many other features. Similar to image localization, an image feature which has to be triangulated is matched to nearby images. It uses a track of matching 2D features and the camera pose of images where the features are found to *triangulate* a 3D point. Traditionally, each image is matched with every other image to find pair-wise matches which are organized into tracks. This is usually done in a pre-processsing step. Performing pair-wise matching of all the images is a computationally expensive process. In this thesis, we introduce the problem of feature triangulation as estimating the 3D coordinates of a 2D feature using the SfM data.

- Feature Matching: Given a query 2D feature, our algorithm searches the space of cameras for visibility of the query feature using visibility information. As a result, a track of matching features is estimated. In contrast to the pair-wise matching approach where all the images are initially matched and then organized into tracks, we directly estimate a track of matching features and triangulate it to estimate the 3D point.

- Point Estimation: Given a track of 2D features and the corresponding camera poses, a 3D point is estimated such that it minimzes the reprojection error. That is, when the 3D point is projected back into the image using camera parameters, the difference between actual feature location and projected feature location is minimum. For multiview triangulation, we minimize the sum of re-projection errors across all images. Similar to the problem of localization where given the 2D and 3D correspondences, the camera projection matrix is estimated, here we estimate the 3D coordinate of a point given its projection coordinates and the corresponding projection matrices. More formally, we have 2D coordinates $\mathbf{x}$ and the projection matrices $\mathbf{P}$ available, in the projection equation $\mathbf{x} \simeq \mathbf{PX}$. We triangulate the 3D point $\mathbf{X}$ by decomposing the equation $\mathbf{x} \times \mathbf{PX} = \mathbf{0}$ using singular value decomposition.

In this thesis, we solve the problem of triangulation efficiently using the visibility probabilities to guide the search for cameras for quickly estimating correspondences across multiple images. Figure 1.5 depicts the process of localization and triangulation. We use the probability structure to efficiently solve the dual problem of localization and triangulation.

1.4 Efficiency of SfM

Correspondence estimation and the global optimization of 3D structure and camera parameters are the primary computational bottlenecks in large scale reconstruction process. Recently, computer vision researchers have tried optimizing the correspondence estimation or image matching step by distributing

the process over multiple compute cores or GPUs. Agarwal et al. addressed this problem by distributing the visual word based matching process over 500 compute cores to reconstruct 1M images of Rome in a day [4]. Frahm et al. used GPUs along with CPUs to reconstruct 3M images of Rome in less than 24 hours [19].

Given the estimated camera poses and triangulated points, the global optimization of cameras and points using bundle adjustment is another important step. It is a major computational bottleneck in the SfM pipeline. It ensures that the error in pose estimates does not get accumulated over the sequence. Bundle Adjustment (BA) is used to jointly optimize camera and point parameters. Bundle adjustment is an iterative step, typically performed using the Levenberg-Marquardt (LM) non-linear optimization scheme. It performs robust non-linear minimization of the re-projection error to accurately recover structure and motion. Bundle adjustment is one of the primary bottleneck of the SfM, consuming about half the total computation time. For example, reconstruction of a set of 715 images of Notre Dame data set took around two weeks of running time [77], dominated by iterative bundle adjustment. The BA step is still performed on a single core, though most other steps are performed on a cluster of processors [4]. Speeding up of BA by parallelizing it can have a significant impact on large scale SfM efforts. The computation requirements of BA grows rapidly with the number of images. However, the visibility aspects of points on cameras places a natural limit on how many images need to be processed together. The current approach is to identify clusters of images and points to be processed together [78]. Large data sets are decomposed into mildly overlapping sets of manageable sizes. An ability to perform bundle adjustment on about 500 images quickly will suffice to process even data sets of arbitrarily large number of images as a result. We focus on exactly this problem in this thesis. We provide a hybrid implementation of sparse bundle adjustment with CPU and GPU working together to achieve a speedup of 30-40 times on an Nvidia Tesla C2050 GPU on a dataset of about 500 images. We discuss its implementation in detail in Chapter 6.

1.5 Contributions of the Thesis

We improve various aspects related to the large scale reconstruction pipeline. In the following section we summarize our contributions as a part of this thesis.

- We define a probabilistic framework using the cameras and points available in the SfM output to effectively guide point and camera search in the case of image localization and feature triangulation respectively. We encode the visibility information between and among points and cameras as visibility probabilities. The conditional visibility probability of a set of points on a point (or a set of camreas on a camera) is used to select points (or cameras) based on their dependence or independence. The probability framework aids in estimating the priority of points and cameras based on their dependence on each other, distances, etc.

- Image localization uses a few 3D points to 2D feature mappings in the query image to estimate its camera parameters. It searches the space of points in the SfM dataset for match with the query image. We use the conditional visibility probability among points to effectively guide these searches. Our probability-guided exploration generates RANSAC inliers in high proportions as the matching proceeds. Our localization method successfully registers as many or more new images as the prior work. We reduce the number of nearest neighbor searches to 10-20% of what was done by Li et al. [42]. Due to the improved accuracy, we only need to generate 25% of matches as Sattler et al. [64].

- We define the problem of feature triangulation as the estimation of 3D coordinate of a given 2D feature using the SfM data. Feature triangulation identifies a few cameras in which the query feature is visible for triangulation. It searches the space of cameras for visibility of the query feature. We use the conditional visibility probability among cameras to effectively guide these searches. Our scheme can match 4 new images per second on a high end workstation. Our feature triangulation method increases the point density at typical locations by 30-50%. It is especially good at identifying points in regions with repetitive appearance, where the 2D-2D match of SfM performs poorly.

- We develop a practical time implementation of bundle adjustment by exploiting the computing resources of the CPU and the GPU. We decompose the LM algorithm into multiple steps, each of which is performed using a kernel on the GPU or a function on the CPU. Our implementation efficiently schedules the steps on CPU and GPU to minimize the overall computation time. The concerted work of the CPU and the GPU is critical to the overall performance gain. The executions of the CPU and GPU are fully overlapped in our implementation, with no idle time on the GPU. We achieve a speedup of 30-40 times on an Nvidia Tesla C2050 GPU on a dataset of about 500 images.

1.6 Outline of the Thesis

- Chapter 2: This chapter reviews the related background work in the area of SfM, image localization and feature triangulation. We discuss the state of art techniques in these areas.

- Chapter 3: In this chapter we discuss the visibility probability and joint visibility formulation defined using the points and cameras in the SfM Dataset.

- Chapter 4: In this chapter we model the image localization problem using joint visibility probability to estimate camera pose using few 3D-2D correspondences.

- Chapter 5: Similarly, we introduce the feature triangulation problem and propose an algorithm to estimate 3D point of a given 2D feature using visibility probability based guided search in the camera space.

- Chapter 6: In this chapter, we explain a hybrid and practical time implementation of bundle adjustment by exploiting the computing resources available on CPU and GPU.
- Chapter 7: In the last chapter, we conclude the thesis by discussing our contributions, existing problems with the model and the possible future work in this direction.

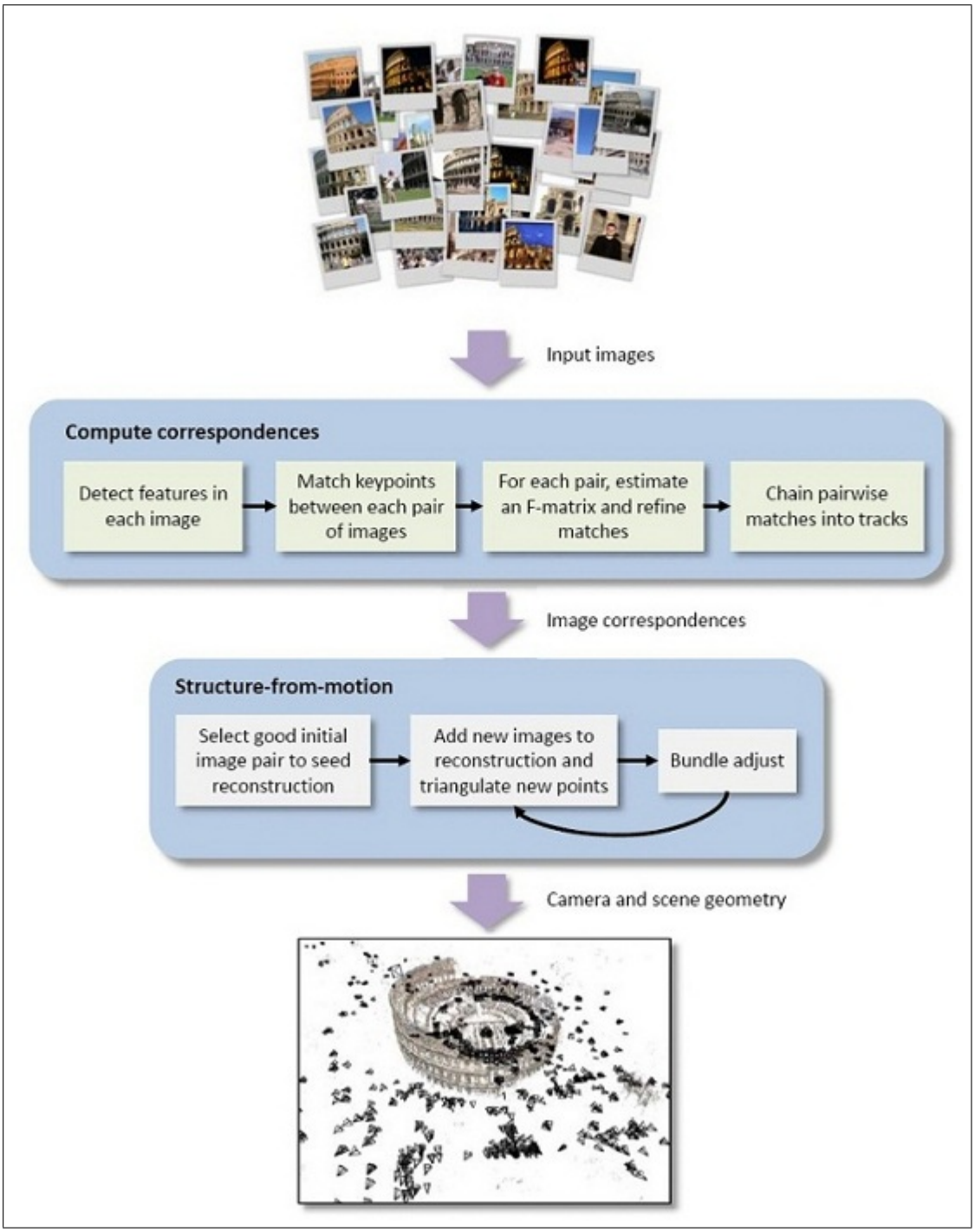


Figure 1.4: Large scale reconstruction pipeline using incremental structure from motion (Image Courtesy: Noah Snavely[75])

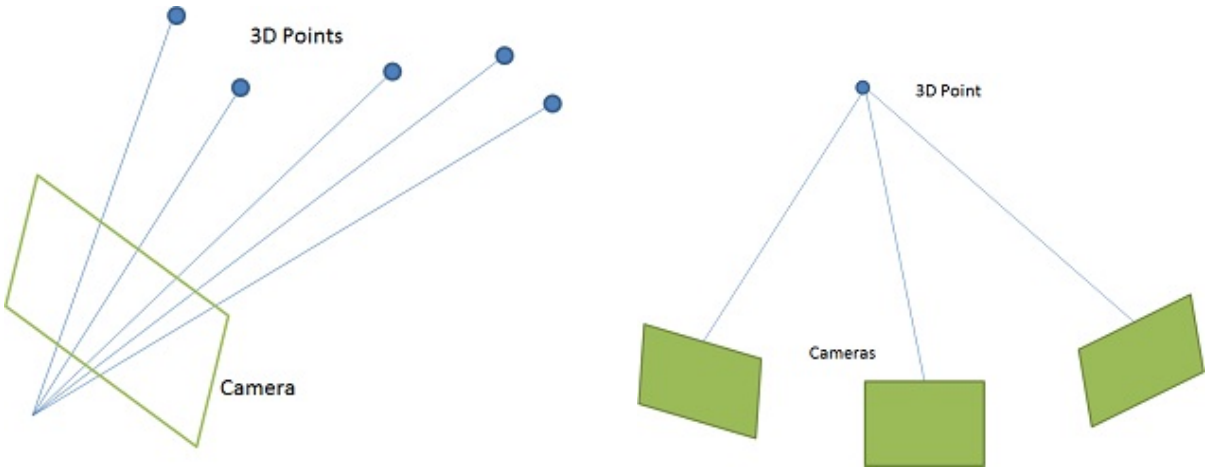


Figure 1.5: From Left to Right: Image depicting the localization and triangulation process respectively

Chapter 2

Related Work

In this chapter, we discuss the related work pertinent to the area of structure from motion, image localization, feature triangulation and bundle adjustment.

The geometric algorithms used in computer vision have its roots in the field of photogrammetry [85]. Photogrammetry is the science of taking 3D measurements of the world through photographs. One of the important problems in photogrammetry (or stereo-photogrammetry) is of estimating the three-dimensional coordinates of points on an object given multiple images from different view points. This problem is also known as estimating Structure (points) from Motion (of cameras). Reconstructing 3D world using images or videos is one of the core computer vision problem. Researchers have extensively worked on this problem. The related work in this area can be broadly divided into two categories: (a) Calibrated case and (b) Uncalibrated or Semi-calibrated case. We now discuss each one of these in detail.

2.1 Calibrated Reconstruction

Earlier research in the area of 3D reconstruction involved known camera matrices. The simplest problem in the area of calibrated reconstruction is binocular stereo or two-view SfM. Two-view SfM estimates the depth map of a scene using two images of the same scene seperated by some distance. Techniques to solve this problem include finding pair-wise correspondences and the corresponding matching costs between the pixels of the two images [8, 29]. Epipolar constraint is used to reduce the search space to a one dimensional line. The estimated depth of each matching pixel is then optimized using a global optimization which ensures smoothness in the world. Graph-cut based techniques were explored to solve this problem [9]. It models the pairwise smoothness in depth estimates and individual depth estimates of each pixel which is solved using a mincut algorithm. Scharstein and Szeliski present a comprehensive and comparative evaluation of several stereo matching algorithms [66].

Two-view SfM was later extended to multiple views. Multi-view stereo is used to reconstruct 3D model from images taken from previously known view points under calibrated conditions. Traditional multi-baseline stereo method was first proposed by Okutami and Kanade [57]. This approach estimated

the correct depth by taking into account the information available in multiple stereo pairs which is used to remove ambiguity. Volumetric approaches in this area represented the 3D scene using different models. Seitz et al. represented the scene as a set of discrete voxels which are filled using the pixel information of the images in which the voxel is projected [69]. Kutulakos et al. uses space carving techniques to reconstruct 3D model using multiple photos [38]. Some methods estimated the depth map of each image which are later merged to reconstruct the model [81]. Many of these techniques also enforce a photo consistency measure or a shape prior [68]. Photo consistency measure ensures an agreement between the model and its projections in the image. Shape priors constrain the 3D model using some pre-defined shape. It helps in the precise estimation of the geometry in low textured regions. Local smoothness constraints are also an example of local shape priors. These can be effectively modelled using graphs [9]. Seitz et al. present a quantitive comparision of different multi view stereo algorithms and show the results on six benchmark datasets [68].

More recently, multi-view stereo techniques have also been used with internet photo collections. Goesele et al. estimates the depth map for an image by doing view selection to find nearby cameras to match and uses SfM 3D points as a base to grow surface on these points [27]. Furukawa et al. decomposed the collection into different clusters which are processed in parallel and then merged [24].

2.2 Un-calibrated and Semi-calibrated Reconstruction

Reconstructing 3D world using uncalibrated views is a challenging problem in computer vision. Traditionally, this problem is solved by calibrating cameras through an offline process [88] and using the calibrated cameras to reconstruct 3D geometry as explained in the previous section. Recently, computer vision researchers have successfully calibrated images using the information available in the images only with no external calibration pattern used. This is done by estimating the pair-wise correspondences within all pairs of images and using those correspondences to find relative pose. Five point algorithms have been developed to estimate the relative camera pose of two views given intrinsic parameters (focal length, radial distortion etc.) [40, 55]. It uses five correspondences to estimate the relative pose. Calibration is completed when we estimate global pose of all the cameras in a common frame. Estimating global pose of many images is a challenging problem. This area can be roughly divided into two sub-areas on the basis of the data source and the calibration condition.

2.2.1 Reconstruction using Video

Most of the work in the area of multi-view reconstruction focussed on semi-calibrated or fully calibrated conditions. For example, frames of a video is a more controlled and organized image collection than the unorganized internet image collection. So, reconstructing 3D world using video frames is an easier problem than the reconstruction of internet photo collection. Researchers have exploited the temporal order of the video and its correlation to spatial coherency of frames to provide real-time recon-

struction from video [58]. Using the spatial coherency of frames, features are tracked across multiple frames. Therefore, it is easier to find correspondence across multiple frames in a video and hence relative pose estimation can be done in real time. Camera location of each frame is estimated using Kalman filter [36] using the measurements accumulated over time to optimize the current estimate. Depth map of each frame is estimated and merged to give us the final reconstruction. Similar problem is solved in the case of robot navigation. Simultaneous localization and mapping (SLAM) in the case of robotic vision is similar to solving SfM problem using videos. Davison et al. [15] demonstrate a real time vision-based SLAM system which has an active approach to mapping and measurements so that least number of image processing operations are done. Kaess et al. proposed a Bayes tree representation and its application is robot trajectory optimization [34]. They represented the non-linear optimization and matrix factorization as an update in the probability densities or the structure of the Bayes tree. Klein and Murray developed a parallel tracking and mapping system and showed its application in the area of augmented reality [37].

2.2.2 Reconstruction using Internet Photo Collection

With the advent of powerful feature matching techniques, reconstructing unorganized and uncontrolled photo collection using Internet imagery has gained recent attention from the computer vision community. Brown and Lowe [10] were one of the first to develop SfM systems that successfully reconstructed unordered image sets captured by a single photographer. Schaffalizsky and Zisserman [65] proposed a system to organize the unorganized image datasets of a particular location taken from several view points. They developed an indexing scheme over the image patches to perform large scale matching. Two and three view constraints are used to remove erroneous matches. Techniques similar to these are also used in the recent structure from motion pipelines. SfM algorithms can be divided into two broad categories. One of them is incremental SfM where we start with small seed reconstruction and then grow it repeteadly using by adding cameras and points. Another one is batchwise SfM which considers all the point and camera parameters at once rather than incrementally building up the solution [77]. We now discuss each one of these in detail.

2.2.2.1 Batchwise SfM

These techniques considers all the camera and point parameters at once and do not order them in any sequence. It considers all the images as equal. It solves the complete problem in a single batch optimization. One of the earlier methods to solve batch wise SfM includes factorization based approaches [7, 80]. Factorization methods work by decomposing measurement matrix $M = PX$ into camera matrices P and 3D points X. It requires all the points to be visible in all the views and hence it cannot handle missing data. Missing data is quite common in community photo collection due to the sparsity of the scene. Linear reference-plane based technique minimizes algebraic error [63]. The problem with this approach is that points close to infinity can bias the reconstruction unless a good initiailization is

provided. L_∞ error based optimization was also explored [2]. It minimizes the maximum reprojection error of the measurements rather than the sum of reprojection error (as in bundle adjustment). The L_∞ based optimization is shown to be convex and hence does not depend on a good initiailization . However, L_∞ methods cannot scale to large number of camera and point variables and are not robust to outliers. Another set of linear methods estimate global camera parameters using pairwise constraints. These are also not very robust to outliers. Some of the recent methods include linear SfM method by Sinha et al. [72]. It incorporates vanishing points in estimating camera orientations. Given rotations, pairwise reconstructions are aligned globally to estimate translation. More recent work by Crandall et al. used an MRF framework to model the constraints between camera and scene pairs to initialize the SfM pipeline well [14].

2.2.2.2 Incremental SfM

Most of the recent SfM systems are sequential where starting with a smaller reconstruction, cameras are added by pose estimation and points by triangulation [4, 19, 77]. Incremental approaches minimize error propogation using intermediate Bundle Adjustment and multiple rounds of outlier removal. Frequent use of non-linear minimization techniques like bundle adjustment slows down these methods. The worst case running time of these methods is $O(n^4)$ in the number of images. Theses methods do not treat all images equally, producing different results depending on the order in which photos are considered. This can results in local minima or large error drift in the final camera/point parameters. Similar to these, hierarchical systems involve clustering images using agglomerative clustering to form a tree and reconstructing the tree from leaves to the root, merging nodes at each level [17, 25]. This method doesn't select pair of views and it copes better with the error drift. PhotoTourism is the first application to create large SfM datasets from unstructured photographs and to interactively browse them using the points and cameras [77]. However basic SfM method has $O(n^2)$ matching complexity as we have to do all pair-wise image matches to find feature correspondences and it is not scalable to city scale collections.

Inspired by the progress in document analysis, computer-vision researchers have recently begun using the idea in the area of content based image retrieval. The idea is to cluster SIFT features in a photo collection into "visual words". By representing each image as a collection of visual words, researchers are able to apply document-retrieval algorithms to efficiently match large dataset of photos. Agarwal et al. addresses this problem by distributing the visual word based matching process over 500 compute cores to reconstruct 1M images of Rome in a day [4]. Similarly Frahm et al. used GPUs along with CPUs to reconstruct 3M images of Rome in less than 24 hours [19]. Even after optimizing the matching step, performing incremental SfM over the whole image collection is prohitively expensive. Internet image collection are inherently redundant. Many images are taken from nearby locations. So, it is necessary to find a minimal set of images that best represent the collection. To address this issue, Snavely et al. constructed skeletal set of the image collection which ensured model completeness and bounded loss of accuracy as compared to full image set reconstruction [78]. Skeletal set is created

by finding maximum-leaf spanning tree of the match graph. Li et al. uses GIST based clustering along with geometric verification to find iconic images that provide a compact scene summary [41]. However, for city scale collections, the size of the compressed sketal or iconic graph can be large too making the process computationally intesive. This problem is currently being explored. Researchers have also tried optimizing the bundle adjustment step using better preconditioned conjugate gradient based optimization which exploits the available sparsity in the scene structure [11, 45, 53]. We will discuss the detailed related work of bundle adjustment in the later part of this chapter.

2.3 Image Localization

The problem of image based location recognition has received much attention from the computer vision community in the recent years due to an increase in massive image collection on the internet and various applications in urban navigation and augmented reality. One of the earliest location recogntion system was developed by Robertson and Cipolla [62], which uses a database of rectified images of building facades to determine the pose of the query image. Another work by Zhang and Kosecka [87] used SIFT features based matching to find the nearest views to the query image and use them to estimate the camera pose. However these methods focussed on small databases of 200-1000 images.

With the advancement in large scale image retrieval techniques [56, 74], similar methods were employed in the problem of location recogntion [31, 32, 64, 67, 21]. Schindler et al. [67] first used informative features to build specific vocabulary tree over a dataset of 30,000 geo-tagged images in order to solve the problem of city scale location recognition. Fraundorfer et al. [21] proposed distance augmented visual word based image retrieval which scores each feature by the distance to its nearest word. Hays and Efros [31] used 6 million geo-tagged images to estimate the location of a new query image as a probability distribution over the entire earth. This is done by finding out K-nearest neighbors of the query image in the geo-tagged dataset and using the corresponding GPS coordinates to estimate the probability distribution.

Image based localization has been studied in the SLAM literature as well. Achar et al. uses modified bag-of-words approach for image localization in large scale urban environments [1]. Distinctive words are given higher weights which are used along with a vocabulary tree to find the nearby image for the given query image. The resulting images are then geometrically validated to get the localization result. Fraundorfer et al. [20] used visual words based localization algorithm for an indoor environment. SLAM based methods as mentioned use video sequence of a camera mounted on a robot as an input and do not take into account the reconstructed SfM output for localization. Another work by Alcantarilla et al. [5] used noisy camera pose priors available through GPS sensors and the reconstructed SfM output to predict the set of 3D points visible in the new query image. Chli and Davison [13] proposed active matching techniques in the context of SLAM algorithms, which resulted in robust, real time matching.

Other than these, many researchers have also explored the usage of compressed scene representation for better image registration or localization. Li et al. [39] reduced the size of feature set used to represent

each image retaining the most informative features. Ni et al. [52] uses epitomes of locations created from videos of different scenes. Similarly in the area of SfM, Li et al. [41] uses iconic scene graph and Snavely et al. [78] proposed skeletal set to compress the model.

Our work on image localization is closely related to the work of Irschara et al. [32], Sattler et al. [64] and Li et al [42].

2.3.1 Vocabulary Tree based approach (Irschara et al.)

Irschara et al. [32] used SfM point cloud to generate synthetic images in order to ensure that the whole model is covered properly and is not just limited to the original views which are a part of the reconstruction. Original and synthetic views are then combined to form the database for image retrieval step. They use vocabulary tree to find nearby image from the set of images and use the 3D points belonging to the features in the retrieved image to form 2D-3D correspondences and use that to estimate the final camera pose. Figure 2.1 shows the synthetic and original images (left) and an example of image registered with respect to the SfM point cloud (right).

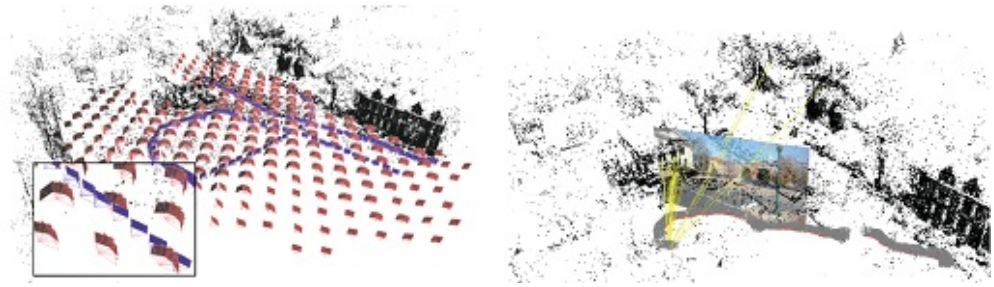


Figure 2.1: (Left) A SfM point cloud and the original images(shown in blue) and the synthetic images(shown in red). (Right) An example of image registered with respect to the SfM point cloud

2.3.2 Prioritized Feature Matching (Li et al.)

Li et al. proposed a 3D-2D matching scheme using a prioritization on SfM points for matching points in the query image [42]. They explore the model by sequentially matching a subset of highly visible points that cover the space. The priority of points changed with more matches using a heuristic based on the visibility of matched points. We take a similar approach but restrict points and prioritize them using the underlying visibility probability structure among points.

2.3.3 2D-3D Matching (Sattler et al.)

Sattler et al. used a 2D-3D point matching method guided by visual words [64]. 3D points are represented by the mean of SIFT vector of all the track points and then quantized to a visual word. Therefore, we get a mapping from each 3D point to a visual word. They arranged the 3D points as

lists that quantized to the same visual word. Features in the query image were matched to these lists using linear search to get a large number of highly noisy matches. They used RANSAC to overcome the highly noisy nature. This produced good results even with noisy matches. We adapt this method to find a seed point and use a probability-guided matching that identifies inliers early with a lot less effort. Figure 2.2 illustrates Sattler et al.'s framework of direct 2D-to-3D matching.

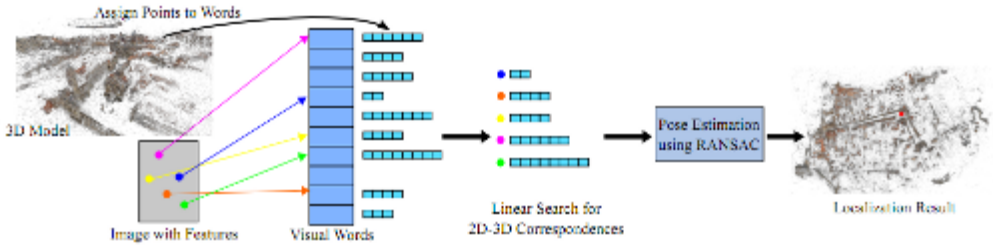


Figure 2.2: Illustration of Sattler et al.'s direct 2D-to-3D matching framework (from Sattler et al. [64])

2.4 Feature Triangulation

Feature triangulation is one of the important problems in photogrammetry. Triangulation is the problem of finding a point in space given its position in two camera images taken with known calibration and pose. Generally triangulation is done using a sub-optimal method which is later minimized using global optimization techniques like bundle adjustment. Optimal solution for the case of two views in shown in [28]. It assumes a Gaussian noise model and triangulates new points using least squares minimization. For three views, the optimal solution is shown by Stewenius et al. [79]. Another method in the case of calibrated cameras is to find the mid-point in 3D space of the rays back- projected from the image points [28]. However this method doesn't work for projective reconstruction. Minimizing L_2 error as in the case of non-linear least squares minimization can result in local minima and requires good initiailization to obtain global minimum. Finding globally optimal triangulation using L_∞ based formulation is also explored in [35]. Using L_∞ optimization comes down to minimizing a cost function with a single minima on a convex domain as compared to L_2 based optimization which can have multiple minimas. Figure 2.3 shows the minimas for a three view triangulation problem. The problem with L_∞ based formulation is that it is not robust to outliers and can not scale to large problems. Since most of these methods rely on accurate camera pose estimation to give good triangulation, bundle adjustment is used which also takes into account the correction of projection matrices by minimizing the sum of squared distances between the measured image points and the reprojected 3D points.

Goesele et al. applied a multiview stereo approach using the images and cameras of community photo collections reconstructed using SfM [27]. They selected nearby images to produces depth maps, which were merged to produce a model. Furukawa et al. decomposed the photo collections into clusters and processed them independently using a patch-based stereo approach before merging the results [24].

Reconstruction using calibrated, uniformly distributed views [26] and using street-view datasets [58, 48] have been reported. Fuhrmann et al. [22] fused depth maps at different resolutions using scale cues to produce adaptive meshes.

Repetitive structures cause problems in the SfM matching pipeline. Roberts et al. [61] recovered structure in the presence of large duplicate structures by removing erroneous matches. They used an expectation maximization based algorithm over camera poses and correctness of the matches.

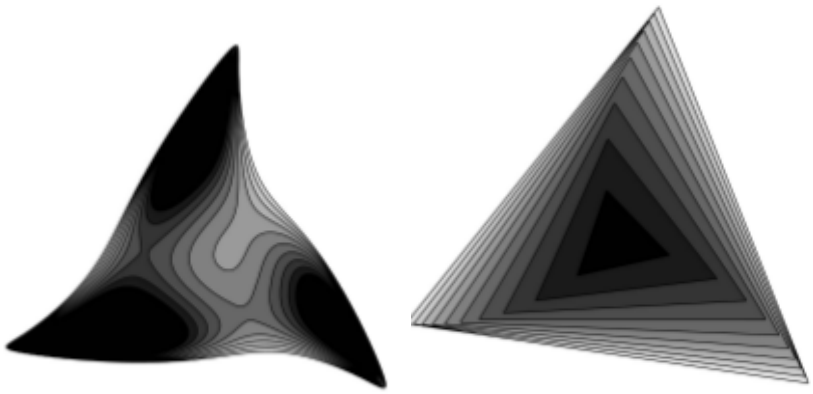


Figure 2.3: (From Left to Right) Contour plot of the L_2 and L_∞ error for a three view triangulation problem. L_2 cost function has three local minimas where as L_∞ cost function has a single minima (Image Courtesy: Fredrik Kahl and Richard Hartley [35])

2.5 Practical Time SfM Computation

Computational efficiency is one of the primary issues in the large scale reconstruction process with correspondence estimation and non-linear minimization using bundle adjustment being one of the major bottlenecks. In this thesis, we also focus on making the process of bundle adjustment more efficient using CPUs and GPUs. We discuss the related work of bundle adjustment here.

Bundle Adjustment (BA) was originally conceived in photogrammetry [45], and has been adapted for large scale reconstructions. It is the problem of refining a visual reconstruction to produce jointly optimal structure and viewing parameter estimates. The names refers to the 'bundle' of light rays leaving each 3D feature and converging on each camear centCalibration can be tougher which are adjusted optimally with respect to both feature and camera positions. In general, a sparse variant of Levenberg-Marquardt minimization algorithm [44] is the most widely used choice for BA. A public implementation is also available [45]. Ni et al. solve the problem by dividing it into several submaps which can be optimized in parallel [53]. Byröd and Åström solve the problem using preconditioned conjugate gradients, utilizing the underlying geometric layout [11]. Agarwal et al. explore the performance of different preconditioning strategies on large scale bundle adjustment datasets [3]. They use the block diagonal of Hessian matrix and the block diagonal of its Schur complement as preconditioners without explicitly

storing the schur complement in the memory. Cao et al. parallelize the dense LM algorithm, but their method is not suited for sparse data [12]. They don't use the GPUs for the BA step. No prior work has been reported that parallelizes BA or the LM algorithm. We provide a hybrid implementation of sparse bundle adjustment with CPU and GPU working together to achieve a speedup of 30-40 times on an Nvidia Tesla C2050 GPU on a dataset of about 500 images.

2.6 Computer Vision on GPUs

The rapid increase in the performance has made the graphics processor unig (GPU) a viable candidate for many compute intensive tasks. the match of the data-parallel computations to many operations on images has made GPUs an ideal choice to port many computer vision algorithms. Recent advances such as CUDA and the OpenCL standard have the potential to accelerate the use of GPUs in many areas for more general purpose computing, including Computer vision GPUs are being used for many computer vision applications [23], such as Graph Cuts [83], tracking [73] and large scale 3D reconstruction [19].

Vineet and Narayanan proposed an implementation of Graph Cuts on GPU by efficiently parallelizing the push-label algorithm and performed 10-12 times faster than the best sequential algorithm reported [83]. Sinha et al. presented a real time GPU based KLT tracker that is able to track about a thousand features in real time. They also proposed a GPU based implementation of SIFT feature extraction algorithm which was 10 times faster than the corresponding state of art on CPU [73]. Frahm et al. used GPUs to efficiently perform apperance based clustering of GIST features corresponding to 3 million images. This lead to an increase in scalibility of the large scale reconstruction systems [19]. Recently, Wasif and Narayanan presented an implementation of K-Means clustering algorithm on multiple GPUs. They were able to cluster up to 6 million 128 dimensional vectors and achieved a speed up of 170 compared to the CPU implementation [50]. Wasif and Narayanan also demonstrated its application in a video organizing system, where they're able to organize a set of 100 sport videos in about 9.5 minutes [49].

2.7 Summary of our Contributions

In the following chapters, we present our contributions in the area of image localization, feature triangulation and bundle adjustment. We propose a visibility probability framework using the cameras and points available in the SfM data to effectively guide point and camera search in the case of image localization and feature triangulation respectively. We encode the visibility information between and among points and cameras as visibility probabilities. The conditional visibility probability of a set of points on a point (or a set of camreas on a camera) is used to select points (or cameras) based on their dependence or independence. The probability framework aids in estimating the priority of points and cameras based on their dependence on each other, distances, etc.

Our image localization algorithm guided using visibility probability uses fewer 2D-3D correspondence to estimate the camera pose. We get high percentage of RANSAC inliers and we register more number of images as compared to the other recent algorithms [64, 42]. Li et al. [42] explore the model by sequentially matching a subset of highly visible points that cover the space. The priority of points changed with more matches using a heuristic based on the visibility of matched points. We take a similar approach but restrict points and prioritize them using the underlying visibility probability structure among points. Sattler et al. [64] used 2D-3D matching and produced good results even with noisy matches. We adapt this method to find a seed point and use a probability-guided matching that identifies inliers early with a lot less effort.

We also introduce the problem of feature triangulation as the estimation of 3D coordinate of a given 2D feature using the SfM data. Recent approach by Agarwal et al. uses bag of visual words based technique to find nearby images to match [4]. Frahm et al. performs clustering based on GIST features to perform image matching [19]. Tracks of matching feature points are then triangulated. In contrast with these approaches, we use visibility probability to guide the search in camera space and finds cameras in which the query feature is visible. Our method is also similar to the work by Goesele et al.. Goesele et al. applied a multiview stereo approach using the images and cameras of community photo collections reconstructed using SfM [27]. They find nearby views using local view and global view selection based on some heuristics. On the other hand, we propose a visibility probability based formulation to guide our search. Our feature triangulation method increases the point density at typical locations by 30-50%. It is especially good at identifying points in regions with repetitive appearance, where the 2D-2D match of SfM performs poorly.

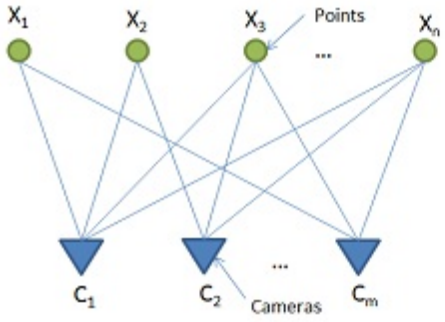
In the last part of this thesis, we discuss a practical time implementation of bundle adjustment by exploiting the computing resources of the CPU and the GPU. The executions of the CPU and GPU are fully overlapped in our implementation, with no idle time on the GPU. We achieve a speedup of 30-40 times on an Nvidia Tesla C2050 GPU on a dataset of about 500 images.

Chapter 3

Visibility Probability and Joint Visibility Probability

We propose a visibility probability framework using the cameras and points available in the SfM data to effectively guide point and camera search in the case of image localization and feature triangulation respectively. The SfM dataset consists of m cameras, n points, and the visibility of each point's track in the cameras. This can be represented as a bipartite, visibility graph [42] as shown in Figure 3.1. Here each node represents either a camera or a 3D point. An edge between the nodes is formed when a camera sees a 3D point or a point is projected onto a camera. A typical SfM dataset consists of a few thousands of images and few million 3D points. It represents a rich sampling of the geometry and appearance.

We encode the visibility information between and among points and cameras as visibility probabilities. Points and cameras act as the random variables in these probabilities. The conditional visibility probability of a set of points on a point (or a set of camreas on a camera) is used to select points (or cameras) based on their dependence or independence. The probability framework aids in estimating the priority of points and cameras based on their dependence on each other, distances, etc. Now we define the visibility probabilities using the points and the camera informations.



$$p(X_i) = \frac{d(X_i)}{m} = \frac{no.of\ cams\ that\ see\ X_i}{m}$$

$$p(C_i) = \frac{d(C_i)}{n} = \frac{no.of\ points\ seen\ in\ C_i}{n}$$

$$p(X_i, X_j) = \frac{no.of\ cams\ that\ see\ X_i\ and\ X_j}{m}$$

$$p(C_i, C_j) = \frac{no.of\ points\ seen\ in\ C_i\ and\ C_j}{n}$$

Figure 3.1: Bipartite graph of points and cameras. Visibility probability is approximated by fractions of cameras that see single point/camera or a pair of them.

3.1 Visiblity Probability

Visibility probability of a point with respect to a number of images can be interpreted as the probability that it is visible in a random query image. Similarly the visibility probability of a camera w.r.t. a set of points can be interpreted as the probability that a camera sees a random point. It is high for popular cameras and points. In the following sections, we define the visibility probabilities of a point and a camera in detail.

3.1.1 Visibility Probability of a Point

We define the *visibility probability* of a point X_i as the probability that a given point X_i is visible in a randomly selected view. We approximate it with respect to the SfM dataset as the fraction of cameras in which the point is visible as

$$p(X_i) = \frac{\text{number of cams that see } X_i}{\text{total number of cameras}} = \frac{d(X_i)}{m}, \tag{3.1}$$

where $d(X_i)$ is the degree of X_i in the graph. This approaches the true visibility probability when camera positions are dense. Visibility probability of a point is high if it seen by large number of cameras which is true in the case of 3D points at popular locations. Therefore, given no information about the query image it is likely to be captured at a popular place. From a tourist's point of view, if a lot of tourists captured images of a location, it is higly probable that the new user will also take a picture of that location. In this way, visibility probability of a point with respect to a CPC takes into account the user statistics of the popularity of a location.

3.1.2 Visibility Probability of a Camera

Similarly we define the *visibility probability* of a camera C_i as the probability that a random point is visible in the given camera C_i. We approximate it with respect to the SfM dataset as the fraction of points visible in the camera as

$$p(C_i) = \frac{\text{number of points seen in } C_i}{\text{total number of points}} = \frac{d(C_i)}{n}, \tag{3.2}$$

where $d(C_i)$ is the degree of C_i in the graph. This approaches the true visibility probability when point positions are dense. Visibility probability of a camera is high if a lage number of 3D points project to that camera. It implies that the visibility probability of a popular camera is more than a unpopular camera. Therefore, given no information about a query point it is likely to be captured by a popular camera which sees relatively higher number of points. From a tourist's point of view, if a tourist took an image of a very wide area or a popular location which contains a lot of reconstructed 3D points, it is highly probable that a new query 3D point is also visible in that image. In this way, visibility probability of a camera takes into account the popularity of an image.

3.2 Joint Visibility Probability

We define joint visibility probabilities to establish point-point and camera-camera relationships and propogate information from one node to another. We use joint visibility probability to guide point search in the case of image localization and to guide camera search in the case of feature triangulation.

3.2.1 Point-Point Relationship

The *joint visibility probability* $p(X_i, X_j)$ of two points is defined as the fraction of cameras in which both are visible (Figure 3.1). It can be formulated as,

$$p(X_i, X_j) = \frac{\text{number of cameras that see } X_i \text{ and } X_j}{m} \tag{3.3}$$

Joint visibility probability of two points is high if it is jointly visible in a large number of cameras and is low if two points are jointly visible in relatively less number of cameras. It measures the mutual dependence of the two points on each other. The joint visibility probability is zero for a pair of points if there exists no camera that sees both points. During image localization, we consider only those points which have non-zero joint visibility probability with the already matched points.

3.2.2 Camera-Camera Relationship

The joint visibility probability $p(C_i, C_j)$ of a cameras C_i, C_j with respect to a SfM dataset as fraction of points visible in C_i and C_j respectively.

$$p(C_i, C_j) = \frac{\text{number of points that is seen in } C_i \text{ and } C_j}{n} \tag{3.4}$$

Joint visibility probability of two cameras is high if a large number of points is jointly visible in both of them and is low if a less number of points is jointly visible in both of them. It measures the mutual dependence of the two cameras on each other. The joint visibility probability is zero for a pair of cameras if there exists no point that is seen in both of the cameras. During feature triangulation, we consider only those cameras which have non-zero joint visibility probability with the already matched cameras.

3.3 Conditional Probability and Dependence

We use conditional probability over the joint visibility probability and visibility probabilities to propagate information from one node to another. It is used to find the probability of node A given node B has matched. Conditional probability is used to guide the search during image localization and feature triangulation. Conditional probability of A given B is defined as the ratio of joint visibility probability of A and B and the visibility probability of B.

3.3.1 Conditional Probability of a Point w.r.t. a set of Points

The conditional probability of a point X_i on X_j is the fraction $d(X_i)$ of cameras that see X_i, in which point X_j is visible. In terms of visibility probabilities, conditional probability of a point X_i on X_j is defined as the ratio of joint visibility probability of X_i and X_j and the visibility probability of X_i.

$$p(X_j|X_i) = \frac{p(X_j, X_i)}{p(X_i)} = \frac{\text{number of cameras that see } X_i \text{ and } X_j}{\text{number of cameras that see } X_i}. \tag{3.5}$$

Equation 3.5 gives the probability of X_j being visible given that X_i is, i.e., the dependence of X_j on X_i. The dependence will be high on nearby points but could be lower if occluded in some views. The conditional probability of a point X_i on X_j is one if all the cameras that see X_i also see X_j. The points X_i and X_j are mutually independent if none of the cameras see both X_i and X_j.

How does the visibility of a set S of points influence the visibility of X_j? A point is independent of a set of other points only if it is independent of each point in the set. This leads to the expression

$$p(X_j|S) = 1 - \prod_{x_i \in S} \left[1 - p(X_j|x_i)\right]. \tag{3.6}$$

Here $p(X_j|S)$ measures the dependence of one point X_j with respect to the set of points S. Similarly, $1 - p(X_j|S)$ is used to measure the independence of a point X_j with respect to the set of points S. $p(X_j|S)$ is zero if the point X_j is mutually independent of all the points in S. $p(X_j|S)$ is one if the points X_j is completely dependent ($p(X_j|X_i) = 1$) on atleast one point in S. The dependence (or independence) between selected points and new points can guide the search for visible points in the given camera to the most promising regions in camera localization.

3.3.2 Conditional Probability of a Camera w.r.t. a set of Camera

The conditional probability of a camera C_i on C_j is the fraction $d(C_i)$ of points that is seen in C_i, which is also visible in C_j. In terms of visibility probabilities, conditional probability of a camera C_i on C_j is defined as the ratio of joint visibility probability of C_i and C_j and the visibility probability of C_i. The conditional visibility probability of cameras C_i on C_j then becomes

$$p(C_j|C_i) = \frac{p(C_j, C_i)}{p(C_i)} = \frac{\text{number of points common between } C_i, C_j}{\text{number of points seen in } C_i}. \tag{3.7}$$

Equation 3.5 gives the probability that a point is visible in C_j, given that it is visible in C_i as well, i.e., the dependence of C_j on C_i. The conditional probability of a camera C_i on C_j is one if all the points that is seen in C_i also see C_j. The cameras C_i and C_j are mutually independent if none of the points is seen in both C_i and C_j. $p(C_j|C_i)$ gives the dependence between the cameras, namely, the probability that C_j also sees a point given that C_i sees it.

A camera is independent of a set of other cameras only if it is independent of each camera in the set. The dependence of a camera C_j on a set S of selected cameras can be defined as

$$p(C_j|S) = 1 - \prod_{C_i \in S} \left[1 - p(C_j|C_i)\right]. \tag{3.8}$$

Here $p(C_j|S)$ measures the dependence of one camera C_j with respect to the set of cameras S. Similarly, $1 - p(C_j|S)$ is used to measure the independence of a camera C_j with respect to the set of cameras S. $p(C_j|S)$ is zero if the point C_j is mutually independent of all the cameras in S. $p(C_j|S)$ is one if the cameras C_j is completely dependent ($p(C_j|C_i) = 1$) on atleast one camera in S. The dependence or independence between previously identified cameras and new ones can guide the search for cameras that see a point.

The probability structure between and among points and cameras can be used to predict which points are likely to be visible in a camera given a few other visible points and which camera see a point given a few other cameras that see it. This helps to efficiently search for points visible in images or cameras that see features.

Chapter 4

Image Localization

Given a query image, localization algorithms answers the question "Where am I?". It is an important problem in the computer vision and robotics. Image localization can either be done using higher semantic level algorithms or using data-driven image retrieval based algorithms . In the case of semantic location recognition, we can use scene/category recogntion algorithms to match an image to a relevant category like whether an image is taken inside a room or a lab or in a field etc. Scene/Category recognition is done on a semantic level using high level features. However in the case of SfM (structure from motion) or SLAM (simultaneous localization and mapping), where we have many images taken at a location, image localization is used to estimate the exact location the image was captured from relative to the already localized images or to the map or 3D model built using those images. Here we solve the problem of localizing an image relative to the model reconstructed using structure from motion. Figure 4.1 depicts the localization question that we solve in this thesis. Image localization aims to compute the camera parameters of a given query image by matching a number of SfM points in them. Image

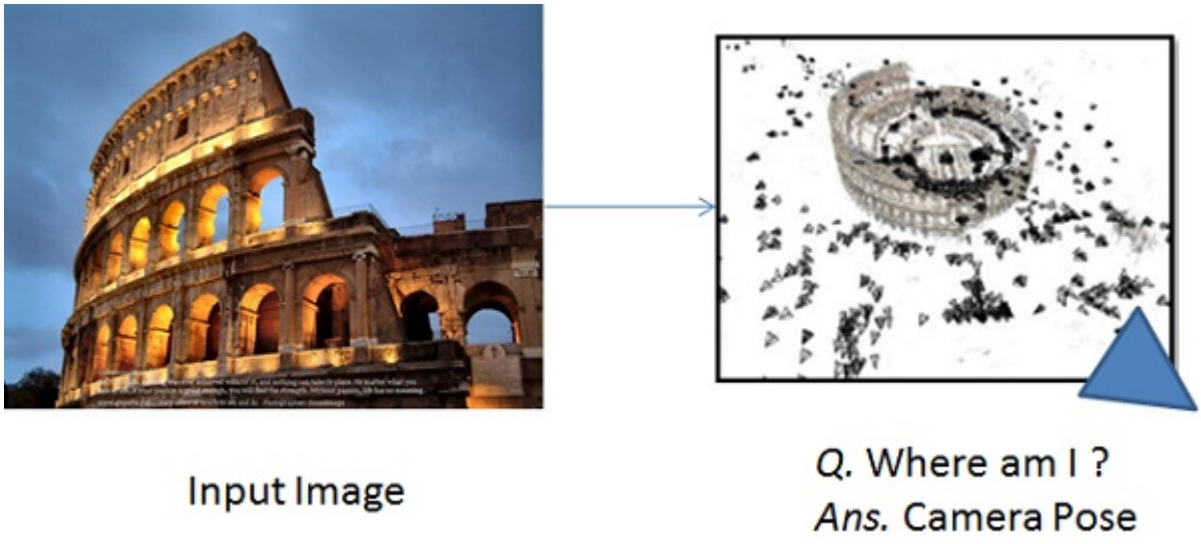


Figure 4.1: The Image Localization Problem: Where am I?

localization can be done using these two major steps:

- Feature Matching: The 2D features in an image are matched to 3D points in a model to estimate the *absolute* orientation with respect to the model.
- Camera Pose Estimation: Given the correspondences, direct linear transform (DLT) is used to estimate the camera projection matrix [29].

Our image localization algorithm is guided by the conditional visibility probabilities among points (Eq. 3.6). The proposed localization algorithm can be divided into three basic parts:

- Seed generation: Seed generation attempts to find a seed point to initiate the matching process.
- Probability-guided point matching: Point matching quickly identifies 3D points that match the query image using the visibility probability to guide the search.
- Camera estimation: In this phase, we recover the camera parameters from the matches obtained. Direct linear transform (DLT) is used within a RANSAC loop to estimate the paramters robustly.

In the following section, we explain the architecture of the complete localization system.

4.1 Architecture of our Localization System

In this section we explain different components used in the localization system. Figure 4.2 shows the architecture of our localization system. The whole architecture is divided into two major sections:

1. Pre-computation, which includes the precomputed datastructures which help in our matching framework
2. Query image localization, which includes online components which are processed when we are given the query image.

We now explain each one of these sections in detail.

4.1.1 Offline Processing

The pre-processing step consists of pre-computing the model and other data-structures which helps in fast query localization. The major pre-computations involved are:

- Model Compression: A typical SfM dataset contains a large number of 3D points and cameras. So, we compress the model to reasonable number of 3D points and cameras and ensuring the coverage of model.

- Mean SIFT Point Cloud: In order to estimate 3D-2D correspondences, both the 3D points and 2D features have to represented in the common SIFT space. To ensure that, point is represented as the mean SIFT of all the track point SIFT features

- Pair-wise probabilities matrix: The joint visibility probability of all pairs of points is pre-computed and stored as a sparse matrix.

- Visual word based word to point list: The mean SIFTs corresponding to each 3D point is quantized to a visual word to estimate a mapping from each word to a set of 3D points.

Figure 4.2 (Right) represents the flow of various pre-computations done as a part of the localization system. We now explain each one of these in detail.

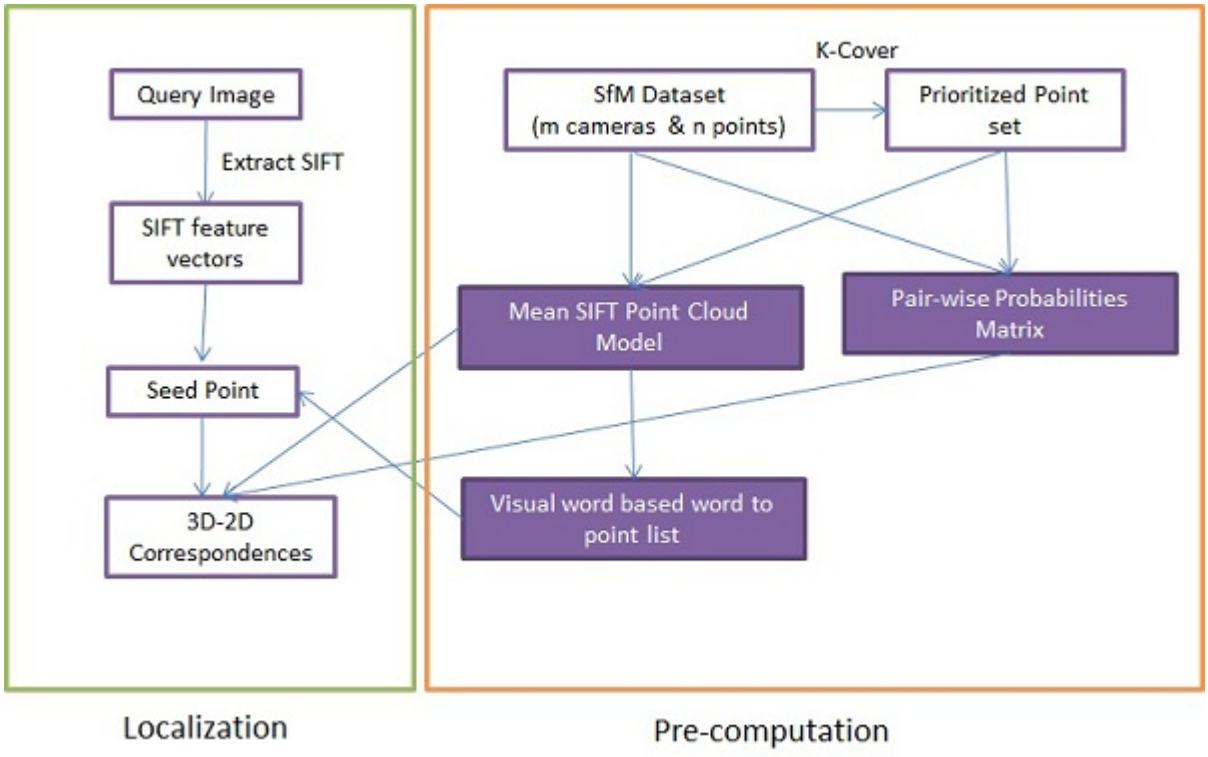


Figure 4.2: Flowchart showing the dependency of various components on each other. Filled boxes represent the pre-computed datastructures.

4.1.1.1 Model Compression

SfM datasets contain large numbers of points, all of which may not be informative. We use the method by Li et al. to compress the number of points using the K-Cover algorithm to include the most valuable points [42]. We use 3% to 5% of the original number of points in our experiments.

K-Cover compresses the model and ensures that each camera sees atleast K points. Optimal K-Cover is NP-Hard. An approximate version of K-Cover uses a greedy algorithm that always selects a point which covers the maximum number of not yet covered images. So, in each iteration we selec points until all images are covered atleast once and this iteration is run K times to cover all the images atleast K times. In case some of the images has less than K points visible, we remove that image from the iteration after it has no more point left. This sort of compression ensures the complete model is covered uniformly and not only in popular regions. We can have points in the sparser region as well. Figure 4.3 shows the reconstructed point cloud model of the Dubrovnik city built using community photo collection. In the left, we have the full model and in the right, we have the compressed model processed using the K-Cover algorithm. We can see that the left model has more number of points than the right one.

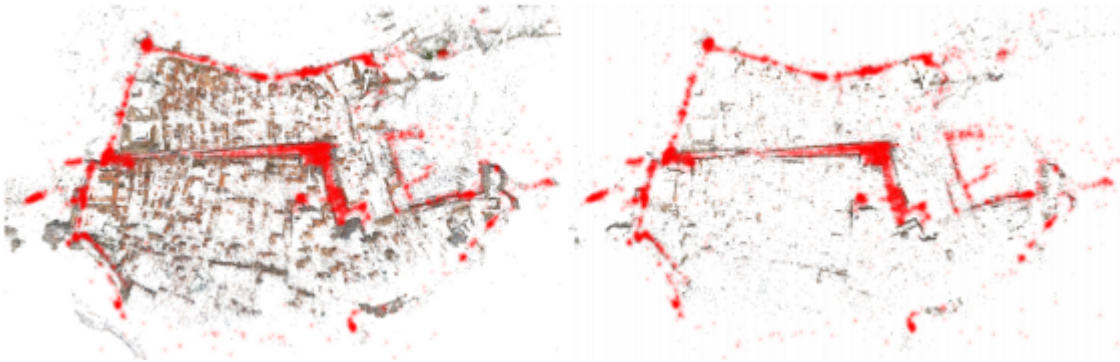


Figure 4.3: Reconstructed 3D point cloud for Dubrovnik city. Left: full model. Right: compressed model using K-Cover. The bright red regions represent the distribution of reconstructed camera positions. (Image Courtesy: Li et al.[42])

4.1.1.2 Mean SIFT Point Cloud Model

The SfM dataset consists of a large number of points and cameras and their mappings onto each other. Each 3D point is seen by a set of cameras along with the corresponding 2D SIFT feature in each of the camera. The set of 2D feature points corresponding to a 3D point is called its *track*. Similarly, each image sees a set of 3D points which maps to the corresponding 2D feature points. We represent each point as the mean SIFT of all SIFT features of its track. The 3D point and the 2D mean SIFT vector represent each 3D point in the common SIFT space. This would help us estimate the 3D-2D correspondences using the SIFT feature vectors. The mean SIFT is only computed for those points which are the part of compressed cover model. Since mean SIFT computation of each point is independent of other points, it can be easily parallelizable. We distribute this computation over 20 compute cores.

4.1.1.3 Pair-wise Probabilities Matrix

The visibility probabilities between all pairs of these points are precomputed as an $n \times n$ matrix with the entry (i, j) storing the number of cameras that see points i and j. We store this highly sparse matrix in a compressed row format. In this matrix, we store the number of cameras in common that see any two points. That is, the entry $A[i, j]$ contains the number of cameras that see both X_i and X_j. For each point, we find all the points that share atleast a camera in common with it and store the point ids as well as the number of cameras in common. Here also the computation of each point is independent of other points, so it is also distributed over 20 compute cores.

4.1.1.4 Visual word based Word to Point list

Given the SfM dataset, we pre-compute the visual-word based sorted list mapping each word to a set of 3D points [64]. This list is used in the process of seed generation. Each 3D point is represented using the mean SIFT of its track points. The mean SIFT corresponding to a 3D point is quantized to the nearest visual word using a prebuilt vocabulary tree. FLANN library is used to find the nearest neighbor visual word corresponding to the mean SIFT. We get a mapping from each of the mean SIFTs to a visual word. Therefore, we have a mapping from each of the 3D point to a visual word. Using these mappings, we find a list of all the 3D points corresponding to a visual word.

4.1.2 Query Image Localization

Given all the precomputed datastructures, we compute the camera pose of a given query image using visibility probability guided matching. Given the query image, SIFT is used to compute its feature descriptors. Each feature descriptor is then quantized to nearest word using the vocabulary tree. Seed point is estimated as explained in the seed generation process. The SIFT vectors of the query image are stored in a Kd-Tree and all the nearest neighbor queries for matching 3D-2D point are done using FLANN library[51]. Given sufficient matches we use DLT inside a RANSAC loop to estimate the camera pose.

The proposed localization algorithm can be divided into three basic parts:

- Seed generation: Seed generation attempts to find a seed point to initiate the matching process.
- Probability-guided point matching: Point matching quickly identifies 3D points that match the query image using the visibility probability to guide the search.
- Camera estimation: In this phase, we recover the camera parameters from the matches obtained. Direct linear transform (DLT) is used within a RANSAC loop to estimate the paramters robustly.

Figure 4.4 shows the flowchart of the complete localization pipeline. In the following sections, we explain each of the above steps in detail.

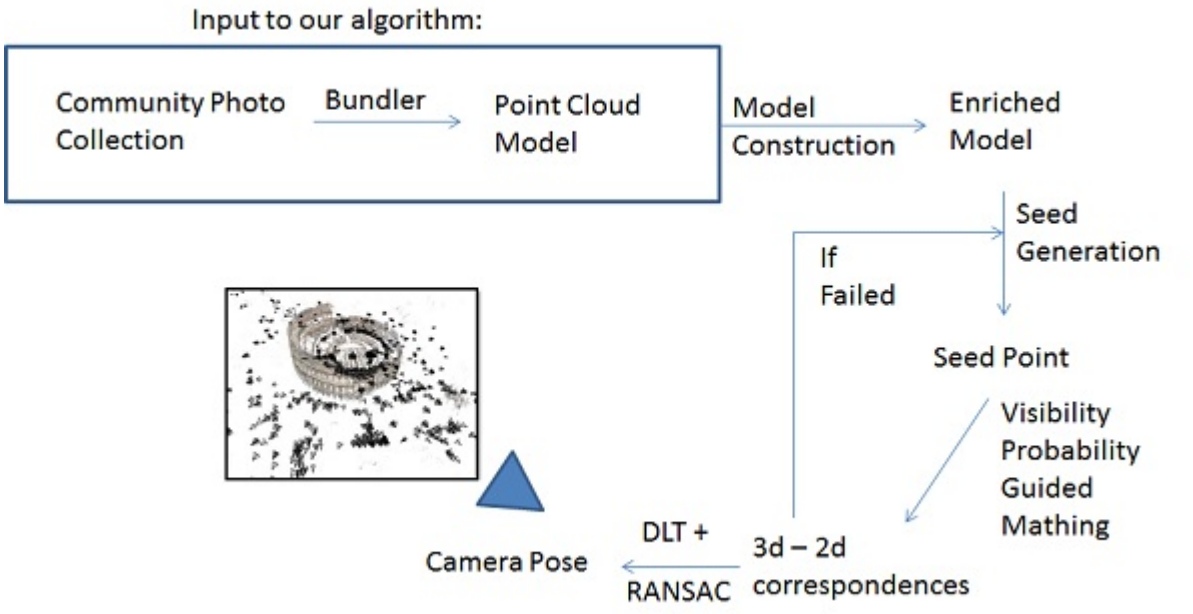


Figure 4.4: Flowchart of the steps involved in the image localization algorithm

4.2 Seed Generation

Localization starts with a query image I and no information about its camera location. We use SIFT (Scale Invariant Feature Transform) to compute the feature descriptors of the query image [46]. SIFT feature descriptors are robust to affine transformations like rotation, scale, shear and translation. SIFT descriptors can effectively handle large variations in images which is quite common in community photo collections. Therefore, SIFT is the preferred descriptor for the purpose of large scale 3D reconstructions.

We adapt the 2D-to-3D searching method used by Sattler et al. to identify a seed point in I [64]. The SfM dataset consists of points and the track information about all the 2D image projections triangulating to that 3D point. Each track point is represented using its SIFT descriptor. The 3D point is represented using the mean SIFT features of its track. Using a standard vocabulary tree of 100K words, we quantize each 3D point to a visual word. Points that map to the same visual word are collected into a list. In our implementation, we pre-compute this list which maps a visual word to a set of 3D points.

SIFT features extracted from the query image are also quantized to visual words. The lists of points for the visual words in the query image are collected. Using the mapping from each visual word to a set of 3D points and the mapping from each SIFT feature vector in the query image to a visual word, we find out a mapping from each SIFT feature vector in the query image to the set of mean SIFTs corresponding to a collection of 3D points. Therefore, we get a mapping from image features to the list of points that quantize to the same visual word. We have a count of the number of 3D points in the list corresponding to a feature descriptor. The list of feature to 3D points mapping is arranged in decreasing order of their

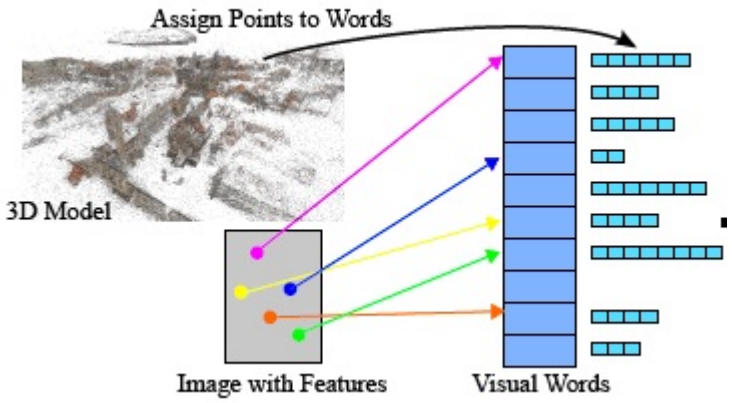


Figure 4.5: Seed Generation Process

lengths (length refers to the number of 3D points mapping to a feture descriptor). Figure 4.5 depicts this process of list generation.

Each query image feature is searched linearly in these lists. The search starts with the longest list to prefer more popular visual words over less popular ones. Linear search is done in the list of points by finding the nearest and the next nearest neighbor point in the list corresponding to the image feature in the SIFT space. Using the SIFT descriptor corresponding to a image feature, we compute distance from the 3D points by measuring the distance of SIFT descriptor of an image feature to mean SIFT feature descriptor of the 3D point. We iterate linearly over the set of 3D points corresponding to an image feature and keep updating the nearest neighbor and the next nearest neighbor by computing the distance of SIFT feature descriptor of an image feature from mean SIFT feature descriptor of a 3D point. The computation time to find the nearest neighbor and next nearest neighbor corresponding to a image feature in the 3D point space is $O(k)$, where k is the size of the list. This computation is fast as the size of the 3D points list mapping from an image feature is generally small in size. A feature matches a point if it clears the ratio test in the list and is considered a potential seed. Ratio test based matching is based on finding the ratio of distance of image feature descriptor from the mean SIFT of nearest neighbor and distance of image feature descriptor from the mean SIFT of the next nearest neighbor. If this $ratio < 0.6$, we declare the nearest neighbor 3D point as a potential seed. Reverse matching of a potential seed using the ratio test in the descriptor space in I confirms it as a seed. We use the mean SIFT of the potential seed and find its nearest and next nearest neighbor among the SIFT feature descriptors in the image. We use FLANN library to estimate the SIFT neighbors in the image corresponding to the mean SIFT [51]. If the ratio of distance from the mean SIFT of 3D point to the nearest neighbor SIFT descriptor in the image and the distance from mean SIFT of 3D point to the next nearest neighbor SIFT descriptor in the image is less than 0.6, we declare that 3D point as the seed point. The seed point is the first 3D-2D match in I.

Sattler et al. search from the shortest list to exploit uniqueness [64]. The descending order of search gives us more reliable potential seeds in practice. We find that the average number of reverse matching steps needed before finding a seed was 14 when searched in the descending order compared to 230 when search was in the ascending order. This can be because the reliability of ratio test based matching is higher given large number of samples as compared to less number of samples. Sattler et al. did not use reverse matching for confirmation and relied on RANSAC to remove the outliers. Our seed generation can be described as a 2D-3D-2D matching approach. Given the seed point, we initiate probability-guided point matching to find rest of the 3D-2D matches. In the following section, we describe this in detail.

4.3 Probability-Guided Point Matching

Probability-guided point matching is used to guide the rest of the matching algorithm, given a seed point. We use visibility probability to guide the point matching algorithm. Algorithm 1 outlines our matching algorithm. It starts with a set S of points that are already matched in the query image I. Initial S contains a singleton seed point. The process involves 3 steps:

1. Identify a candidate set G of points with promise,
2. Prioritize them based on parameters like their independence of S, distance, etc.,
3. Search in the query image based on the priority to add to S.

These steps are repeated until a minimum number of points are matched. The process restarts with a new seed otherwise. It fails after trying a certain number of seeds. Figure 4.7 shows the results for a few steps.

4.3.1 Identifying the Candidate Set

In the normal mode, the candidate set G is the subset of SfM points that are jointly visible to all points in S (Step 4). We include a point X_j into G if the product $\prod_{X_i \in S} p(X_j, X_i)$ is above a threshold t_0. Mathematically, G is defined as,

$$G = \{X_j \mid \prod_{X_i \in S} p(X_j, X_i) > t_0\}. \tag{4.1}$$

A very small threshold t_0 is used to include more points. G can be computed from the precomputed probabilities. It confines the search to local set of points which are jointly visible to all the matched points. The candidate set is large when S has a single point and shrinks fast as points are added to S. The ideal condition for this technique to work perfectly is estimating the candidate set in popular regions or densely photographed areas.

1: Compute SIFT interest points and descriptors of the query image I
2: Find a seed point and set it as set S.
   If number of seeds generated exceeds `maxSeed`, declare failure and exit
3: **while** S contains fewer than t_1 points **do**
4: Update the *candidate set* G of points using joint visibility to points in S
5: If $|S| < t_2$, set $p(X_i|S)D(X_i, S)$ as the priority of points $X_i \in G$
6: Otherwise, set $(1 - p(X_i|S))\, D(X_i, S)$ as the priority.
7: Search each $X_i \in G$ in priority order by finding 2 closest descriptors in I
8: If distances to the descriptors satisfy the ratio test, add X_i to S
9: If no more points in G, proceed to Step 11
10: **end while**
11: If $|S| < t_2$, discard S, G and start over from Step 2
12: If $|S| < t_1$, repeat from Step 3 in the expansion mode.
   Start over from Step 2 if t_1 points can't be found.
13: Estimate camera pose using RANSAC over DLT
14: If number of inliers is less than 12, start over from Step 2
15: Declare success and return camera matrix

Algorithm 1: Localize (Image **I**)

In rare cases, an expansion mode is used with G as points X_j with the sum $\sum_{X_i \in S} p(X_j, X_i)$ above a threshold t_0. We define it as,

$$G = \{X_j \mid \sum_{X_i \in S} p(X_j, X_i) > t_0\} \tag{4.2}$$

G then contains points that are jointly visible with any point in S. This happens when G gets empty with a few promising matches in S (Step 12). This happens in sparsely connected regions which are not so photographed and therefore have less number of 3D points. In these cases, we try to control the rate of expansion by selecting points which are already jointly visible with already matched points. Selecting points which are jointly visible with very less number of already matched points can grow size of the candidate set at an exponential rate.

4.3.2 Prioritizing the Candidate Set

The priority assigned to points in G should depend on the potential contribution to the task. Points independent of the already matched ones can contribute more information. Since $p(X_i|S)$ gives the dependence of X_i on points in S, a priority of $1 - p(X_i|S)$ will consider independent points early (Step 7). Distant matching points contribute more to the linear estimation of camera matrix than proximate ones. Independent and distant points can be preferred with $(1 - p(X_i|S))D(X_i, S)$ as the priority to points $X_i \in G$ in Step 5, where $D()$ a distance measure. Therefore, we define the priority as,

$$Pr(X_i|S) = (1 - p(X_i|S)) \times D(X_i, S) \tag{4.3}$$

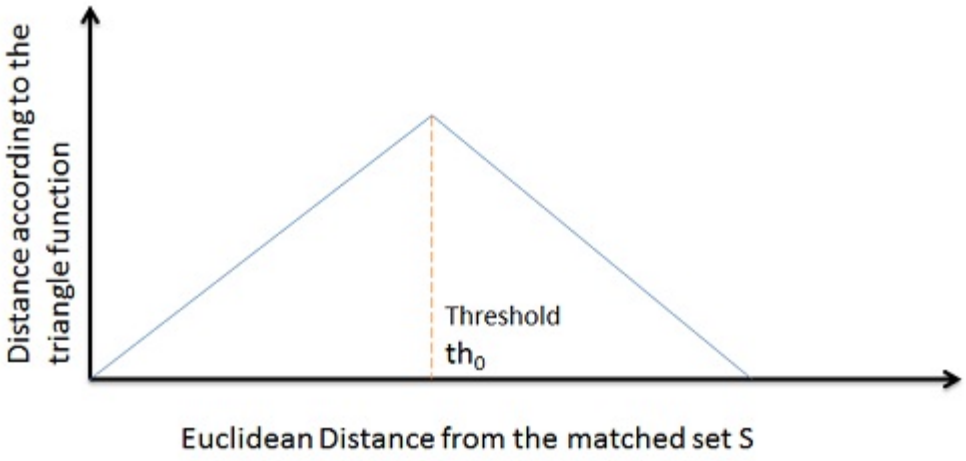


Figure 4.6: Triangle function for a given threshold

We use a triangle function of the Euclidean distance of X_i to the closest point in S as $D(X_i, S)$. The function increases linearly with distance upto a certain threshold and falls steeply beyond it, emphasizing distant points while de-emphasizing very far off ones. We use the mean distances in the SfM model to set the threshold. Mathematically, we define distance as,

$$D(X_i, S) = \min_{X_j \in S} d(X_i, X_j) \tag{4.4}$$

where $d(X_i, X_j)$ is the euclidean distance between the 3D coordinates of X_i and X_j. Using this definition and threshold th_0, the triangle function is defined as

$$D(X_i, S) = \begin{cases} D(X_i, S), & \text{if } D(X_i, S) < th_0. \\ 2 \times th_0 - D(X_i, S), & \text{if } D(X_i, S) \geq th_0. \end{cases} \tag{4.5}$$

Figure 4.6 depicts the triangle function for a given threshold th_0.

In practice, the same localization error is obtained using about half the number of matches if distance is used for priority. As each matched point is added to S, the candidate set shrinks (Step 4) and the priorities of points in it are updated.

Considering independent points can quickly shrink the candidate set and we can have no more points left to search to reach the minimum number of required match threshold. In these cases, we start the process in a safe mode. A safe mode that matches points close to S is used in the initial phase when only a few points are in S. A priority measure of $p(X_i|S)D(X_i, S)$ facilitates this by considering dependent points early.

$$Pr(X_i|S) = p(X_i|S) \times D(X_i, S) \tag{4.6}$$

This continues until t_2 points are matched (Step 6). We use 5 as the value for t_2 in the present system. If the matching cannot produce t_2 points, the seed selection is faulty and we restart the process (Step 11).

After the threshold t_2 is reached, we use $(1 - p(X_i|S))D(X_i, S)$ as the priority to points until t_1 points are matched. Expansion mode is used if fewer than t_1 matches are obtained, which happens less than 10% of the time. In expansion mode, we populate the set G such that it contains points X_j with the sum $\sum_{X_i \in S} p(X_j, X_i)$ above a threshold t_0. In this phase, priority measure of $p(X_i|S)D(X_i, S)$ is used until we have t_1 points matched or no more point is left. We consider the dependent points to slow down the rate of expansion. Considering independent points would increase the size of the candidate set G at an exponential rate. A new seed is tried if these efforts cannot produce sufficient matches (Step 12).

4.3.3 Searching the Candidate Set

Points are considered in the descending order of priority for matching in I (Step 7). A point matches if its closest two neighbors in the descriptor space satisfies the ratio test [46]. If the ratio of distance from mean SIFT of 3D point to the nearest neighbor SIFT descriptor in the image and the distance from mean SIFT of 3D point to the next nearest neighbor SIFT descriptor in the image is less than 0.6, we declare that 3D point as the matching point. We proceed to a RANSAC-based camera estimation after finding t_1 matches. Since community photo collections contain large amount of noise and can lead to wrong matches, we use RANSAC to robustly estimate the inliers. RANSAC divides the set of matches into two categories, inliers which consists of good matches and outliers which are bad matches. Table 4.1 shows the performance as t_1 is increased. The high RANSAC inlier percentage shows that our guided matching scheme produces inliers early.

The threshold t_1 controls the matching effort and the accuracy of localization. Table 4.1 shows that generating 20 to 30 points provides a good balance between accuracy and effort. The matching methods by Li et al. [42] and Sattler et al. [64] need to produce 100 or more potential matches before the camera estimation step. Our method also uses far fewer nearest-neighbor queries as shown later. If RANSAC produces fewer than 12 inliers, the seed is discarded (Step 14). If the number of seeds tried exceeds a limit, the localization of the query image fails. Around 88% of the query images in the Dubrovnik dataset are successfully localized using the first seed. The average number of seeds tried was 1.3 for successful query images and 5 for unsuccessful ones (Table 4.3).

4.4 Camera Estimation

We estimate the camera parameters using RANSAC (RAndom SAmple Consensus) over DLT (Direct Linear Transform) using the 3D-2D correspondences in the matching process. RANSAC tries to remove outliers by assuming that the inilers can be explained using a model where as outliers are independent of each other and do not follow any model. It runs by randomly selecting 6 points to estimate camera parameters using DLT and estimating the number of inliers that are within the error threshold of the

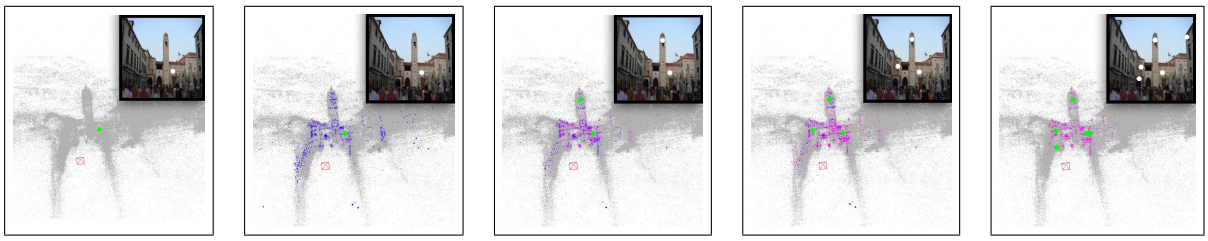


Figure 4.7: Steps shown over the point cloud. Query image is the inset. From left: Seed (green) and ground truth camera, candidate set G with priority (increasing from blue to red), matched point and new G with priority, prioritized G after 3 matches, and after 5 matches. Please view the magnified electronic version.

No. points in S (t1)	15	20	30	50	100
Inlier percentage	79.9	78.12	76.15	75.32	66.76
No. images registered	727	788	789	789	789
No. points searched	523	839	1679	2893	4597
Error in localization using RANSAC					
Median	0.0037	0.0023	0.0023	0.0022	0.0021
70th percentile	0.011	0.0073	0.0058	0.0063	0.006
Mean	0.1177	0.0267	0.0498	0.0261	0.0512
Error in localization without RANSAC					
70th percentile	0.0231	0.0243	0.0263	0.0345	0.0425

Table 4.1: Performance for different number of point matches for Dubrovnik dataset

model. After some iteration, the model which had maximum number of inliers is considered as the desired model and is re-evaluated using all the inliers.

Given the inliers, direct linear transform (DLT) is used to estimate the camera projection matrix [29]. Using the 3D-2D correspondences, direct linear transform projects the optimization problem as a linear least squares problem. The projection equation is represented by $\mathbf{x} \simeq \mathbf{PX}$ where $\mathbf{X}$ is a vector of 3D points, $\mathbf{x}$ is the corresponding vector of 2D features and $\mathbf{P}$ is the estimated projection matrix mapping 3D points to the corresponding 2D features. $\mathbf{P}$ is estimated by solving the equation $\mathbf{x} \times \mathbf{PX} = \mathbf{0}$ using singular value decomposition. It requires a minimum of six correspondences to estimate the projection matrix using DLT. Our approach generates reliable matches, with over 78% of the matches being RANSAC inliers. About 20 matches are sufficient in most cases, but an accuracy-speed tradeoff is provided by adjusting t_1 as seen in Table 4.1. RANSAC itself has only a small impact on the localization accuracy, which is the primary strength of our approach. We use the basic p6p method for camera estimation [29]. It requires minimum 6 points to estimate the camera pose. DLT uses singular value

Dataset	#Cameras	#Points		#Query Images
		Full	Compressed	
Dubrovnik	6044	2,208,645	75,000	800
Rome	15,179	4,312,820	100,000	1000
Vienna	1,324	1,132,745	50,000	266

Table 4.2: Details of the datasets used for the experiments

decomposition to estimate the projection matrix which is further decomposed to get translation and rotation.

Other techniques like p4pfr and p3p do not improve the accuracy by much, but require higher computation time. p4pfr estimates camera pose using 4 points and unknown focal length and radial distortion [33]. It solves for camera pose, focal length and radial distortion. The system of polynomial equations is solved using Grobner basis methods. Similarly, if focal length and other intrinsic parameters are known we can use p3p method which requires only three points to estimate camera parameters [18]. It gives 4 camera matrices of which the camera matrix which results in minimum reprojection error is chosen as the result.

4.5 Experimental Results

We use the Dubrovnik (6044 images, 2M points), Rome (15179 images, 4M points), and Vienna (1324 images, 1M points) datasets used by others for our experiments [42, 64]. The number of points used for experiments were compressed to 75K, 100K and 50K points respectively. Information on the datasets used for the experiments is given in Table 4.2. We use the same query images (800 for Dubrovnik, 1000 for Rome and 266 for Vienna) to facilitate direct comparison.

Table 4.3 gives the matching performance of our approach. We show the number of registered images out of all query images, mean RANSAC inlier ratio, number of images that can be registered using one seed, average number of seeds required to register an image, average number of 3D-2D nearest neighbor queries done, average registeration time for all the registered images. Table 4.3 also gives the statistics of average number of seeds required to reject an image, average number of 3D-2D nearest neighbor queries done and the average rejection time. We match more images than previous efforts and do so with lesser effort in terms of the number of nearest neighbor queries [64, 42]. We also generate promising matches early as seen from the RANSAC inlier ratio. On an average, we are able to match 4 images per second. Our rejection time is also comparable to the registration time. We register 88% of the images using the first seed itself.

Table 4.4 compares the localization accuracy of our approach with prior work on the Dubrovnik dataset. We achieve better localization compared to Li et al. [42]. The mean error seems worse because we match more images than them, with most of them being of low quality for matching. Sattler et al.

Successfully registered images						Rejected images		
# of Images	Inlier Ratio (Mean)	With 1 Seed	Mean # Seeds	Mean # NN queries	Reg. Time	Mean # Seeds	Mean # NN queries	Rej. Time
788	0.78	692	1.3	839.6	0.25	5.01	4199	0.51
977	0.81	755	2.74	3253	0.27	6.21	5876	0.61
219	0.74	144	2.25	972	0.40	4.23	3369	0.49

Table 4.3: Performance of our localization approach for registered and rejected images in Dubrovnik, Rome and Vienna datasets (from top to bottom)

Method	#Reg. Images	Mean[m]	Median[m]	1st Quartile[m]	3rd Quartile[m]
Our Method	788	34.79	3.104	0.88	11.83
Sattler et al.	784	53.9	1.4	0.4	5.9
Li et al.	753	18.3	9.3	7.5	13.4

Table 4.4: Comparison of our localization accuracy with previous work

[64] perform better on median error, but has very high mean error due to several images localized very badly. This is true even though they register fewer images.

To study the rejection performance of the approach, images from Rome dataset were searched in Dubrovnik and Vienna datasets and images from Dubrovnik in Rome. Table 4.5 shows the results. The candidate set gets exhausted fast and the search terminates quickly in our approach. Rejection involves trying more nearest-neighbor queries than successful registration. Images were rejected using 2-4 seed points and took on an average 0.4, 0.54 and 0.22 seconds respectively. Our rejection times are only slightly more than registration times. This is in contrast to rejection being typically an order of magnitude slower for the previous methods [42, 64]. Rejecting images from other datasets is slightly faster than rejecting images from the same dataset. This is because images from other datasets require less number of nearest neighbor queries than the number of nearest neighbor queries required to reject images from the same dataset. For example, in order to reject images from the other dataset when localized to the Dubrovnik dataset requires 3451 NN queries on an average whereas to reject images from the same dataset when localized to the Dubrovnik dataset requires 4199 NN queries on an average.

Our localizing performance is compared with methods by Li et al. [42] and Sattler et al. [64] in Table 4.6. We show the statistics of the number of images registered, the number of 3D points searched and the registeration time for three datasets of Dubrovnik, Rome and Vienna and compared it with the corresponding numbers by Li et al. [42] and Sattler et al. [64].

Dataset	#Query Images	#Rej. Images	#Seeds	#NN Queries	Rej. Time[s]
Dubrovnik	1000	1000	2.8	3451	0.4
Rome	800	800	2.70	5026	0.54
Vienna	1000	1000	3.44	2810	0.22

Table 4.5: Performance on negative examples from other datasets

Figure 4.8: Six registered (left) and four rejected (right) new images of Dubrovnik

Li et al. use a prioritized selection of points for matching [42]. They use a small set of very high priority points to increase the chance of matching instead of a separate seed generation step. They update the priorities of jointly visible points when a new point is matched by a factor inversely proportional to its degree. Our probability structure allows better pruning, prioritizing and reordering of the remaining points than the above heuristic. Our method successfully registers more images in each data set while searching far fewer points. We search 10-20% points of what was done by Li et al (Table 4.6). and get better localization accuracy (Table 4.4) as a result.

Sattler et al. [64] extend the 2D-3D matching until a large number of matches are found, with static priorities for points. While they perform well on images that localized well, they performs much more work on rejection. Our rejection and registration times are comparable (Table 4.5). The visibility probability is thus successful in guiding the search to the successful points and in rejecting matches.

The performance of localizing new images of the same space is given in Table 4.7. We collected general images from Flickr using relevant keywords for Dubrovnik, Rome, and Vienna. These images were taken in 2011 or later. They registered quickly and produced low reprojection errors. Six registered and four rejected images of Dubrovnik are shown in Figure 4.8.

We compared the performance of various prioritization techniques in the localization process. Table 4.8, 4.9 and 4.10 show the results on Dubrovnik, Rome and Vienna datasets. The prioritization measures used in comparison were Dependence Measure ($p(X|S)$), Independence Measure ($(1-p(X|S))$), Distance weighted Dependence Measure ($p(X|S)D(X|S)$) and Distance weighted Independence Measure ($(1-p(X|S))D(X|S)$) and the combined method as described in Section 4.2. Table 4.11 shows the results on monument scale datasets of Colosseum, Pantheon and St. Peters Basilica.

Method	**Dubrovnik**			**Rome**			**Vienna**		
	#reg. images	#pts searched	reg. time[s]	#reg. images	#pts searched	reg. time[s]	#reg. images	#pts searched	reg. time[s]
Our	789	839.6	0.25	977	3253	0.27	219	972	0.40
[42]	784	-	0.28	975	-	0.25	207	-	0.46
[64]	753	9756	0.73	921	12963	0.91	204	6245	0.55

Table 4.6: Performance comparison with earlier methods on 3 datasets

Dataset	#Query Images	#Reg. Images	#Seeds	#NN Queries	RANSAC Inlier Ratio	Reproj. Error	Reg. Time[s]
Dubrovnik	70	64	2.81	950	0.80	1.25	0.21
Rome	44	40	3.1	3834	0.78	1.7	0.3
Vienna	40	27	2.59	2381	0.81	0.5	0.36

Table 4.7: Registration and rejection performance on new images

Formulation	Successfully registered images							Rejected images		
	# of Images	Inlier Ratio (Mean)	With 1 Seed	Mean # Seeds	Mean # NN queries	Loc. Acc (Mean)	Reg. Time	Mean # Seeds	Mean # NN queries	Rej. Time
Dependence Measure	787	0.77	660	1.44	822	91.6	0.23	5	4305.5	0.40
Independence Measure	790	0.73	615	1.78	1181.1	47.10	0.24	4.2	3185.9	0.29
Distance weighted Dependence Measure	787	0.766	659	1.45	938	82.06	0.27	5	4327.9	0.45
Distance weighted Independence Measure	786	0.729	621	1.76	1323	42.32	0.254	4.14	3858.6	0.42
Combined Method (Section 4.2)	788	0.78	692	1.3	839.6	34.79	0.25	5.01	4199	0.51

Table 4.8: Performance of various probability formulations on images in **Dubrovnik** Dataset

4.6 Conclusions

In this chapter, we propose an image localization system which uses the visibility probability framework to guide correspondence search. It uses a few 3D points to 2D feature mappings in the query image to estimate its camera. We use the conditional visibility probability among points to effectively guide these searches. Our probability-guided exploration generates RANSAC inliers in high proportions as matching proceeds. Our localization method successfully registers as many or more new images as the prior work. We reduce the number of nearest neighbor searches to 10-20% of what was done by Li et al [42]. Due to the improved accuracy, we only need to generate 25% of matches as Sattler et al [64].

Formulation	Successfully registered images						Rejected images		
	# of Images	Inlier Ratio (Mean)	With 1 Seed	Mean # Seeds	Mean # NN queries	Reg. Time	Mean # Seeds	Mean # NN queries	Rej. Time
Dependence Measure	968	0.78	618	2.54	3707.01	0.40	5.43	9700	0.75
Independence Measure	972	0.75	559	2.85	3606.5	0.34	3.71	8436.6	0.7
Distance weighted Dependence Measure	974	0.76	617	2.55	3685.5	0.35	4.5	5.61	0.71
Distance weighted Independence Measure	976	0.78	539	2.94	3647	0.31	3.7	7475	0.65
Combined Method (Section 4.2)	977	0.81	755	2.74	3253	0.27	6.21	5876	0.61

Table 4.9: Performance of various probability formulations on images in **Rome** Dataset

Formulation	Successfully registered images						Rejected images		
	# of Images	Inlier Ratio (Mean)	With 1 Seed	Mean # Seeds	Mean # NN queries	Reg. Time	Mean # Seeds	Mean # NN queries	Rej. Time
Dependence Measure	216	0.760	142	2.34	1892.5	0.422	4.08	3325	0.455
Independence Measure	225	0.74	135	2.75	2223.33	0.426	4.12	3050.9	0.436
Distance weighted Dependence Measure	218	0.754	144	2.22	1424.7	0.429	4.14	3324.5	0.464
Distance weighted Independence Measure	218	0.75	144	2.22	1917.7	0.424	4.14	3324.5	0.45
Combined Method (Section 4.2)	219	0.74	144	2.25	972	0.40	4.23	3369	0.49

Table 4.10: Performance of various probability formulations on images in **Vienna** Dataset

Dataset	#Query Images	#Reg. Images	#Seeds (mean)	#NN Queries (mean)	Inlier Ratio	Reproj. Error	Reg. Time[s]
Colosseum (1167 Cameras, 483621 Pts)	180	176	1.36	362.6	0.87	0.92	0.183
Pantheon (1809 Cameras, 301397 Pts)	200	199	1.15	393	0.848	1.17	0.2
St. Peters Basilica (1180 Cameras, 318362 Pts)	200	182	1.07	444.9	0.87	0.81	0.23

Table 4.11: Registration and rejection performance on Colosseum, Pantheon and St. Peters Basilica

Chapter 5

Feature Triangulation

We define the problem of feature triangulation as follows: Given a feature-point in an image with 2D coordinates and the descriptor, find its 3D coordinates. Triangulation is an important problem in computer vision. In the context of SfM duality, it is a dual problem of image localization, with the roles of cameras and points reversed. Each camera sees multiple points and each point is seen by multiple cameras in a SfM dataset. Each image that is localized is matched to many 3D points and similarly, each feature that is triangulated has to be matched to many other features. Similar to image localization, an image feature which has to be triangulated is matched to nearby images.

Triangulating a 2D feature to a 3D point requires a track of features that matches to the query 2D feature and the corresponding camera poses. Given a track of 2D features and the corresponding camera poses, a 3D point is estimated such that it minimizes the reprojection error. Generally, a 3D point is estimated using a simple method which is later optimized using a global optimization technique like bundle adjustment. By assuming that the feature is taken from the space of the SfM dataset, we can match it in two or more cameras. Triangulation can give the point coordinates after matching in cameras (whose calibration matrices are known). Overall the feature triangulation algorithm can be divided into two basic steps:

- Feature Matching: Given a query 2D feature, we have to search the nearby cameras to find the matching 2D features. As a result, a track of matching features is estimated in this step.
- Point Estimation: Given a track of 2D features and the corresponding camera poses, a 3D point is estimated such that it minimizes the reprojection error when projected back onto the respective cameras.

However, there usually are at least two orders of magnitude more features than there are images. Hence, it will be more useful to triangulate a number of features found in a given image. We address the problem of triangulating features that are found in an image of the SfM dataset.

Given a 2D feature, our algorithm searches the space of cameras for visibility of the query feature guided using visibility information. We use the conditional visibility probability among cameras to effectively guide these searches. Our scheme can match 4 new images per second on a high end work-

station. Our feature triangulation method increases the point density at typical locations by 30-50%. It is especially good at identifying points in regions with repetitive appearance, where the 2D-2D match of SfM performs poorly. Our feature triangulation algorithm is also the dual of the localization process described in the previous chapter. It has five steps:

- Feature-Point selection: This step is used to find novel feature-points for triangulation. This ensures that we do not resonstruct an already triangulated feature and add novel 3D points to the triangulation.

- Probability-Guided Camera Selection: Our algorithm searches the space of cameras for visibility of the query feature guided using visibility information. We use the conditional visibility probability among cameras to effectively guide these searches.

- Local-View Selection Prior: We incorporate the local information around the query feature and use this prior to boost the performance of probability-guided camera selection

- Two-way matching: This matching technique uses two-way epipolar constraint on the SIFT nearest neighbors of a feature point to confirm a match. This can also work in the case of repetitive structures.

- Point Estimation: Given the track of matching features, we estimate the 3D coordinates using singular value decomposition to solve the equation $\mathbf{x} \times \mathbf{PX} = \mathbf{0}$. It is done inside a RANSAC loop to make it more robust

We now explain each one of these in detail.

5.1 Feature-Point Selection

Given an already registered image, we use feature-point selection to select novel feature-points which are matched to nearby images and reconstruct new 3D points. This step identifies the 2D feature-points in an image to be triangulated. We use a camera-image C_0 from the SfM dataset as the image from which features are selected. We detect interest points in the specified image and describe them using SIFT vectors. We restrict the features to lie sufficiently far from the image borders. Alternately, the user can interactively draw a rectangle to restrict the features inside it. We also discard interest points that are within 4 pixels of an existing point present in the image. This is done by finding all the features which are already reconstructed and has a 2D-3D correspondence. These reconstructed points are a part of SfM output and reconstructing them again won't add novel 3D points. So, we set a threshold of 4 pixels on the euclidean distance from the already reconstructed feature points. This ensures we use novel features-points for triangulation. A camera and a 2D feature-point in it are sent to the triangulation process.

1: Initialize S as the camera C_0
2: **while** S contains fewer than q_1 cameras **do**
3: Update the *candidate set* G of cameras using visibility to cameras in S
4: Assign $(1 - p(C_i|S))D(C_i, S)\alpha(C_i, S)$ as the priority to cameras $C_i \in G$
5: Examine camera $C_i \in G$ in priority order. Match F in it.
6: If C_i satisfies two-way matching, add it to S
7: If no more cameras in G, proceed to Step 9
8: **end while**
9: **if** $|S| \geq 2$ **then**
10: Triangulate camera projections to estimate a 3d point
11: **else**
12: Declare Failure
13: **end if**

Algorithm 2: Feature Triangulation (Camera C_0, Feature F)

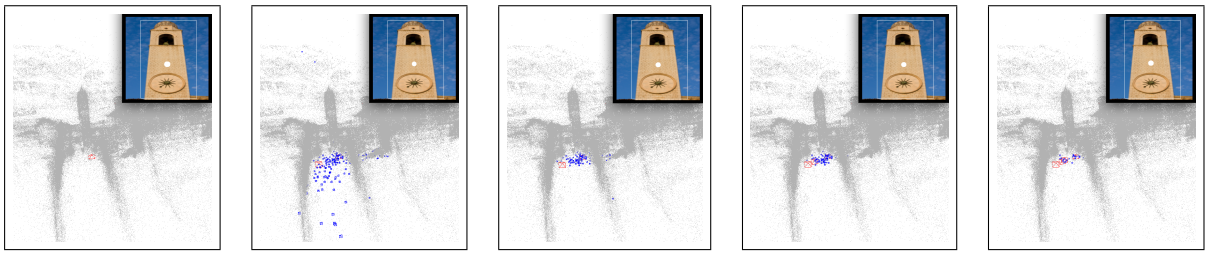


Figure 5.1: Steps shown over the point cloud with chosen camera and query feature as inset. From left: First camera, candidate set G (blue), next matched camera and new G, prioritized G after 3 matches, and after 4 matches. Please view in the electronic version.

5.2 Probability-Guided Camera Selection

Our algorithm searches the space of cameras for visibility of the query feature guided using visibility information. Algorithm 2 outlines our camera selection process. It uses the set S of selected cameras in which the query feature F is matched. S starts with camera C_0 in which feature F is present. The process involves 3 steps:

1. Identify a candidate set G of cameras with promise,

2. Prioritize them based on parameters like their independence of S, distance, etc.,

3. Search in the cameras for the query feature-point F based on the priority to add to S.

These steps are repeated until a minimum number of feature-points are matched. Triangulation fails if F matches in less than 2 cameras.

5.2.1 Identifying the Candidate Set

Cameras that are jointly visible to every camera in S are selected as the candidate set G (Step 3). We include cameras C_j for which $\prod_{C_i \in S} p(C_j, C_i)$ is greater than a threshold in G. Mathematically, we define G as,

$$G = \{C_j \mid \prod_{C_i \in S} p(C_j, C_i) > t_0\} \tag{5.1}$$

G can be computed from the precomputed probabilities. It confines the search to local set of cameras which are jointly visible to all the matched cameras. The candidate set is large when S has a single camera and shrinks with each camera added to S. The ideal condition for this technique to work perfectly is estimating the candidate set in areas having high density of 3D points.

5.2.2 Prioritizing the Candidate Set

The priority assigned to cameras in G should depend on the potential contribution to the task. Cameras of G that are most independent of S are likely to contribute most to point triangulation. Since $p(C_i|S)$ gives the dependence of C_i on cameras in S, a priority of $1 - p(C_i|S)$ will consider independent cameras early (Step 4). Therefore, using $1 - p(C_i|S)$ as the priority for each camera $C_i \in G$ will contribute most to point triangulation.

Cameras far from those in S and making larger angles with them are likely to be more useful in triangulation. We include a distance measure $D(C_i, S)$ and an angle measure $\alpha(C_i, S)$ in the priority. These are defined as follows:

- $D(C_i, S)$ is a triangle function based on the distance of C_i to the closest camera in S. The function increases linearly with distance upto a certain threshold and falls steeply beyond it, emphasizing distant cameras while de-emphasizing very far off ones. We use the mean distances in the SfM model to set the threshold. Mathematically, we define distance as,

  $$D(C_i, S) = \min_{C_j \in S} d(C_i, C_j) \tag{5.2}$$

  where $d(C_i, C_j)$ is the euclidean distance between the camera centers of C_i and C_j. Using this definition and threshold th_0, the triangle function is defined as

  $$D(C_i, S) = \begin{cases} D(C_i, S), & \text{if } D(C_i, S) < th_0. \\ 2 \times th_0 - D(C_i, S), & \text{if } D(C_i, S) \geq th_0. \end{cases} \tag{5.3}$$

- $\alpha(C_i, S)$ is a triangle function of the minimum of average angle of points common in C_i with cameras in S. Mathematically, we define it as,

  $$\alpha(C_i, S) = \min_{C_j \in S} \beta(C_i, C_j) \tag{5.4}$$

$$\beta(C_i, C_j) = \frac{\sum_{X_i \in F_{C_i} \cap F_{C_j}} ang(X_i, C_i, C_j)}{\text{Number of 3D points seen in both } C_i \text{ and } C_j} \tag{5.5}$$

where F_x is the set of 3D points observed in camera C_i. $ang(X_i, C_i, C_j)$ is the angle between the lines of sight from X_i to its projections in C_i and C_j. Figure 4.6 depicts the triangle function for a given threshold th_0.

Using the visibility probability, distance and angle, the overall priority of C_i is given as **(Step 4)** :

$$P(C_i, S) = (1 - p(C_i|S))D(C_i, S)\alpha(C_i, S) \tag{5.6}$$

As each matched camera is added to S, the candidate set shrinks (Step 3) and the priorities of camera in it are updated. Considering independent cameras can quickly shrink the candidate set.

5.2.3 Searching the Candidate Set

The cameras are selected in their priority order to match the query feature F in them. We use a two-way matching process (Step 5). Two-way matching is used to match a query feature to other features in the prioritized camera. As each camera is added to S, the candidate set shrinks (Step 3) and the priorities of cameras change. The guided camera selection continues while there are cameras left in G or when q_1 cameras are in S. Values of q_1 between 7 and 10 perform well in practice. Triangulation fails if F matches in less than 2 cameras. Figure 5.1 shows results of a few steps of the process.

5.3 Local-View Selection Prior

In order to boost the performance of the probability guided camera selection, we multiply the overall camera priority with a local view selection prior. Local view selection prior is dependent on the feature point which is being triangulated. To estimate this value, we find 10 already triangulated nearest neigbhors of the feature point. Each of the nearest neighbor point corresponds to a 3D point which is visible in some cameras. The set of all 3D points corresponding to the nearest neighobr feature points is known as L. So, the local view prior of a camera is computed as the number of 3D points belonging to L that is visible in the respective camera. Mathematically, we formulate the local view selection prior of a camera as,

$$lv(C_i) = \sum_{X_j \in L} k(C_i, X_j) \tag{5.7}$$

where $k(C_i, X_j)$ is 1 if X_j is visible in C_i otherwise zero. The prioritized camera is multiplied by $1 + lv(C_i)$ to obtain a locally relevant priority order. It is given as

$$P(C_i, S) = (1 + lv(C_i)) \times ((1 - p(C_i|S))D(C_i, S)\alpha(C_i, S)) \tag{5.8}$$

Local view selection prior is based on the hypothesis that if a camera sees most of the 3D points corresponding to the nearest neighbors of a feature point, it is highly probable that the new feature point

will be visible in that camera. The priority of a camera is unchanged if it sees zero number of points corresponding to the nearest neigbhors of the query feature point.

In the following section we describe our two way matching approach.

5.4 Two-Way Matching

To match F in C_i, we find the 10 nearest neighbors of F in C_i in the SIFT descriptor space. FLANN library is used to find ten nearest neighbors of query feature SIFT point from the set of features corresponding to C_i [51]. Since we know the camera poses of the two images (C_0 and C_i), we estimate the epipolar line of feature F corresponding to C_0 in C_i [29]. The feature-point p among the ten nearest neighbors, closest to the epipolar line of F in C_i is considered a potential match if its epipolar distance in C_i is less than 4 pixels.

We then find the SIFT nearest neighbor of the potential match p in camera C_0. If its nearest neighbor is the feature F, we declare C_i as a matching camera and feature p as the matching feature-point. Figure 5.2 shows an example of two-way matching technique. This method finds matches of even repetitive feature-points, which are rejected by the traditional ratio test. Finding 3D points among difficult repetitive structures is the primary advantage of our feature triangulation approach.

5.5 Point Estimation

The 3D coordinates of the query feature F are computed by triangulating its projections in the cameras in S. Given a track of 2D features and the corresponding camera poses, a 3D point is estimated such that it minimzes the reprojection error. Reprojection error is the difference between actual feature location and projected feature location, when the 3D point is projected back into the image using camera parameters. Euclidean distance is used to measure the reprojection error. For multiview triangulation, we minimize the sum of reprojection errors across all images. We define the sum of reprojection errors across all images as,

$$Reprojection\ Error(C_i, X_0) = \sum_{C_i \in S} d(Q(C_i, X_0), x_{ij})^2 \tag{5.9}$$

where $Q(C_i, X_0)$ is the predicted projection of the reconstructed point X_0 on camera C_i and $d(x, y)$ the Euclidean distance between the inhomogeneous image points represented by x and y. We perform the summation over all the cameras that belong to the matched set S.

Similar to the problem of localization where given the 2D and 3D correspondences, we estimate the camera projection matrix, here we estimate the 3D coordinate of a point given its projection coordinates and the corresponding projection matrices. More formally, we have 2D coordinates $\mathbf{x}$ and the projection matrices $\mathbf{P}$ available, in the projection equation $\mathbf{x} \simeq \mathbf{PX}$. We triangulate the 3D point $\mathbf{X}$ by decomposing the equation $\mathbf{x} \times \mathbf{PX} = \mathbf{0}$ using singular value decomposition.

Figure 5.2: An example of two-way matching technique. Blue square in the left images is the query feature point. Epipolar lines are shown in blue. Red squares in the right image corresponds to the nearest neighbors of the query feature point. Green square represents the matching point using two-way matching.

We use RANSAC over a simple 2-view triangulation for this. RANSAC tries to remove outliers by assuming that the inilers can be explained using a model where as outliers are independent of each other and do not follow any model It runs by randomly selecting 2 points to estimate camera parameters using SVD (as explained above) and estimating the number of inliers that are within the error threshold of the model. After some iteration, the model which had maximum number of inliers is considered as the desired model and is re-evaluated using all the inliers. Robust triangulation is a well studied problem [71]. However, our problem involves highly noisy cameras for which some of these methods aren't suitable. We get adequate accuracy using the simple method, though better techniques should be explored for this step in the future.

5.6 Experimental Results

We carry out an experiment using the Dubrovnik dataset to analyse the performance of our algorithm. We selected 18K random 3D points for triangulation and comparison with the ground truth. We used the feature-point of the points track as the query feature. Table 5.1 and 5.2 give triangulation results,

Track Length	No. pts triangulated	No. images tried	Triangulation error	Reprojection error	Running time [s]
2	8743	219.75	0.0548	0.09498	0.44
3	2187	261.34	0.0095	1.8092	0.59
4	1104	243.76	0.0050	2.0842	0.49
5	791	250.70	0.0059	2.2168	0.63
> 5	5441	181.33	0.0030	2.4178	0.33

Table 5.1: Triangulation statistics for different track lengths for 19K Dubrovnik points

separated by their track lengths and reprojection errors. We show the number of points triangulated, number of images tried, the triangulation error, reprojection error and the running time. Triangulation error is the Euclidean distance between original point from the SfM dataset and point generated by our method. Table 5.1 shows that points with longer tracks are triangulated better. Better triangulated points involve larger variance in viewing angles and show a larger reprojection error (Table 5.2). That is, low reprojection error is not indicative of better triangulation.

Reprojection Error	No. pts triangulated	No. images Tried	Triangulation error	Reprojection error	Running time [s]
< 1 pixels	6037	238.23	0.0409	0.49	0.52
< 2 pixels	12186	223.73	0.0373	0.9927	0.47
< 3 pixels	15850	218.44	0.0288	1.3315	0.45

Table 5.2: Triangulation statistics for different reprojection errors for 19K points

Table 5.3 compares the results of two-way matching and matching using ratio test on 9K points from Dubrovnik. It also shows adding new points on images from the dataset with repetitive structures shown in Figure 5.3. Ratio test based matching is done by taking the ratio of nearest neighbor and the nearest neighbor of the query SIFT feature and using a threshold of 0.6 on the ratio to declare it as a match. Two way matching is performed as explained earlier. Both of these methods worked well on the SfM points because these points perhaps passed the ratio test in the original SfM process.

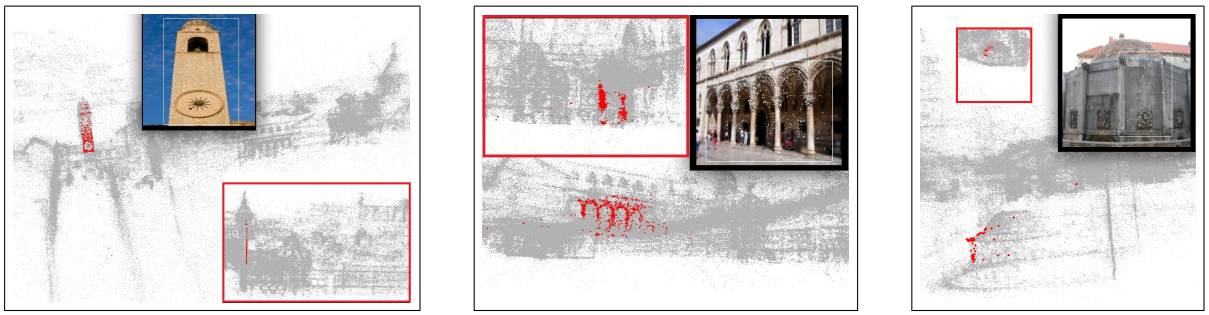


Figure 5.3: Triangulated points shown in red from different view points. Side view is shown with red borders. The starting image is the inset

To find 2D-2D correspondences in the case of repetitive structure is a challenging problem. In the case of repetitive structures the SIFT feature descriptors of similar looking entities are very near in SIFT space. Normal ratio test based matching fails in these cases because the distance of the query feature from the nearest neighbor and the next nearest neighbor feature is almost the same because of their closeness in the SIFT space. Since the state of art SfM reconstruction pipeline is based of ratio test based matching, these are difficult images for the original SfM process and had fewer points in them. Our proposed matching technique using the two-way test adds 10 to 50 times the number of points on images with repetitive structures. The point density at these locations also increase by 33-50%. Image 1 (Fig. 5.4, left) had 1946 points to which 669 new points were added. Image 2 (Fig. 5.4, center) had 1747 triangulated points to which 686 new points are added. Image 3 (Fig. 5.4, right) had 187 triangulated points to which 120 new points were added.

Dataset	Two Way Matching					Using Ratio Test				
	#Pts added	#Images Tried	Tri. error	Reproj. error	time [s]	#Pts added	#Images Tried	Tri. error	Reproj. error	time [s]
9126 Pts	8830	267.36	0.0105	1.40	0.728	8950	270.29	0.0097	1.7	0.69
Image 1	669	99	-	1.16	0.38	14	99	-	1.54	0.28
Image 2	686	95	-	1.11	0.70	60	124	-	1.14	0.65
Image 3	120	90	-	1.17	0.75	4	91	-	1.07	0.78

Table 5.3: Effect of Different Matching techniques on Repetitive Structures

Table 5.4 compares the triangulation performance using different prioritization formulation. We use 30K random points of Dubrovnik dataset for triangulation and comparison with ground truth. The prioritization measures used in comparison were Dependence Measure ($p(X|S)$), Independence Measure ($(1 - p(X|S))$), Distance weighted Dependence Measure ($p(X|S)D(X|S)$), Distance weighted Independence Measure ($(1 - p(X|S))D(X|S)$), Distance and Angle weighted Dependence measure and Distance and Angle weighted Independence measure (Section 5). For each on of these, we provide statistics on the average number of points triangulated, average number of images tried, average triangulation error, average reprojection error and the average running time.

Table 5.5 gives results of adding new points to an image with repetitive structure. Two-way matching technique successfully adds new points to difficult images with repetitive structures. Figure 5.4 shows the accuracy of reconstructed points from different views.

5.7 Conclusions

Feature triangulation identifies a few cameras in which the query feature is visible for triangulation. It searches the space of cameras for visibility of the query feature. We use the conditional visibility probability among cameras to effectively guide these searches. Our scheme can match 4 new images per second on a high end workstation. Our feature triangulation method increases the point density

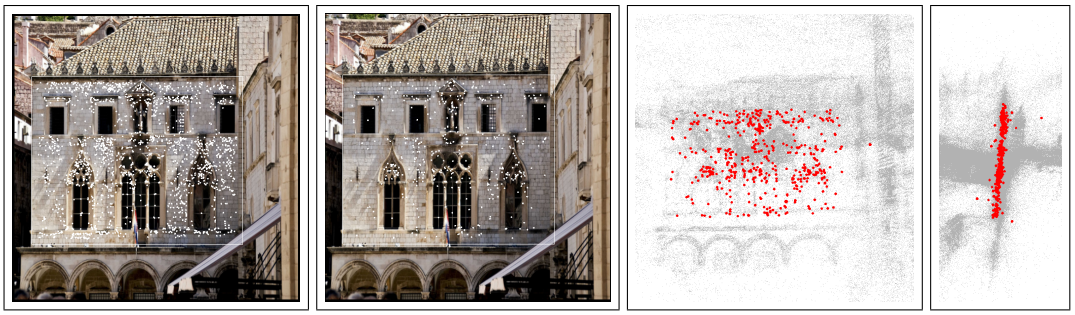
Formulation	No. pts triangulated	No. images tried (mean)	Triangulation error	Reprojection error	Running time [s]
Dependence Measure	18844	191.5	0.0576	1.91	0.84
Independence Measure	16533	439.4	0.0928	1.28	1.04
Distance weighted Dependence Measure	22657	244.8	0.0176	1.80	0.74
Distance weighted Independence Measure	20996	442.65	0.075	1.36	1.09
Distance and Angle weighted Dependence Measure	23197	290.40	0.0210	1.71	0.85
Distance and Angle weighted Independence Measure (Section 5)	25478	177.66	0.0141	1.93	0.65

Table 5.4: Triangulation statistics using different formulation for 30K Dubrovnik points

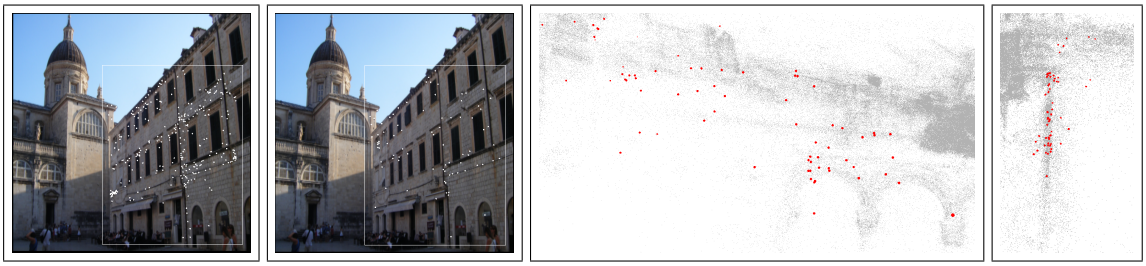
Image	#Pts already triangulated	#Pts added	#Images Tried	Reproj. error	time [s]
Image 1	1961	501	120.2	1.83	0.71
Image 2	343	85	168.4	1.25	0.85
Image 3	3328	347	220.9	1.39	0.89

Table 5.5: Additional Results of using Two-Way matching on images with Repetitive Structures

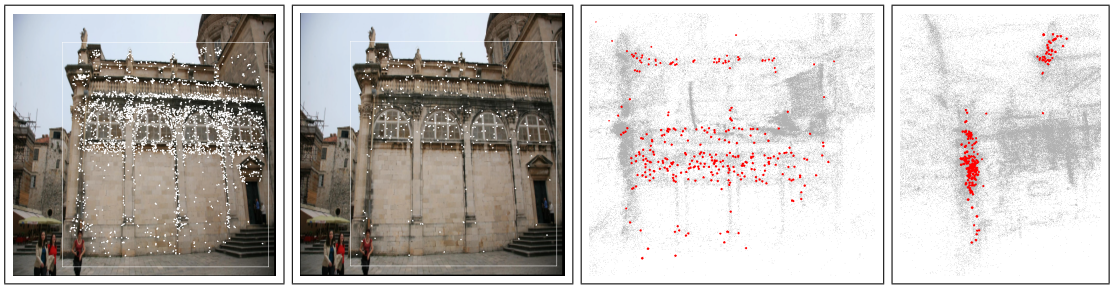
at typical locations by 30-50%. It is especially good at identifying points in regions with repetitive appearance, where the 2D-2D match of SfM performs poorly.



(a) Image 1



(b) Image 2



(c) Image 3

Figure 5.4: Results of adding new points to an image with repetitive structure. From left: Image with original triangulated points (white), same image with newly added points (white), triangulated points shown in red from front view and side view of the same is shown.

Chapter 6

Bundle Adjustment

Bundle Adjustment (BA) refers to the optimal adjustment of bundles of rays that leave 3D feature points onto each camera centres with respect to both camera positions and point coordinates. It produces jointly optimal 3D structure and viewing parameters by minimizing the cost function for a model fitting error [45, 82]. The re-projection error between the observed and the predicted image points, which is expressed for m images and n points as,

$$Error(P, X) = \sum_{i=1}^{n} \sum_{j=1}^{m} d(Q(P_j, X_i), x_{ij})^2 \tag{6.1}$$

where $Q(P_j, X_i)$ is the predicted projection of point i on image j and $d(x, y)$ the Euclidean distance between the inhomogeneous image points represented by x and y. Using the reprojection error, the cost function is defined as,

$$Cost(P, X) = \min_{P,X} Error(P, X) \tag{6.2}$$

Bundle Adjustment is carried out using the Levenberg-Marquardt algorithm [44, 59] because of its effective damping strategy to converge quickly from a wide range of initial guesses. Given the parameter vector $\mathbf{p}$, the functional relation f, and measured vector $\mathbf{x}$, it is required to find δ_p to minimize the quantity $\|x - f(\mathbf{p} + \delta_p)\|$. Assuming the function to be linear in the neighborhood of p, this leads to the equation

$$(\mathbf{J^T J} + \mu \mathbf{I})\delta_p = \mathbf{J^T}\epsilon \tag{6.3}$$

where J is the Jacobian matrix $J = \frac{\partial \mathbf{x}}{\partial \mathbf{p}}$. LM Algorithm performs iterative minimization by adjusting the damping term μ[54], which assure a reduction in the error ϵ.

BA can be cast as non-linear minimization problem as follows [45, 82]. A parameter vector $\mathbf{P} \in \mathbf{R}^M$ is defined by the m projection matrices and the n 3D points, as

$$\mathbf{P} = (\mathbf{a}_1^T, \ldots, \mathbf{a}_m^T, \mathbf{b}_1^T, \ldots, \mathbf{b}_n^T)^T, \tag{6.4}$$

where $\mathbf{a}_j$ is the j^{th} camera parameters and $\mathbf{b}_i$ is the i^{th} 3D point coordinates. A measurement vector $\mathbf{X}$ is the measured image coordinates in all cameras:

$$\mathbf{X} = (\mathbf{x}_{11}^T, \ldots, \mathbf{x}_{1m}^T, \mathbf{x}_{21}^T, \ldots, \mathbf{x}_{2m}^T, \ldots, \mathbf{x}_{n1}^T, \ldots, \mathbf{x}_{nm}^T)^T. \tag{6.5}$$

The estimated measurement vector $\hat{\mathbf{X}}$ using a functional relation $\hat{\mathbf{X}} = f(\mathbf{P})$ is given by

$$\hat{\mathbf{X}} = (\hat{\mathbf{x}}_{11}^T, \ldots, \hat{\mathbf{x}}_{1m}^T, \hat{\mathbf{x}}_{21}^T, \ldots, \hat{\mathbf{x}}_{2m}^T, \ldots, \hat{\mathbf{x}}_{n1}^T, \ldots, \hat{\mathbf{x}}_{nm}^T)^T, \tag{6.6}$$

with $\hat{\mathbf{x}}_{ij} = \mathbf{Q}(\mathbf{a_j}, \mathbf{b_i})$ [82]. BA minimizes the squared Mahalanobis distance $\epsilon^T \Sigma_x^{-1} \epsilon$, where $\epsilon = \mathbf{X} - \hat{\mathbf{X}}$, over $\mathbf{P}$. Using LM Algorithm, we get the normal equation as

$$(\mathbf{J^T \Sigma_X^{-1} J} + \mu \mathbf{I})\delta = \mathbf{J^T \Sigma_X^{-1}} \epsilon. \tag{6.7}$$

Apart from the notations above, mnp denotes the number of measurement parameters, cnp the number of camera parameters and pnp the number of point parameters. The total number of projections onto cameras is denoted by nnz, which is the length of vector $\mathbf{X}$.

The solution to Equation 6.7 has a cubic time complexity in the number of parameters and is not practical when the number of cameras and points are high. The Jacobian matrix for BA, however has a sparse block structure. Sparse BA uses a sparse variant of the LM Algorithm [45]. It takes as input the parameter vector **P**, a function **Q** used to compute the predicted projections $\hat{\mathbf{x}}_{ij}$, the observed projections $\mathbf{x}_{ij}$ from i^{th} point on the j^{th} image and damping term μ for LM and returns as an output the solution δ to the normal equation as given in Equation 6.7. Algorithm 3 outlines the SBA and indicates the steps that are mapped onto the GPU. All the computations are performed using double precision arithmetic to gain accuracy.

6.1 Data Structure for the Sparse Bundle Adjustment

Since most of the 3D points are not visible in all cameras, we need a visibility mask to represent the visibility of points onto cameras. Visibility mask is a boolean mask built such that the $(i, j)^{th}$ location is true if i^{th} point is visible in the j^{th} image. We propose to divide the reconstruction consisting of cameras and 3D points into camera tiles or sets of 3D points visible in a camera. Since the number of cameras is less than number of 3D points and bundle of light rays projecting on a camera can be processed independent of other cameras, this division can be easily mapped into blocks and threads on fine grained parallel machines like GPU. The visibility mask is sparse in nature since 3D points are visible in nearby cameras only and not all. We compress the visibility mask using Compressed Column Storage (CCS) [16]. Figure 6.1 shows a visibility mask for 4 cameras and 4 points and its Compressed Column Storage. We do not store the val array as in standard CCS [16] as it is same as the array index in 3D point indices array. The space required to store this is $(nnz + m) \times 4$ bytes whereas to store the whole visibility matrix is $m \times n$ bytes. Since the projections $\hat{\mathbf{x}}_{ij}, \mathbf{x}_{ij}$ and the Jacobian $\mathbf{A}_{ij}, \mathbf{B}_{ij}$ is non zero only when the i^{th} 3D point is visible in the j^{th} camera, it is also sparse in nature and thereby stored in contiguous locations using CCS which is indexed through the visibility mask.

1: Compute the Predicted Projections $\hat{x}_{ij}$ using $\mathbf{P}$ and $\mathbf{Q}$. # Computed on GPU

2: Compute the error vectors $\epsilon_{ij} \leftarrow x_{ij} - \hat{x}_{ij}$ # Computed on GPU

3: Assign $\mathbf{J} \leftarrow \frac{\partial \mathbf{X}}{\partial \mathbf{P}}$ (Jacobian Matrix) where
$\mathbf{A}ij \leftarrow \frac{\partial \hat{x}_{ij}}{\partial a_j} = \frac{\partial \mathbf{Q}(a_j, b_i)}{\partial a_j}$ ($\frac{\partial \hat{x}_{ij}}{\partial a_k} = 0 \, \forall i \neq k$) and
$\mathbf{B}ij \leftarrow \frac{\partial \hat{x}_{ij}}{\partial b_i} = \frac{\partial \mathbf{Q}(a_j, b_i)}{\partial b_i}$ ($\frac{\partial \hat{x}_{ij}}{\partial b_k} = 0 \, \forall j \neq k$) # Computed on GPU

4: Assign $\mathbf{J^T \Sigma_X^{-1} J} \leftarrow \begin{pmatrix} \mathbf{U} & \mathbf{W} \\ \mathbf{W^T} & \mathbf{V} \end{pmatrix}$ where U,V,W is given as
$\mathbf{U}_j \leftarrow \sum_i \mathbf{A}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{A}_{ij}$, $\mathbf{V}_i \leftarrow \sum_j \mathbf{B}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$ and
$\mathbf{W}_{ij} \leftarrow \mathbf{A}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$ # Computed on GPU

5: Compute $\mathbf{J^T \Sigma_X^{-1}} \epsilon$ as $\epsilon_{a_j} \leftarrow \sum_i \mathbf{A}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \epsilon_{ij}$,
$\epsilon_{b_i} \leftarrow \sum_j \mathbf{B}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \epsilon_{ij}$ # Computed on CPU

6: Augment $\mathbf{U}_j$ and $\mathbf{V}_i$ by adding μ to diagonals to yield
$\mathbf{U}_j^*$ and $\mathbf{V}_i^*$ # Computed on GPU

7: Normal Equation: $\begin{pmatrix} \mathbf{U}^* & \mathbf{W} \\ \mathbf{W^T} & \mathbf{V}^* \end{pmatrix} \binom{\delta_a}{\delta_b} = \binom{\epsilon_a}{\epsilon_b}$ # Using Equation (6.7)

8: $\begin{pmatrix} \underbrace{\mathbf{U}^* - \mathbf{W}\mathbf{V}^{*-1}\mathbf{W}^T}_{\mathbf{S}} & 0 \\ \mathbf{W}^T & \mathbf{V}^* \end{pmatrix} \binom{\delta_a}{\delta_b} = \binom{\underbrace{\epsilon_a - \mathbf{W}\mathbf{V}^{*-1}\epsilon_b}_{\mathbf{e}}}{\epsilon_b}$ # Using Schur Complement

9: Compute $\mathbf{Y}_{ij} \leftarrow \mathbf{W}_{ij} \mathbf{V}_i^{*-1}$ # Computed on GPU

10: Compute $\mathbf{S}_{jk} \leftarrow \mathbf{U}_j^* - \sum_i \mathbf{Y}_{ij} \mathbf{W}_{ik}^T$ # Computed on GPU

11: Compute $e_j \leftarrow \epsilon_{a_j} - \sum_i \mathbf{Y}_{ij} \epsilon_{b_i}$ # Computed on CPU

12: Compute δ_a as $(\delta_{a_1}^T, \ldots, \delta_{a_m}^T)^T = \mathbf{S}^{-1}(e_1^T, \ldots, e_m^T)^T$ # Computed on GPU

13: Compute $\delta_{b_i} \leftarrow \mathbf{V}_i^{*-1}(\epsilon_{b_i} - \sum_j \mathbf{W}_{ij}^T \delta_{a_j})$ # Computed on GPU

14: Form δ as $(\delta_a^T, \delta_b^T)^T$

Algorithm 3: SBA ($\mathbf{P}, \mathbf{Q}, x, \mu$)

6.2 Computation of the Initial Projection and Error Vector

Given **P** and **Q** as input, the initial projection is calculated as $\hat{\mathbf{X}} = \mathbf{Q(P)}$ (Algorithm 3,line 1) where $\hat{\mathbf{X}}$ is the estimated measurement vector and $\hat{\mathbf{x}}_{ij} = \mathbf{Q}(\mathbf{a_j}, \mathbf{b_i})$ is the projection of point b_i on the camera a_j using the function $\mathbf{Q}$. The error vector is calculated as $\epsilon_{ij} = \mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij}$ where $\mathbf{x}_{ij}$ and $\hat{\mathbf{x}}_{ij}$ are the measured and estimated projections. The estimated projections and error vectors consumes memory space of $nnz \times mnp$ each. Our implementation consists of m thread blocks running in parallel, with each thread of block j computing a projection to the camera j. The number of threads per block is limited by the total number of registers available per block and a maximum limit of number of threads per block. Since the typical number of points seen by a camera is of the order of thousands (more than the limit on threads) we loop over all the 3D points visible by a camera in order to compute projections. The GPU kernel to calculate the initial projection and error vector is shown in Algorithm 4.

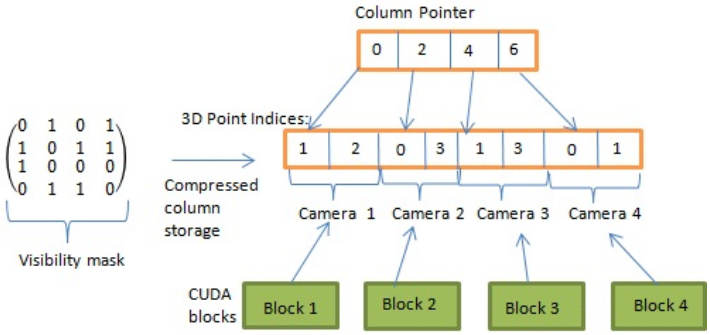


Figure 6.1: An example of the compressed column storage of visibility mask having 4 cameras and 4 3D Points. Each CUDA Block processes one set of 3D points.

1: CameraID $\leftarrow$ BlockID
2: Load the camera parameters into shared memory
3: **repeat**
4: Load the 3D point parameters (given ThreadID and CameraID)
5: Calculate the Projection $\hat{x}_{ij}$ given 3D Point i and Camera j
6: Calculate the Error Vector using $\epsilon_{ij} = \mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij}$
7: Store the Projections and Error Vector back into global memory
8: **until** all the projections are calculated

Algorithm 4: CUDA_INITPROJ_KERNEL ($\mathbf{P}, \mathbf{Q}, \mathbf{X}$)

6.3 Computation of the Jacobian Matrix (J)

The Jacobian matrix is calculated as $\mathbf{J} = \frac{\partial \mathbf{X}}{\partial \mathbf{P}}$ (Algorithm 3, line 3). For $\hat{\mathbf{X}} = (\hat{\mathbf{x}}_{11}^T, \ldots, \hat{\mathbf{x}}_{n1}^T, \hat{\mathbf{x}}_{12}^T, \ldots, \hat{\mathbf{x}}_{n2}^T, \ldots, \hat{\mathbf{x}}_{1m}^T, \ldots, \hat{\mathbf{x}}_{nm}^T)^T$, the Jacobian would be $(\frac{\partial \hat{\mathbf{x}}_{11}}{\partial \mathbf{P}}^T, \ldots, \frac{\partial \hat{\mathbf{x}}_{n1}}{\partial \mathbf{P}}^T, \frac{\partial \hat{\mathbf{x}}_{12}}{\partial \mathbf{P}}^T, \ldots, \frac{\partial \hat{\mathbf{x}}_{n2}}{\partial \mathbf{P}}^T, \ldots, \frac{\partial \hat{\mathbf{x}}_{1m}}{\partial \mathbf{P}}^T, \ldots, \frac{\partial \hat{\mathbf{x}}_{nm}}{\partial \mathbf{P}}^T)$. Since $\frac{\partial \hat{x}_{ij}}{\partial a_k} = 0 \, \forall i \neq k$ and $\frac{\partial \hat{x}_{ij}}{\partial b_k} = 0 \, \forall j \neq k$, the matrix is sparse in nature.

For the example, shown in Figure 6.1, the Jacobian matrix would be

$$J = \begin{pmatrix} A_{10} & 0 & 0 & 0 & 0 & B_{10} & 0 & 0 \\ A_{20} & 0 & 0 & 0 & 0 & 0 & B_{20} & 0 \\ 0 & A_{01} & 0 & 0 & B_{01} & 0 & 0 & 0 \\ 0 & A_{31} & 0 & 0 & 0 & 0 & 0 & B_{31} \\ 0 & 0 & A_{12} & 0 & 0 & B_{12} & 0 & 0 \\ 0 & 0 & A_{32} & 0 & 0 & 0 & 0 & B_{32} \\ 0 & 0 & 0 & A_{03} & B_{03} & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{13} & 0 & B_{13} & 0 & 0 \end{pmatrix}, \tag{6.8}$$

where, $\mathbf{A}ij = \frac{\partial \hat{x}_{ij}}{\partial a_j} = \frac{\partial \mathbf{Q}(a_j,b_i)}{\partial a_j}$ and $\mathbf{B}ij = \frac{\partial \hat{x}_{ij}}{\partial b_i} = \frac{\partial \mathbf{Q}(a_j,b_i)}{\partial b_i}$. The matrix when stored in compressed format would be
$\mathbf{J} = (A_{10}, B_{10}, A_{20}, B_{20}, A_{01}, B_{01}, A_{31}, B_{31}, A_{12}, B_{12}, A_{32}, B_{32}, A_{03}, B_{03}, A_{13}, B_{13})$. The memory required is $(cnp+pnp) \times mnp \times nnz \times 4$ bytes. The CUDA grid structure used in Jacobian computation is similar to initial projection computation. Block j processes the A_{ij} and B_{ij}, corresponding to the j^{th} camera. The kernel to calculate the Jacobian Matrix is shown in Algorithm 5.

1: CameraID ← BlockID
2: **repeat**
3: Load the 3D point parameters and Camera parameters (given ThreadID and CameraID) into thread memory.
4: Calculate B_{ij} followed by A_{ij} using scalable finite differentiation
5: Store the A_{ij} and B_{ij} into global memory at contiguous locations.
6: **until** all the projections are calculated

Algorithm 5: CUDA_JACOBIAN_KERNEL ($\mathbf{P}, \mathbf{Q}$)

6.4 Computation of $J^T \Sigma_X^{-1} J$

$\mathbf{J^T \Sigma_X^{-1} J}$ is given as $\begin{pmatrix} \mathbf{U} & \mathbf{W} \\ \mathbf{W^T} & \mathbf{V} \end{pmatrix}$ where $\mathbf{U}_j = \sum_i \mathbf{A}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{A}_{ij}$, $\mathbf{V}_i = \sum_j \mathbf{B}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$ and $\mathbf{W}_{ij} = \mathbf{A}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$. For the example in Figure 6.1, $\mathbf{J^T \Sigma_X^{-1} J}$ is given as:

$$\mathbf{J^T \Sigma_X^{-1} J} = \begin{pmatrix} U_0 & 0 & 0 & 0 & 0 & W_{10} & W_{20} & 0 \\ 0 & U_1 & 0 & 0 & W_{01} & 0 & 0 & W_{31} \\ 0 & 0 & U_2 & 0 & 0 & W_{12} & 0 & W_{32} \\ 0 & 0 & 0 & U_3 & W_{03} & W_{13} & 0 & 0 \\ 0 & W_{01}^T & 0 & W_{03}^T & V_0 & 0 & 0 & 0 \\ W_{10}^T & 0 & W_{12}^T & W_{13}^T & 0 & V_1 & 0 & 0 \\ W_{20}^T & 0 & 0 & 0 & 0 & 0 & V_2 & 0 \\ 0 & W_{31}^T & W_{32}^T & 0 & 0 & 0 & 0 & V_3 \end{pmatrix} \tag{6.9}$$

6.4.1 Computation of U:

The CUDA grid structure consists m blocks, such that each block processes U_j where j is the BlockID. Thread i in block j processes $\mathbf{A}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{A}_{ij}$, which is stored in the appropriate segment. The summation is faster when using a segmented scan[70] on Tesla S1070 whereas a shared memory reduction is faster on the Fermi GPU. The memory space required to store $\mathbf{U}$ is $cnp \times cnp \times m \times 4$ bytes. The computation of $\mathbf{U}$ is done as described in Algorithm 6.

1: CameraID ← BlockID
2: **repeat**
3: Load A_{ij} where j = CameraID (for a given thread)
4: Calculate $A_{ij} \times A_{ij}^T$ and store into appropriate global memory segment
5: **until** all the A_{ij} are calculated for the j_{th} camera
6: Perform a shared memory reduction to get final sum on Fermi. Write to global memory and perform a segmented scan on Tesla S1070.

Algorithm 6: CUDA_U_KERNEL (**A**)

6.4.2 Computation of V:

The CUDA grid structure and computation of V is similar to the computation of U. The basic difference between the two is that $\mathbf{B}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$ is stored in the segment for point i for reduction using segmented scan on Tesla S1070 where as a shared memory reduction is done on Fermi. The memory space required to store **V** is $pnp \times pnp \times n \times 4$ bytes.

6.4.3 Computation of W:

The computation of each W_{ij} is independent of all other W_{ij} as there is no summation involved as in **U** and **V**. Therefore the computation load is equally divided among all blocks in GPU. $\lceil \frac{nnz}{10} \rceil$ thread blocks are launched with each block processing 10 W matrices. This block configuration gave us the maximum CUDA occupancy. The memory space required to store **W** is $pnp \times cnp \times nnz \times 4$ bytes. The computation of **W** is done as described in Algorithm 7.

1: Load A_{ij} and B_{ij} for each warp of threads.
2: Calculate $A_{ij} \times B_{ij}^T$
3: Store W_{ij} back into global memory at appropriate location.

Algorithm 7: CUDA_W_KERNEL (**A**, **B**)

6.5 Computation of $\mathbf{S} = \mathbf{U}^* - \mathbf{W}\mathbf{V}^{*-1}\mathbf{W}^T$

The computation of S is the most demanding step of all the modules (Algorithm 3, line 10). Table 6.1 shows the split up of computation time among all components. After calculating U,V and W, augmentation of U,V is done by calling a simple kernel, with m, n blocks with each block adding μ to the respective diagonal elements. Since V^* is a block diagonal matrix, it's inverse can be easily calculated through a kernel with n blocks, with each block calculating the inverse of V^* submatrix (of size $pnp \times pnp$).

6.5.1 Computation of $\mathbf{Y} = \mathbf{WV}^{*-1}$:

The computation of each Y_{ij} is independent of all other Y_{ij} as there is no summation involved as in **U** and **V**. Therefore the computation load is equally divided among all blocks in GPU. $\lceil \frac{nnz}{10} \rceil$ thread blocks are launched with each block processing 10 Y matrices and each warp of thread computing $W_{ij} \times V_i^{*-1}$. This block configuration gave us the maximum CUDA occupancy. The memory space required to store **Y** is $pnp \times cnp \times nnz \times 4$ bytes. The computation of **Y** is done as described in Algorithm 8.

1: Load W_{ij} and V_i^{*-1} for each warp of threads.
2: Calculate $W_{ij} \times V_i^{*-1}$
3: Store Y_{ij} back into global memory at appropriate location.

Algorithm 8: CUDA_Y_KERNEL ($\mathbf{W}, \mathbf{V}^{*-1}$)

6.5.2 Computation of $\mathrm{U}^* - \mathbf{YW}^{\mathbf{T}}$:

S is a symmetric matrix, so we calculate only the upper diagonal. The memory space required to store **S** is $m \times m \times 81 \times 4$ bytes. The CUDA grid structure consists of $m \times m$ blocks. Each block is assigned to a 9×9 submatrix in the upper diagonal, where each block calculates one $S_{ij} = U_{ij} - \sum_k Y_{ki} W_{kj}^T$. Limited by the amount of shared memory available and number of registers available per block, only 320 threads are launched. The algorithm used for computation is given in Algorithm 9.

1: **repeat**
2: for S_{ij}
3: Load 320 3D Point indices (given camera set i) into shared memory
4: Search for loaded indices in camera set j and load them into shared memory.
5: **for all** 320 points loaded in shared memory **do**
6: Load 10 indices of the camera set i and j from the shared memory.
7: For each warp, compute $Y_{ki} W_{kj}^T$ and add to the partial sum for each warp in shared memory
8: **end for**
9: Synchronize Threads
10: **until** all the common 3D points are loaded.
11: Sum up the partial summations in the shared memory to get the final sum.
12: **if** $i == j$ **then**
13: Compute $Y_{ii} W_{ii}^T \leftarrow U_{ii}^* - Y_{ii} W_{ii}^T$
14: **end if**
15: Store $Y_{ij} W_{ij}^T$ into global memory.

Algorithm 9: CUDA_S_KERNEL ($\mathbf{U}^*, \mathbf{Y}, \mathbf{W}^{\mathbf{T}}$)

6.6 Computation of the Inverse of S

As the S Matrix is symmetric and positive definite, Cholesky decomposition is used to perform the inverse operation (Algorithm 3, line 12). Cholesky decomposition is done using the MAGMA library [47], which is highly optimized using the fine and coarse grained parallelism on GPUs as well benefits from hybrids computations by using both CPUs and GPUs. It achieves a peak performance of 282 GFlops for double precision. Since GPU's single precision performance is much higher than it's double precision performance, it used the mixed precision iterative refinement technique, in order to find inverse, which results in a speedup of more than 10 over the CPU.

6.7 Scheduling of Steps on CPU and GPU

Figure 6.2 shows the way CPU and GPU work together, in order to maximize the overall throughput. While the computationally intense left hand side of the equations are calculated on GPU, the relatively lighter right hand side are computed on CPU. The blocks connected by the same vertical line are calculated in parallel on CPU and GPU. The computations on the CPU and the GPU overlap. The communications are also performed asynchronously, to ensure that the GPU doesn't lie idle from the start to the finish of an iteration.

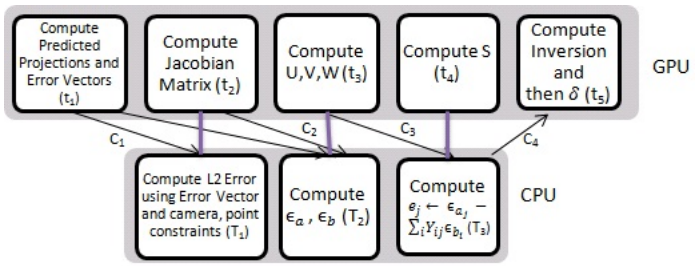


Figure 6.2: Scheduling of steps on CPU and GPU. Arrows indicate data dependency between modules. Modules connected through a vertical line are computed in parallel on CPU and GPU.

6.8 Experimental Results

In this section, we analyze the performance of our approach and compare with the CPU implementation of Bundle Adjustment [45]. We use an Intel Core i7, 2.66GHz CPU. For the GPU, we use a quarter of an Nvidia Tesla S1070 [43] with CUDA 2.2 and an Nvidia Tesla C2050 (Fermi) with CUDA 3.2. All computations were performed in double precision, as single precision computations had correctness issues for this problem.

We used the Notre Dame 715 dataset [76] for our experiments. We ran the 3D reconstruction process on the data set and the input and output parameters ($\mathbf{P}, \mathbf{Q}, x, \mu, \delta$) were extracted and stored for bundle adjustment. We focussed on getting good performance for a dataset of around 500 images as explained before. The redundancy is being exploited for larger data sets using a minimal skeletal subset of similar size by other researchers [4, 78]. We used a 488 image subset to analyze the performance and to compare it with the popular implementation of bundle adjustment [45].

Computation Step	Time Taken (in seconds)									
	GPU1	GPU2	GPU1	GPU2	GPU1	GPU2	GPU1	GPU2	GPU1	GPU2
	38 Cameras		104 Cameras		210 Cameras		356 Cameras		488 Cameras	
Initial Proj	0.02	0.01	0.02	0.03	0.05	0.04	0.06	0.04	0.06	0.05
Jacobian	0.1	0.04	0.2	0.07	0.32	0.12	0.39	0.16	0.45	0.17
U, V, W Mats	0.14	0.04	0.23	0.09	0.39	0.15	0.5	0.18	0.56	0.2
S Matrix	0.25	0.09	0.97	0.27	2.5	0.56	4.63	1.01	6.55	1.3
S Inverse	0.01	0	0.09	0.02	0.28	0.08	0.87	0.19	1.74	0.39
L2 Err (CPU)	0		0.01		0.01		0.01		0.02	
ϵa, ϵb (CPU)	0.05		0.12		0.17		0.21		0.24	
e (CPU)	0.03		0.05		0.08		0.1		0.11	
Total Time	**0.52**	**0.19**	**1.51**	**0.51**	**3.54**	**0.97**	**6.44**	**1.61**	**9.36**	**2.15**

Table 6.1: Time in seconds for each step in one iteration of Bundle Adjustment for different number of cameras on the Notre Dame data set. Total time is the time taken by hybrid implementation of BA using CPU and GPU in parallel. GPU1 is a quarter of Tesla S1070 and GPU2 is Tesla C2050.

Table 6.1 shows the time taken for a single iteration for each major step. The **S** computation takes most of the time, followed by the **S** inverse computation. The Schur complement takes about 70% of the computation time for **S**, as it involves $\mathcal{O}(m^2 \times mnp \times pnp \times cnp \times mnvis)$ operations, where $mnvis$ is the maximum number of 3D points visible by a single camera. On the GPU, each of the m^2 blocks performs $\mathcal{O}(mnp \times pnp \times cnp \times mnvis)$ computations. 60% of **S** computation is to find the partial sums, 30% for the reduction, and 10% for the search operation. It is also limited by the amount of shared memory. The Jacobian computation is highly data parallel and maps nicely to the GPU architecture. Rest of the kernels (U, V, W and initial projection) are light.

As shown in Figure 6.3, the total running time on the GPU is $t = t_1 + t_2 + t_3 + t_4 + C_4 + t_5$ and on CPU is $T = T_1 + C_1 + T_2 + C_2 + T_3 + C_3$ where t_i is the time taken by GPU modules, T_i time taken by CPU modules and C_i communication time. The total time taken is $max(t, T)$. CPU-GPU parallel operations take place only when $max(t, T) < (t + T)$. For the case of 488 cameras, the time taken by GPU completely overlaps the CPU computations and communication, so that there is no idle time for the GPU. Figure 6.4 compares the time taken by our hybrid algorithm for each iteration of Bundle Adjustment with the CPU only implementation on the Notre Dame dataset. The hybrid version with Tesla C2050 gets a speedup of 30-40 times over the CPU implementation.

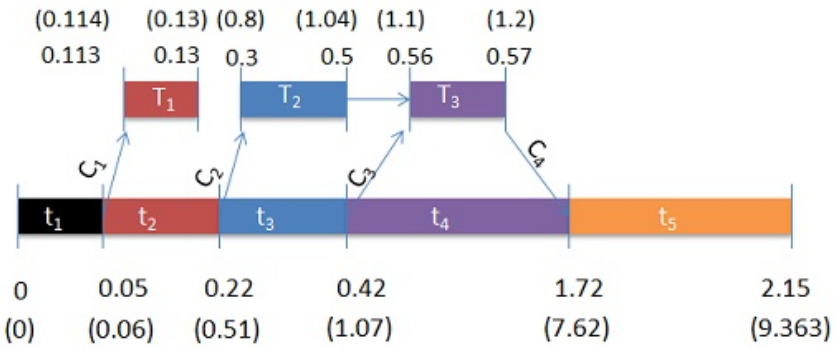


Figure 6.3: Starting and ending times for each step including memory transfer for one iteration using 488 cameras. Times in paranthesis are for the use of the S1070 and others for the C2050.

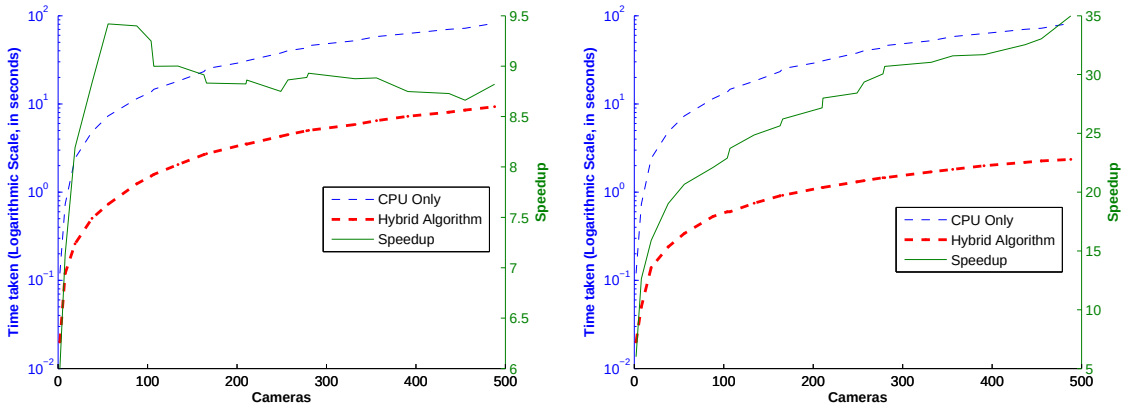
6.8.1 Memory Requirements

The total memory used can be a limiting factor in the scalability of bundle adjustment for large scale 3D reconstruction. As we can see in Figure 6.5, the total memory requirement is high due to temporary requirements in the segmented scan [70] operation on the earlier GPU. The extra memory required is of the size $3 \times nnz \times 81 \times 4$ bytes which is used to store the data, flag and the final output arrays for the segmented scan operation. The permanent memory used to store the permanent arrays such as **J, U, V, W,** and **S** is only a moderate fraction of the total memory required. The Fermi has a larger shared memory and the reduction is performed in the shared memory itself. Thus, the total memory requirement is the same as the permanent memory requirement when using Tesla C2050.

6.9 Conclusions

In this chapter, we introduced a hybrid algorithm using the GPU and the CPU to perform practical time bundle adjustment. The time taken for each iteration for 488 cameras on using our approach is around 2 seconds on Tesla C2050 and 9 seconds on Tesla S1070, compared to 81 seconds on the CPU. This can reduce the computation time of a week on CPU to less than 10 hours. This can make processing larger datasets practical. Most of the computations in our case is limited by the amount of available shared memory, registers and the limit on number of threads. The double precision performance is critical to the GPU computation; the better performance using Fermi GPUs may also be due to this.

Faster bundle adjustment will enable processing of much larger data sets in the future. One option is to explore better utilization of the CPU. Even the single-core CPU is not used fully in our implementation currently. The 4-core and 8-core CPUs that are freely available can do more work, and will need a relook at the distribution of the tasks between the CPU and the GPU. The use of multiple GPUs to increase the available parallelism is another option. Expensive steps like the computation of **S** matrix



(a) Using Tesla S1070

(b) Using Tesla C2050

Figure 6.4: Time and speedup for one iteration of Bundle Adjustment on the CPU using Tesla S1070 and Tesla S2050.

can be split among multiple GPUs without adding enormous communication overheas. This will further change the balance between what can be done on the CPU and on the GPU.

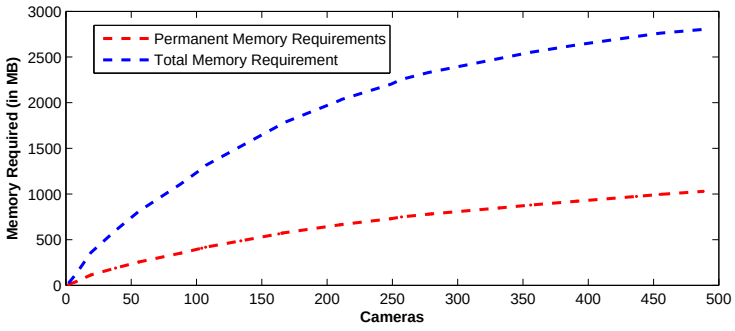


Figure 6.5: Memory required (in MB) on the GPU for different number of cameras.

Chapter 7

Conclusions

In the first part of this thesis, we presented the visibility probability structure of an SfM dataset and its applications in the dual problems of image localization and feature triangulation. The probability-guided search produced fast and efficient point and camera matches for this problem for both registration and rejection. we were able to increase point density at a location by 30-50% using the feature triangulation step, especially in places with repetitive structures.

The high storage requirements for the joint visibility information is a drawback of this work. The formulation depends on the probability structure to guide the matching process. This can have difficulty in sparse regions of space, where the empirical probabilities are unreliable.

In this second part of this thesis, we introduced a hybrid algorithm using the GPU and the CPU to perform practical time bundle adjustment. The time taken for each iteration for 488 cameras on using our approach is around 2 seconds on Tesla C2050 and 9 seconds on Tesla S1070, compared to 81 seconds on the CPU. This can reduce the computation time of a week on CPU to less than 10 hours. This can make processing larger datasets practical. Most of the computations in our case is limited by the amount of available shared memory, registers and the limit on number of threads. The double precision performance is critical to the GPU computation; the better performance using Fermi GPUs may also be due to this.

As a possible future direction, we are working on an alternate SfM pipeline using efficient localization, triangulation and fast bundle adjustment. Given the visibility information available in the reconstructed SfM output, we can quickly localize our image using our proposed image localization algorithm. New points from the localized can be triangulated using the visibility probability guided feature triangulation technique. This can be followed by a local or some times global bundle adjustment. The alternate pipeline can help in easily updating the reconstructed output and it will not require all the images to be present before running the SfM algorithm.

Another direction which we are exploring is the application of efficient localization to provide an immersive visualization of the personal photo collection. Personal photographs along with the photos available in the community photo collection can provide an immersive experience. Personal images can be localized using our proposed localization algorithm Different user statistics along with the temporal

data available in the personal photographs can be used to approximately estimate the path taken by the person. It can be used to provide a virtual tour of the places visited using the personal photographs.

Related Publications

1. Practical Time Bundle Adjustment for 3D Reconstruction on GPU. Proceedings of ECCV Workshop on Computer Vision on GPU (CVGPU'10)

2. Visibility Probability Structure from SfM Datasets and Application. In European Conference on Computer Vision (ECCV 2012)

Bibliography

[1] S. Achar, C. V. Jawahar, and K. M. Krishna. Large scale visual localization in urban environments. In *ICRA*, 2011.

[2] S. Agarwal, N. Snavely, and S. M. Seitz. Fast algorithms for l infinity problems in multiview geometry. In *CVPR*. IEEE Computer Society, 2008.

[3] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *Proceedings of the 11th European conference on Computer vision: Part II*, ECCV'10, pages 29–42, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] S. Agarwal, N. Snavely, I. Simon, S. Seitz, and R. Szeliski. Building rome in a day. In *ICCV*, 2009.

[5] P. F. Alcantarilla, K. Ni, L. M. Bergasa, and F. Dellaert. Visibility learning for large-scale urban environment. In *ICRA*, 2011.

[6] C. Arth, D. Wagner, M. Klopschitz, A. Irschara, and D. Schmalstieg. Wide area localization on mobile phones. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 73–82, Washington, DC, USA, 2009. IEEE Computer Society.

[7] A. Bartoli, M. Trudeau, N. Guilbert, and S. Roy. Algorithms for batch matrix factorization with application to structure-from-motion. In *CVPR*, 2007.

[8] A. Blake and A. Zisserman. *Visual reconstruction*. MIT Press, Cambridge, MA, USA, 1987.

[9] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:2001, 2001.

[10] M. Brown and D. G. Lowe. Unsupervised 3d object recognition and reconstruction in unordered datasets. In *3DIM*, 2005.

[11] M. Byröd and K. Äström. Bundle adjustment using conjugate gradients with multiscale preconditioning. In *BMVC*, 2009.

[12] J. Cao, K. A. Novstrup, A. Goyal, S. P. Midkiff, and J. M. Caruthers. A parallel levenberg-marquardt algorithm. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, pages 450–459, 2009.

[13] M. Chli and A. J. Davison. Active Matching. In *ECCV*, 2008.

[14] D. Crandall, A. Owens, N. Snavely, and D. P. Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *CVPR*, 2011.

[15] A. Davison, I. Reid, N. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052 –1067, june 2007.

[16] J. Dongarra. Compressed column storage. [web page] `http://netlib2.cs.utk.edu/linalg/html_templates` 1995.

[17] M. Farenzena, A. Fusiello, and R. Gherardi. Structure-and-motion pipeline on a hierarchical cluster tree. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1489 –1496, 27 2009-oct. 4 2009.

[18] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[19] J.-M. Frahm et al. Building rome on a cloudless day. In *ECCV*, 2010.

[20] F. Fraundorfer, C. Engels, and D. Nistér. Topological mapping, localization and navigation using image collections. In *IROS*, 2007.

[21] F. Fraundorfer, C. Wu, J.-M. Frahm, and M. Pollefeys. Visual word based location recognition in 3d models using distance augmented weighting. In *3DPVT*, 2008.

[22] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. In *SIGGRAPH Asia*, 2011.

[23] J. Fung and S. Mann. Openvidia: parallel gpu computer vision. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 849–852, 2005.

[24] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Towards internet-scale multi-view stereo. In *CVPR*, 2010.

[25] R. Gherardi, M. Farenzena, and A. Fusiello. Improving the efficiency of hierarchical structure-and-motion. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1594 –1600, june 2010.

[26] M. Goesele, B. Curless, and S. Seitz. Multi-view stereo revisited. In *CVPR*, 2006.

[27] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. Seitz. Multi-view stereo for community photo collections. In *ICCV*, 2007.

[28] R. I. Hartley and P. F. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997.

[29] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[30] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.

[31] J. Hays and A. A. Efros. im2gps: estimating geographic information from a single image. In *CVPR*, 2008.

[32] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof. From structure-from-motion point clouds to fast location recognition. In *CVPR*, 2009.

[33] K. Josephson and M. Byrd. Pose estimation with radial distortion and unknown focal length. In *CVPR*, pages 2419–2426. IEEE, 2009.

[34] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research, IJRR*, 31:217–236, Feb 2012.

[35] F. Kahl and R. I. Hartley. Multiple-view geometry under the linfinity-norm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1603–1617, 2008.

[36] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[37] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[38] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38:199–218, 2000. 10.1023/A:1008191222954.

[39] F. Li. Probabilistic location recognition using reduced feature set. In *ICRA*, 2006.

[40] H. Li and R. Hartley. Five-point motion estimation made easy. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 630 –633, 0-0 2006.

[41] X. Li, C. Wu, C. Zach, S. Lazebnik, and J. michael Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV*, 2008.

[42] Y. Li, N. Snavely, and D. P. Huttenlocher. Location recognition using prioritized feature matching. In *ECCV*, 2010.

[43] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28:39–55, 2008.

[44] M. Lourakis. levmar: Levenberg-marquardt nonlinear least squares algorithms in C/C++. [web page] `http://www.ics.forth.gr/~lourakis/levmar/`, Jul. 2004.

[45] M. A. Lourakis and A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[46] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 2004.

[47] H. Ltaief, S. Tomov, R. Nath, and J. Dongarra. Hybrid multicore cholesky factorization with multiple gpu accelerators. Technical report, University of Tennessee, 2010.

[48] B. Micusik and J. Kosecka. Piecewise planar city 3d modeling from street view panoramic sequences. In *CVPR*, 2009.

[49] K. W. Mohiuddin and P. J. Narayanan. A gpu-assisted personal video organizing system. In *ICCV Workshops*, pages 538–544, 2011.

[50] K. W. Mohiuddin and P. J. Narayanan. Scalable clustering using multiple gpus. In *HiPC*, pages 1–10, 2011.

[51] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[52] K. Ni, A. Kannan, A. Criminisi, and J. Winn. Epitomic location recognition. In *CVPR*, 2008.

[53] K. Ni, D. Steedly, and F. Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *International Conference on Computer Vision (ICCV)*, 2007.

[54] H. Nielsen. Damping parameter in marquardt's method. Technical Report http://www.imm.dtu.dk/ hbn, Technical University of Denmark, 1999.

[55] D. Nister. An efficient solution to the five-point relative pose problem. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II – 195–202 vol.2, june 2003.

[56] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[57] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(4):353–363, April 1993.

[58] M. Pollefeys et al. Detailed real-time urban 3d reconstruction from video. *Int. J. Comput. Vision*, 2008.

[59] A. Ranganathan. The levenberg-marquardt algorithm. Technical Report http://www.ananth.in/docs/lmtut.pdf, Honda Research Institute, 2004.

[60] T. Review. Noah snavely. `http://www.technologyreview.com/tr35/profile.aspx?TRID=1095`, 2011.

[61] R. Roberts, S. Sinha, R. Szeliski, and D. Steedly. Structure from motion for scenes with large duplicate structures. In *CVPR*, 2011.

[62] D. Robertson and R. Cipolla. An image-based system for urban navigation. In *IN BMVC*, 2004.

[63] C. Rother and S. Carlsson. Linear multi view reconstruction and camera recovery. In *International Journal of Computer Vision*, pages 42–51, 2002.

[64] T. Sattler, B. Leibe, and L. Kobbelt. Fast image-based localization using direct 2d-to-3d matching. In *ICCV*, 2011.

[65] F. Schaffalitzky and A. Zisserman. Multi-view matching for unordered image sets, or "how do i organize my holiday snaps?". In *Proceedings of the 7th European Conference on Computer Vision-Part I*, ECCV '02, pages 414–431, London, UK, UK, 2002. Springer-Verlag.

[66] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, Apr. 2002.

[67] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR*, 2007.

[68] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 519–528, Washington, DC, USA, 2006. IEEE Computer Society.

[69] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1067–, Washington, DC, USA, 1997. IEEE Computer Society.

[70] S. Sengupta, M. Harris, and M. Garland. Efficient parallel scan algorithms for gpus. Technical report, NVIDIA Technical Report, 2008.

[71] Y. Seo and R. Hartley. Sequential l_∞ norm minimization for triangulation. In *ACCV*, 2007.

[72] S. Sinha, D. Steedly, and R. Szeliski. A multi-stage linear approach to structure from motion. In *RMLE 2010 - ECCV Workshop*, 2010.

[73] S. N. Sinha, J. michael Frahm, M. Pollefeys, and Y. Genc. Gpu-based video feature tracking and matching. Technical report, In Workshop on Edge Computing Using New Commodity Architectures, 2006.

[74] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[75] K. N. Snavely. *Scene Reconstruction and Visualization from Internet Photo Collection*. PhD thesis.

[76] N. Snavely. Notre dame dataset. `http://phototour.cs.washington.edu/datasets/`, 2009.

[77] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, 2006.

[78] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal sets for efficient structure from motion. In *CVPR*, 2008.

[79] H. Stewenius, F. Schaffalitzky, and D. Nister. How hard is 3-view triangulation really? In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01*, ICCV '05, pages 686–693, Washington, DC, USA, 2005. IEEE Computer Society.

[80] P. Strum and B. Triggs. Factorization based algorithm for multi-image projective structure and motion. In *ECCV*, 1996.

[81] R. Szeliski. A multi-view approach to motion and stereo. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:1157, 1999.

[82] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, 2000.

[83] V. Vineet and P. J. Narayanan. Cuda cuts: Fast graph cuts on the gpu. *Computer Vision and Pattern Recognition Workshop*, 2008.

[84] Webpage. Noah snavely. `http://www.cs.cornell.edu/~snavely/`, 2012.

[85] Wikipedia. Photogrammetry. [web page] `http://en.wikipedia.org/wiki/Photogrammetry`.

[86] Wikipedia. Structure from motion. [web page] `http://en.wikipedia.org/wiki/Structure_from_motion`.

[87] W. Zhang and J. Kosecka. Image based localization in urban environments. In *3DPVT*, 2006.

[88] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *in ICCV*, pages 666–673, 1999.