Reconstruction of Large Scale 3D Models from Images

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Roman Parys
aus Wrocław/Polen

Tübingen
2013

Tag der mündlichen Qualifikation:	*18.12.2013*
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Andreas Schilling
2. Berichterstatter:	Prof. Dr. Wolfgang Straßer

Zusammenfassung

In dieser Arbeit konzentrieren wir uns auf Methoden zur automatischen Rekonstruktion großer 3D-Szenen basierend auf Bildern. In der Literatur werden Lösungsmethoden dieses Problems als Multi-View Stereo (MVS) Algorithmen bezeichnet, und stellen eine sehr interessante Alternative zum Erwerb von Geometriedaten mittels Laserscanner dar, nachdem die dafür notwendige Ausrüstung - Digitalkameras - wesentlich preiswerter ist. Die Grundlagen von Multi-View Stereo Algorithmen sind Gegenstand zahlreicher wissenschaftlicher Arbeiten und demnach gut erforscht, weshalb sich derzeitige Bemühungen in Richtung Verarbeitung großer Datenmengen verschoben haben. Realistische Modelle von Städten erfordern je nach Größe zur Erfassung ihrer Geometrie Millionen von Bildern. Die Verarbeitung solcher enormen Mengen an Daten erfordert viel Rechenleistung und führt selbst High Performance Computer teilweise an ihre Grenzen. Selbst der Einsatz von Techniken zur Parallelisierung ist oft nicht ausreichend, da sie in den meisten Fällen nur zu einer linearen Verbesserung des Rechenaufwandes führen. Ziel dieser Arbeit ist es, zu zeigen, dass dieser Aufwand nicht nur durch Parallelisierung, sondern auch durch den Einsatz von intelligenteren algorithmischen Ansätzen reduziert werden kann.

Die Notwendigkeit einer qualitativen Bewertung von MVS Algorithmen und die Vielzahl verschiedener Ansätze und deren algorithmische Umsetzungen führten dazu, dass Forscher ein Ranking etablierten [SCD$^+$06]. Die vielversprechendsten Ansätze datieren aus dem Jahre 2009, jedoch zeigen neuere Veröffentlichungen (2011) in diesem Ranking eindeutig einen Trend hin zur Verarbeitung großer Datensätze bei gleichbleibender Qualität der Rekonstruktionen. Generell ist deutlich zu erkennen, dass sich der Schwerpunkt der Forschung in diesem Bereich in Richtung Adaptierung bekannter Methoden auf große Datenmengen verschoben hat.

In dieser Arbeit präsentieren wir einen neuen Ansatz für die Rekonstruktion von Geometrie basierend auf Bilddaten. Die Grundzüge dieses Ansatzes sind wie folgt: Zuerst werden die Daten aus Video- oder Bildersequenzen gewonnen, um anschließend aus jeder einzelnen Sequenz Bildmerkmale zu extrahieren und kompakte Deskriptoren zu generieren. Mittels einer Kalibrierung der Kameras werden für jede Sequenz Kamera Parameter und erste dünn besetzte Punktwolken errechnet. Mit den zuvor generierten kompakten Deskriptoren, berechnen wir einen Ähnlichkeitsgraphen, wobei jeder Knoten innerhalb dieses Graphen eine Sequenz darstellt und die Kanten Verbindungssequenzen mit überlappender Geometrie kennzeichnen. Im nächsten Schritt werden die Transformationsmatrizen der zuvor einzeln während der Kamerakalibrierung generierten 3D-Punktwolken zu einem globalen Koordinatensystem berechnet. Im darauf folgenden Schritt wird zur Verbesserung der bereits berechneten Kamera Parameter, 3D Punkte und der Transformationsmatrizen ein umfassender Bündel-Ausgleich durchgeführt. Abschließend werden dichte Punktwolken anhand traditioneller MVS Methoden erstellt und mittels der optimierten Transformationsmatrizen zu einem Ge-

samtmodell in einem globalen Koordinatensystem zusammengefügt.

Wir werden zeigen, dass die zeitintensivsten Berechnungsschritte unseres Algorithmus parallel ausgeführt werden können. Jedoch gibt es auch Schritte in dem vorgestellten Ansatz, welche nicht auf einfache und natürliche Art und Weise zu parallelisieren sind. Als Beispiele für derartige Schritte wären hier die Konstruktion des Ähnlichkeitsgraphen und der hochdimensionale Bündel-Ausgleich zu nennen.

Abstract

In this thesis, we focus on methods for automatic reconstruction of large 3D scenes directly from images. In the literature, methods solving this problem are referred to as multi-view stereo (MVS) algorithms, and they are a very interesting alternative to the acquisition of geometry with laser scanners, as the equipment - digital cameras - is not expensive. As the MVS reconstruction is a well-researched topic, current efforts are shifted towards a large scale reconstruction. City models require millions of images to capture their geometry. Processing such amounts of data requires a lot of computational effort, even for current super-computers. Exploiting parallelization is often not sufficient, as it leads only to a linear improvement in computational complexity. This effort can be reduced, as described in this thesis, not only by using parallelization, but also with a smart algorithmic approach.

The need of quality evaluation for MVS algorithms and a large number of different approaches has led researchers to establish a ranking [SCD$^+$06]. The most promising approaches are from the year 2009, and recently two new publications were released in 2011, which shows a loss of interest in improving the quality, as there is not much improvement to achieve. It can be clearly seen, that the focus of research in this area has shifted to the application of current methods to large data sets.

In this thesis, we present a new approach to the large scale reconstruction problem. The general outline of this approach is as follows: First we gather data as video or image sequences. We extract image features and build compact descriptors for each sequence. We calibrate cameras for each sequence to obtain camera parameters and sparse 3D point clouds. With our compact descriptors, we compute a similarity graph, where each node is a sequence, and edges are joining sequences representing scenes with overlapping geometry. The next step is to compute transformation matrices between sparse 3D point clouds obtained during the camera calibration process. We compute transformations of sub-models to a global coordinate system. We perform a large scale bundle adjustment to improve camera matrices, 3D points, and transformation matrices. For each image sequence, we compute a dense point cloud with traditional MVS methods. Using the matrices, we bring dense sub-models to a global coordinate system, to obtain a final large model.

As it can be seen, the most time consuming steps of the algorithm can be performed in parallel. However, there are certain steps of our approach, that do not parallelize in an easy, natural way. These are the similarity graph construction, and the large scale bundle adjustment. Thanks to our compact descriptor and our large scale bundle adjustment algorithm these steps can be performed on a single PC. One of the big advantages of our approach is a possibility of incremental model construction. The data does not need to be available at the beginning of the process, and the quality of the global model will be refined as more data will become available.

Contributions

In this thesis, the author focuses on the application of small scale reconstruction methods to large datasets. As the multi-view stereo methods are well developed now, and in terms of accuracy they can be compared to laser scanning techniques, not much space for improvement is still there. Instead of improving the algorithms, the author proposes using them in a larger framework of city reconstruction. Any progress in terms of accuracy and completeness of the small-scale methods can be easily incorporated into the author's software system without changing a single line of code. The author's contributions to the state-of-the-art are as follows:

Compact Similarity Descriptor for Image Sequences

As we work on sets of unorganized models (sometimes GPS data is available), there is a great demand for a compact descriptor that can be used for measuring similarity between those models. Similarity information is required to construct a similarity graph, that is used in later stages of the algorithm to compute transformation matrices between sub-models. Transformation matrices are used to bring all sub-reconstructions to a global coordinate system. As the quality of the 3D model depends on the quality of images, reconstruction algorithm and the model itself (areas with high frequency textures are reconstructed better), we build our descriptor using information contained in the images only. This allows us to construct similarity graphs before a surface is known.

Our descriptor is usually compact - for a standard sequence of up to 100 high resolution images it needs just about 2-10 MB. The similarity evaluation is very fast, and it takes approximately the same time as the similarity evaluation of two images using SIFT features. The output of a comparison function is not only the similarity measure, but a set of possibly matching images between sub-models. Having matching images, together with a camera calibration information, it is easy to compute transformation matrices between sub-reconstructions.

Simple and Effective Approach to Parallelization

Most of the large scale reconstruction approaches try to parallelize current algorithms starting from the camera calibration, to the dense surface reconstruction. The author's approach is to subdivide the the whole large model into a set of sub-reconstructions that can be acquired and processed independently, by many users at the same time. It is not necessary to run the reconstruction on multiple, interconnected cluster nodes. Sub-reconstructions are computed independently, and uploaded to the server that has only one task - to merge them together. The author’s approach shows how to build the final model in an incremental way, and improve the model, when more data is available. Sub-problems, that are not parallelized in a natural way, can be solved on a single computer, thanks to our compact descriptor and the new approach to large scale bundle adjustment.

Camera Calibration for Large Models

In case of methods, that try to reconstruct scenes from huge, unorganized image databases, the problem of camera calibration is becoming difficult. The most time consuming step of camera calibration is the matching of images to the similar ones. Often image search databases are built, and from there, a sparse similarity graph is constructed. These approaches work well for static models. In case, when more images are available, usually the image database needs to be recomputed. In our case, the camera calibration needs to be done on a small sub-model scale, what can be usually 20-500 images, where image matching time is not exploding. Our system depends on external software for this task, so any improvements in the field of camera calibration can be included in our system without making any modifications to the core reconstruction system. When we have transformation matrices between different sub-models, we can easily compute external camera parameters in a common coordinate system.

Global-Error Driven Spanning Tree Construction

As the sub-reconstructions can be imprecise, they may not fit well to the global model, computed on a larger scale. Possible problems are the loop closure and error accumulation, caused by ’weak links’ between sub-models, that can affect global reconstruction error measure. In section 5.5, the author shows how to merge sub-reconstructions into a common coordinate system, while trying to minimize the effect of ’weak links’. The general idea is to find the best order of merging sub-models, that can be interpreted as a spanning tree construction in the similarity graph, with the awareness of the global error. When an optimal merging order is computed, the initial model is closer to the optimum. A good starting point is necessary to obtain a better convergence of the optimization algorithm.

Large Scale Bundle Adjustment

The camera calibration process outputs a set of cameras with parameters associated with them, and a sparse 3D point cloud computed from image features. Bundle adjustment is the final step of the camera calibration. Having initial camera parameters and a sparse 3D point cloud of a model, a reconstruction error function can be defined. The error function measures the inaccuracy of camera calibration. More precisely, it measures the reprojection error of 3D points to images using a camera model. Image features are defined with a sub-pixel precision, and correspondences among different images between the different features are known. Projecting 3D points to images, and computing distances between projected positions and feature positions yields the quality measure for established camera parameters. The error function defined in this way can be optimized, in order to improve positions of 3D points and camera parameters.

The error function has many parameters to optimize: usually it is 12 parameters for each camera, and $3n$ parameters for n points. Having partial solutions of the energy functions within sub-models, we introduce a new approach to bundle adjustment. We propose a new type of energy function with highly reduced parameter count to optimize. Our new, global energy function has only $12m$ parameters, where m is the number of sub-models. We could achieve this by having a transformation matrix associated with each sub-model, that has a direct influence on all cameras and points inside this sub-model, and also describes relations between different sub-models in a global coordinate system.

Texture Compression for Efficient Storage and Rendering

Storing and rendering of reconstructed models is an important problem. In addition to the large number of images containing a lot of redundant information, we need to store intermediate files containing SIFT features and sparse point clouds. We use this data to generate dense point clouds, that tend to be space consuming. A point cloud is not an efficient data for rendering, especially in high resolution, but it can be converted to any other representation.

From the perspective of the Graphics Processing Units (GPUs), a good representation for 3D models are textured triangle meshes, because GPUs were designed to process and display this kind of data from the very beginning. In case of large scale reconstruction, the amounts of 3D data may exceed the system and video memory by a few orders of magnitude. Due to this fact, we need to consider a form of compression, in order to efficiently store and render the data. One of possible options, it to use the S3TC compression scheme, that is supported by most of the consumer graphics cards. Unfortunately, the compression ratio of four bits per pixel is not very high, however the hardware support makes it easy to use. For large data sets, it is necessary to obtain better compression ratios. Very often, in many compression schemes, efficient storing does not come along with efficient and fast decompression. A good example is the JPEG scheme, which is very efficient in terms of storage requirement, but the decompression speed is too slow for real time operation. In many applications, JPEG is used to store data on a fixed storage, but for rendering tasks, the image data is stored in video memory in uncompressed or S3TC compressed form.

As a remedy for the above mentioned problem, we have evaluated the possibility of using Residual Vector Quantization compression for real-time texturing. Our approach allows us to store texture data in the video memory of the GPU, in a compressed form, and decompress it during the rendering. The compression ratio, we have obtained is 1bpp (one bit per RGB pixel), well suited for random access. The scheme is using very simple instructions to decompress the data. Details of this method can be found in Chapter 6.

Acknowledgements

I would like to thank Prof. Dr. Dr. E.h. Wolfgang Straßer and Prof. Dr. Andreas Schilling for the great opportunity of joining the Department of Interactive Graphics Systems at the Univeristy of Tübingen (GRIS). Thanks to those two gentlemen, it was possible for me participate in very interesting projects, to work in an inspiring environment, and to meet extraordinary people.

I would like to thank Prof. Dr. Andreas Schilling and Dr. Günter Knittel for their great support in my work and innumerable hours spent on interesting discussions. Florian Liefers also deserves great acknowledgements for joint efforts in pushing our projects forward.

I want to thank Constanze Christ and Simon Pabst, who helped me to turn myself up in German reality at the beginning of my stay in Germany

Of course the support of my family was very important during my stay in Germany. I would like to thank my parents my sister, and my grandparents. I would like to thank my wife for demonstrating a great patience and understanding for my countless hours spent at the lab.

Contents

1 Introduction **1**
- 1.1 Large Scale Reconstruction 2
- 1.2 Organization of the Thesis 3

2 Introduction to the Multi-view Geometry **5**
- 2.1 Overview 5
- 2.2 A Camera Model and Projections 5
  - 2.2.1 The Pinhole Camera Model 5
  - 2.2.2 Projection Using Homogeneous Coordinates 6
  - 2.2.3 Shift of Principal Point 6
  - 2.2.4 Camera Rotation and Translation 7
  - 2.2.5 Consumer Digital Cameras 7
  - 2.2.6 Computation of the Camera Projection Matrix 8
  - 2.2.7 Camera Matrix Decomposition 9
- 2.3 Multi View Geometry 10
  - 2.3.1 Epipolar Geometry 10
  - 2.3.2 The Fundamental Matrix 10
  - 2.3.3 The Essential Matrix 11
  - 2.3.4 Multi-view Reconstruction 11
- 2.4 Bundle Adjustment 14

3 Review of the State of the Art Reconstruction Methods **16**
- 3.1 Overview and Comparison of MVS algorithms 16
- 3.2 Simple Reconstruction Algorithm 17
- 3.3 Patch-Based Multi View Stereo 18
- 3.4 Reconstruction with Graph Cuts 19

4 Compact Descriptors for Video Sequence Matching **25**
- 4.1 Introduction 25
- 4.2 Related Work 26
  - 4.2.1 Video Sequence Matching 26
  - 4.2.2 State-of-the-Art Image Matching Algorithms 26
  - 4.2.3 Bag-of-Words Approaches 28
- 4.3 Compact Descriptor and Matching 30
  - 4.3.1 Computation of Compact Descriptor 30
  - 4.3.2 Computation of Similarity Measure 31
  - 4.3.3 Computation of Image Occurrence Statistics 32
  - 4.3.4 Computation of Matching Frames 32

4.4 Results 34
4.5 Conclusions and Future Work 38

5 Large Scale Reconstruction 39
5.1 Overview 39
5.2 Related Work 40
  5.2.1 Photo Tourism 40
  5.2.2 Skeletal Graphs 41
  5.2.3 Parallel Approach to Camera Calibration 42
  5.2.4 Efficient Image Relation Graph Construction 43
  5.2.5 Other Approaches for Camera Calibration 44
5.3 Similarity Graph for Sub-models 45
5.4 RANSAC-based Model Matching 45
5.5 Large Scale Calibration of Extrinsic Parameters 47
  5.5.1 Dijkstra's Algorithm 49
  5.5.2 Minimum Spanning Tree 50
  5.5.3 Global-error Driven Minimum Spanning Tree Construction . 50
  5.5.4 Improving the Computational Complexity 51
5.6 Large Scale Bundle Adjustment 53
  5.6.1 Global Geometric Fitting Error Function 54
  5.6.2 Photoconsistency-based Error Function 55
5.7 Similarity Graph Optimization 56
5.8 Single Camera Optimization 57
5.9 Results 57

6 Compression Algorithm for Efficient Storage and Rendering 63
6.1 Overview 63
6.2 Rendering with Compressed Textures 63
  6.2.1 Introduction 63
  6.2.2 Related Work 65
  6.2.3 Residual Vector Quantization 66
  6.2.4 Texture Compression Using RVQ 66
  6.2.5 Real-Time Rendering From Compressed Textures 67
  6.2.6 Results 69
6.3 Rendering of Compressed Volume Data Sets 69
  6.3.1 Introduction 69
  6.3.2 Related Work 70
6.4 The Giga-Voxel System 71
  6.4.1 Residual Vector Quantization for Volume Data Sets 71
  6.4.2 Compressing the Visible Human Female 72
  6.4.3 Rendering 72
  6.4.4 The Raycaster 75
  6.4.5 Blending and Occlusion Culling 76
  6.4.6 Multi-Resolution Rendering 76
  6.4.7 Parallel Rendering on the Cluster 77
  6.4.8 Performance 80

Chapter 1

Introduction

Re-creation of 3D models of real world scenes has always been an interesting topic for film and game industry, digitalization of cultural heritage, and virtual reality. Building virtual 3D scenes has always required talented people with artistic sense, specialized and expensive software, strong computational resources for photorealistic visualization, and the most important - a lot of manual work.

Recently, a lot of research has been done in the field of automatic acquisition of 3D models and scenes. Research device prototypes are often not ready for commercial market. Laser scanners are in a well developed state, however in practice they produce massive point clouds, that are hard to process and visualize. As current graphics processing units (GPUs) are designed to render triangle meshes, point clouds need to be triangulated in order to achieve good rendering performance. Very often, reconstruction of triangular surfaces is an ill-posed problem, and due to noise and outliers, it is very hard to obtain a high quality surface in a completely automated way. The other approach to rendering, using point clouds directly, lacks the dedicated hardware support. Of course, the visualization algorithms for point clouds exist, but they are limited by the following issues. The first issue, is the complexity of the algorithms, that depends on the screen resolution. Currently a standard is a resolution of 2 mega-pixels, and there are even research efforts to build giga-pixel displays. The second issue of direct point rendering is the storage and bandwidth requirement, as point clouds tend to be bigger in size, than equivalent triangular models. Despite of these problems, a point cloud is an interesting alternative data representation, as it allows for manipulation without taking care of connectivity between elements. As research is progressing, more and more advanced techniques are being developed for processing point clouds, so this representation is very useful in intermediate stages of processing.

Commercial solutions based on laser scanners and structured light scanners are available, however they are often limited to acquisition of small objects. Additionally, due to high prices, they are beyond of reach for the consumer area. For small objects acquired under controlled conditions, usually it is much easier to produce triangle meshes. For small objects, any manual editing is not as tedious, as it would be in case of really huge scenes.

A very interesting area of research is the automatic reconstruction of scenes based on images and video sequences only. Methods developed within the area can be brought to the consumer area, as digital cameras and camcorders are widely available at relatively low prices. Multi-view stereo (MVS) methods, used for 3D reconstruction from images, are quite well developed. These methods are ranked by accuracy and completeness, and can be compared in the Middlebury Database ([SCD$^+$06]). Accuracy means, how close the reconstructed models are from the ground truth. Current MVS methods presented in the Middlebury Database have an accuracy of less than 0.5 mm measured for the test object. The completeness is the percentage of surface reconstructed, and for some of the test data sets, this number reaches even up to 99.9%. The main disadvantage of MVS methods is inability of precise surface reconstruction of texture-less areas, as these areas look very similar on many images, and no information about the real surface can be inferred. Different methods have different approaches - skip surface reconstruction in these areas in order to preserve better overall accurracy, or make certain assumptions on the surface in order to produce more complete models, but with lower accuracy. Different MVS methods can produce different types of 3D models, starting from point clouds, volume models, to triangle meshes.

1.1 Large Scale Reconstruction

There is not much space left for improvements in quality and completeness of models produced by MVS methods. This can be clearly seen in the Middlebury database, where the most influential papers are from 2009, and no significant progress has been made since then.

The MVS methods work well for small scenes, build from a small number of images, however a large scene reconstruction is a big and complex problem, that requires a new concept. The main sub-problems of the large scale approach are as follows:

Organizing a Large Number of Input Images

The input set of images can be unorganized or only partially organized. In both cases, it is required to group images depicting the same scene geometry, as it is a pre-requisite for camera calibration. The problem becomes expensive in terms of computational complexity, if the number of images is increasing. A naive solution would be to consider all the pairs of images and compute their similarity, however the time required to accomplish this task grows quadratically with the number of images. This is a challenging problem, that is also addressed in this thesis.

Large Scale Camera Calibration

Once the image set has been organized, a recovery of camera parameters used during image capture is necessary. A good initial estimate of the internal camera parameters - focal length, radial distortion, center of projection - can be computed using a small set of images. External parameters (position, rotation) are embedded in a global coordinate system of a huge model. This makes it hard to compute their positions using only local information, without taking into account the whole model.

Building a High Quality Large Model

Once the camera parameters have been computed, it is possible to use a MVS method to recover the 3D geometry of a scene. Since MVS methods by definition need images as input, the problem of storing images in the system memory arises. It is clear that for huge data sets it is not possible to store the whole set of images in the memory, and frequent disk accesses slow down the reconstruction algorithm. Even with the assumption of having very fast and large memory, current CPU speeds still can be a limiting factor. This brings us to the next problem - parallelization.

Effective Parallelization

Parallelization is often used as a remedy for a high demand for computational resources. However the improvements achieved in this way are often not sufficient, as they can provide a linear speedup only. The non-regularity of the large scale reconstruction problem makes it hard to achieve a proper load balancing. Despite this fact, we show an effective way of distributing the computation that is complemented by extremely effective algorithms that reduce the computational complexity of certain parts of the algorithm.

1.2 Organization of the Thesis

The work presented in this thesis is organized as follows.

Chapter 2: Introduction to the Multi-view Geometry

In this chapter, we present the foundations of all multi-view stereo methods. We describe the pinhole camera model and projections in homogeneous coordinates, that are used in the most of the algorithms. We also present basic algorithms for the computation of the camera projection matrix, two-view and multi-view geometry, and camera calibration. Basic terms defined in this chapter are required to fully understand the main subject of the thesis, however a reader with a computer vision experience may safely skip this part. The author is using equations and algorithms from this part as a reference for software implementation of the large scale reconstruction system.

Chapter 3: Review of the State of the Art Reconstruction Methods

As our contributions are build on top of the state-of-the art small scale reconstruction methods, we present a short review of the most important algorithms in this area.

Chapter 4: Compact Descriptors for Video Sequence Matching

As a basic building block used in our large scale reconstruction algorithm, we use an image sequence or set. The reason for this, is that during image acquisition with digital cameras or camcorders, usually consequent images depict the same geometry, they are located in the same area, so exploiting this partial organization, will save a lot of computation in later stages of the algorithm. We do not assume any inter-sequence organization of input data, however the GPS coordinates save a lot of time during later processing.

This part describes our contribution, a compact descriptor for an image sequence. The descriptor is used to compute the similarity between two sequences or sets of images. Additionally, it allows us to select individual matching images from sequences. With our compact descriptor, and optionally using the GPS data, we are able to organize large amounts of data in a very short time. This is equivalent to building a similarity graph where each node is an image sequence, and edges connect nodes containing images of the same scene geometry.

Chapter 5: Large Scale Reconstruction

In this part of the thesis we describe another contribution - our algorithm of large scene reconstruction - and compare it with other competing approaches. The algorithm starts with a set of image sequences and 3D sub-models created for those sequences, and creates one large, coherent 3D model.

Chapter 6: Compression Algorithm for Efficient Storage and Rendering

Another contribution presented in this thesis is the evaluation of Residual Vector Quantization compression in the context of storing and rendering large amounts of pixel data. As the reconstructed model can be converted to a textured mesh, or it can be rendered as a point cloud with a splatting algorithm that is using textures to store colors of the points, we need an efficient compression scheme. For efficient decompression and rendering the following requirements must be met:

- High compression ratio.
- Random access to pixels in the compressed representation.
- Extremely high decompression speed, that will allow us to render directly from the compressed representation.

Unfortunately, none of the standard techniques fulfill all the requirements at the same time. However, the Residual Vector Quantization method, invented more than 30 years ago, and forgotten due to insufficient computing power in former times, required for the data compression, can satisfy all of the above conditions. To our knowledge, our attempt of evaluation of this method for large data sets, is the first one.

Additionally, we present the decompression method in the context of real-time rendering with a 6.4 giga-pixel sized texture and interactive rendering of a giga-voxel sized volume model.

Chapter 2

Introduction to the Multi-view Geometry

2.1 Overview

In this chapter, we describe the foundations of most of the existing 3D reconstruction methods. We introduce the *pinhole camera* model that is used in most standard reconstruction methods. This model is used to describe an image formation process, in which 3D scenes are projected onto the surface of the camera sensor or film. Knowing the camera projection parameters, we can infer some information about 3D geometry of a photographed scene. This chapter mostly follows [HZ04].

2.2 A Camera Model and Projections

In this section, we describe the simple pinhole camera model, as introduced in [HZ04]. This model is sufficient to describe the properties of the most consumer digital cameras, and is widely used in the state-of-the-art reconstruction algorithms. For special, non-standard models, the reader is referred to [HZ04].

2.2.1 The Pinhole Camera Model

The pinhole camera model describes, how 3D points are projected onto a plane. In this model, we assume, that the center of projection $\mathbf{C}$ is the origin of Euclidean coordinate system, and the projection plane, often called *image plane* or *focal plane* is perpendicular to the Z axis, located at offset f from the origin. The value of f is known as *focal length*.

In this model, the projection of a 3D point $\mathbf{X} = (X, Y, Z)^T$ is the intersection of a 3D line connecting $\mathbf{C}$ and $\mathbf{X}$, with the image plane. The projection can be computed with the following formula:

$$(X, Y, Z)^T \to (fX/Z, fY/Z, f)^T \tag{2.1}$$

Dropping the third coordinate gives us a mapping from 3D space to a 2D image. The basic setup of the pinhole camera model is shown in Fig. 2.1.

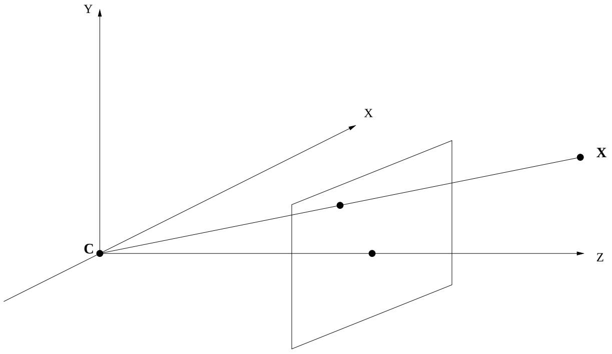


Figure 2.1: Pinhole camera model and projection.

2.2.2 Projection Using Homogeneous Coordinates

Sometimes it is convenient to express the mapping 2.1 as a matrix multiplication. In order to do it, we need to define this projection in homogeneous coordinates. This projection can be written in the following form:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{pmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

When using the standard notation, where $\mathbf{X}$ is a 3D point in homogeneous coordinates $(X, Y, Z, 1)^T$, $\mathbf{x}$ is the image point represented as homogeneous three element vector, and $\mathbf{P}$ is the 3×4 homogeneous *camera projection matrix*, we can simply write the equation 2.2 as

$$\mathbf{x} = \mathbf{PX} \tag{2.3}$$

2.2.3 Shift of Principal Point

The origin of the image coordinate system is called the *principal point*. In practice, it may not be the case, that the principal point is located in the center of the projection plane. Having the origin of the coordinate system in the image corner is more convenient for accessing image pixels directly after the projection. This can be achieved by introducing the *principal point offset*. The projection can then be described as

$$(X, Y, Z)^T \rightarrow (fX/Z + p_x, fY/Z + p_y)^T \tag{2.4}$$

In the matrix form, the projection can be written then as

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.5}$$

Now we can define the *camera calibration matrix* $\mathbf{K}$ in the following form:

$$\mathbf{K} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.6}$$

The projection now can be written in a compact form as

$$\mathbf{x} = \mathbf{KX} \tag{2.7}$$

2.2.4 Camera Rotation and Translation

The previous model assumes, that all 3D points are defined within the local camera coordinate system. In many applications, we need to consider more than a single camera observing the same scene. This leads us to a common *world coordinate system*, in which the cameras are embedded. There is a certain relation between the local camera coordinate system and the world coordinate system. This relation between the two systems, is expressed by using a rotation and a translation. Let $\tilde{\mathbf{X}}$ be a 3D point in the world coordinate system, $\tilde{\mathbf{X}}_{cam}$ is the same point in the local camera coordinate system. Then we may write $\tilde{\mathbf{X}}_{cam} = \mathbf{R}(\tilde{\mathbf{X}} - \tilde{\mathbf{C}})$, where $\tilde{\mathbf{C}}$ is the camera center in world coordinates, and $\mathbf{R}$ is 3×3 rotation matrix that describes the rotation of the camera in the world coordinate system. In matrix form, this relation between the different coordinate frames can be written as

$$\mathbf{X}_{cam} = \begin{bmatrix} R & -R\tilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{2.8}$$

In order to project a homogeneous point expressed in the world coordinate frame, the two equations 2.7 and 2.8 together can be written in a short form as

$$\mathbf{x} = \mathbf{KR}[I| - \tilde{\mathbf{C}}]\mathbf{X} \tag{2.9}$$

The entries of matrix $\mathbf{K}$ are called *internal* camera parameters, and the matrix $\mathbf{R}$ and vector $\tilde{\mathbf{C}}$ are known as *external* camera parameters.

In order to simplify the projection of 3D points from the world coordinate system to an image plane (Equation 2.9), $\mathbf{K}$,$\mathbf{R}$, and $\tilde{\mathbf{C}}$ can be combined into a single 4×3 camera projection matrix:

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{2.10}$$

where $\mathbf{t} = -\mathbf{R}\tilde{\mathbf{C}}$.

2.2.5 Consumer Digital Cameras

The model described above is very simple, but somehow limited. It does not include any distorting properties of the lens, and assumes that pixels are square, what not always needs to be the case.

Non-square Pixels

Non-square pixels may cause unequal scaling in two image directions. If the number of pixels per unit distance in x and y direction is denoted by m_x and m_y, then we need to include those two factors in the projection from world coordinates to the image frame. We can compensate for this by multiplying the corresponding $\mathbf{K}$ matrix elements by those values:

$$\mathbf{K} = \begin{pmatrix} fm_x & 0 & p_x m_x \\ 0 & fm_y & p_y m_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.11}$$

Radial Distortion

Real lenses cannot be accurately described by the previous linear camera model. The most influential factor is the radial distortion, seen especially in low quality, wide angle lenses. In order to be able to use the pinhole camera model, the image measurements need to be corrected.

Let $(\tilde{x}, \tilde{y})$ be the image coordinates of an ideal pinhole projection (with the principal point in the origin). Radial distortion can be modeled as

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = L(\tilde{r}) \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} \tag{2.12}$$

where $(\tilde{x}, \tilde{y})$ is the ideal image position, (x_d, y_d) is the actual image position, $\tilde{r}$ is the Euclidean distance from the ideal image position to the center, and $L(\tilde{r})$ is a distortion function. The corrected pixel coordinates are the following:

$$\hat{x} = x_c + L(r)(x - x_c) \quad \hat{y} = y_c + L(r)(y - y_c)$$

where (x, y) are the projected coordinates, $(\hat{x}, \hat{y})$ are the corrected coordinates, and (x_c, y_c) is the center of radial distortion.

The following choice for the distortion function and the distortion center is suggested in [HZ04]. As the function $L(r)$ is defined in the positive domain, and $L(0)$=1, then an arbitrary function with these properties can be approximated by a Taylor expansion $L(r) = 1 + \sum_{i=1}^{\infty} \kappa_i r^i$. The choice for center of radial distortion is often the principal point. Radial distortion parameters, additionally to the matrix $\mathbf{K}$ elements, are also considered as internal camera parameters.

In reconstruction algorithms, there are two possible options, how to handle the radial distortion. The first one can directly use the correction formulas, while executing the algorithm. The other way is to undistort the images, and during execution of the algorithm, use standard linear projection model. The disadvantage of the second concept is the quality reduction in the input images, however the advantage of purely linear projection model, could be exploited in the course of geometry optimization, where faster or simpler optimization methods could be applied.

2.2.6 Computation of the Camera Projection Matrix

Once we have defined the pinhole camera model, we will show how to recover camera parameters from a set of images and a small number of correspondences between measured 3D points and their image projections. Despite the fact, that in this thesis we focus on fully automatic methods, that do not have any clues about the 3D geometry and correspondences in the photographed scenes, this paragraph is still important. Very often we have situations, where internal camera parameters are constant, so they can be computed only once. With fixed internal parameters, we can reduce the complexity of full camera calibration, by limiting the number of unknown variables in the course of optimization.

Our assumption is, that we have a set of correspondences $C = \{\mathbf{X}_i \leftrightarrow \mathbf{x}_i\}_{i=1}^N$ between 3D points and their 2D image projections. The task is to find the camera projection matrix $\mathbf{P}$, such that $\forall i\ \mathbf{x}_i = \mathbf{P}\mathbf{X}_i$. In order to simplify the solution of this set of equations, we may rewrite this in terms of a cross product: $\forall i\ \mathbf{x}_i \times \mathbf{P}\mathbf{X}_i = 0$. This is possible, since the homogeneous vectors $\mathbf{x}_i$ and $\mathbf{P}\mathbf{X}_i$ should point into the same direction, but they may differ in scale factor.

This leads directly to the following relationship:

$$\begin{bmatrix} \mathbf{0}^T & -w_i\mathbf{X}_i^T & y_i\mathbf{X}_i^T \\ w_i\mathbf{X}_i^T & \mathbf{0}^T & -x_i\mathbf{X}_i^T \\ -y_i\mathbf{X}_i^T & x_i\mathbf{X}_i^T & \mathbf{0}^T \end{bmatrix} \begin{pmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{pmatrix} = 0 \tag{2.13}$$

where $\mathbf{p}^{iT}$ is the i-th row of $\mathbf{P}$, a four dimensional vector. In order to reduce the number of equations, by exploiting the linear dependence of the equations in 2.13, we can use the following:

$$\begin{bmatrix} \mathbf{0}^T & -w_iX_i^T & y_iX_i^T \\ w_iX_i^T & \mathbf{0}^T & -x_iX_i^T \end{bmatrix} \begin{pmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{pmatrix} = 0 \tag{2.14}$$

Using a set of n correspondences, and using equation 2.14, we can build a matrix $\mathbf{A}$ of size $2n \times 12$, and solve the equation set of $\mathbf{A}\mathbf{p} = \mathbf{0}$ for the elements of $\mathbf{P}$ contained in $\mathbf{p}$. For example, the Singular Value Decomposition can be used to obtain a solution. At least 6 correspondences are required to solve this equation set, but often the measurements are not precise enough to obtain a stable solution. In this case, more correspondences should be used. In this case, the solution will be obtained by minimizing the geometric or algebraic error. For more information on this topic, the reader is referred to [HZ04].

2.2.7 Camera Matrix Decomposition

In Section 2.2.6, it is shown how to obtain the full camera projection matrix. In many cases, we are interested in obtaining internal camera parameters. This is important, when those parameters are always fixed, because it allows us to use them later for reducing the number of parameters required for full camera matrix estimation for many images. The initial calibration can be performed with a known 3D model and manual correspondence finding. Often a calibration object like a chessboard is used, and automatic detection of corners can create a set of correspondences in an automated way. In this subsection, we present a camera decomposition method, which can extract internal and external camera parameters from the previously computed camera matrix.

Camera Center

The camera center $\mathbf{C}$ fulfills the equation $\mathbf{P}\mathbf{C} = 0$. We can use Singular Value Decomposition of the matrix $\mathbf{P}$ in order to solve for elements of $\mathbf{C}$. The center $\mathbf{C} = (X, Y, Z, T)^T$ can be also obtained with an algebraical method from the following formulas:

$$X = det([\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]) \qquad Y = det([\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4])$$

$$Z = det([\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_4]) \qquad T = det([\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3])$$

Orientation and Internal Parameters

From Equation 2.9, we can write

$$\mathbf{P} = [\mathbf{M}| - \mathbf{M}\tilde{\mathbf{C}}] = \mathbf{K}[\mathbf{R}| - \mathbf{R}\tilde{\mathbf{C}}]$$

From this equation it is clear, that we can find $\mathbf{K}$ and $\mathbf{R}$ by decomposition of $\mathbf{M}$, because $\mathbf{M} = \mathbf{KR}$. The RQ-decomposition algorithm can split $\mathbf{M}$ into a product of two matrices: upper-triangular matrix $\mathbf{K}$, and orthogonal matrix $\mathbf{R}$. As the decomposition may lead to the ambiguity, we require that $\mathbf{K}$ must have positive entries on the diagonal. The $\mathbf{K}$ matrix is

$$\mathbf{K} = \begin{pmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{pmatrix}$$

where α_x and α_y are scaling factors in x and y direction, s is the skew parameter, and $(x_0, y_0)^T$ is the principal point.

2.3 Multi View Geometry

2.3.1 Epipolar Geometry

The term *epipolar geometry* describes the intrinsic projective geometry between two views. This geometry does not depend on the scene geometry. It depends on the internal camera parameters and a relative pose between cameras.

2.3.2 The Fundamental Matrix

The intrinsic geometry is described by the *fundamental matrix* $\mathbf{F}$. The size of this matrix is 3×3, and rank is 2. If a 3D point $\mathbf{X}$ projects to points $\mathbf{x}$ and $\mathbf{x}'$ in two cameras, then those points satisfy the relation

$$\mathbf{x}'\mathbf{F}\mathbf{x} = 0. \tag{2.15}$$

This relation can be interpreted as follows. The term $\mathbf{F}x = (l_x, l_y, l_z)^T = l$ represents a 2D line equation $l_x x + l_y y + l_z = 0$. The term $\mathbf{x}'l = 0$ requires the point $\mathbf{x}'$ to be located on the line l (the distance from $\mathbf{x}'$ to l is 0). The line l is called the *epipolar line*. The visual representation of epipolar geometry is shown in Figure 2.2.

An important term is the *epipole*. It is the intersection point of the line joining camera centers $\mathbf{C}$ and $\mathbf{C}'$ in Figure 2.2 with the image plane. The line $(\mathbf{C}, \mathbf{C}')$ is called the *baseline*.

In situation from Figure 2.2, let's suppose, we have only a projection $\mathbf{x}$ of a 3D point $\mathbf{X}$, and the fundamental matrix between the two views, but we do not know the exact position of point $\mathbf{X}$ in 3D space, and its projection $\mathbf{x}'$ to the second camera. In this case the position of point $\mathbf{x}'$ is restricted to the epipolar line corresponding to the point $\mathbf{x}$.

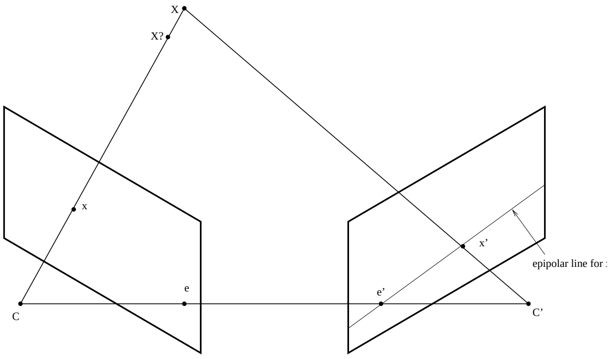


Figure 2.2: Epipolar geometry

2.3.3 The Essential Matrix

The *essential matrix* is a specialization of the fundamental matrix for the case of normalized image coordinates. We define the normalized coordinates. Let $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ and $\mathbf{x} = \mathbf{P}\mathbf{X}$. When the calibration matrix $\mathbf{K}$ is known, we may write $\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x} = [\mathbf{R}|\mathbf{t}]\mathbf{X}$, where $\hat{\mathbf{x}}$ is the image point expressed in normalized coordinates. The matrix $\mathbf{K}^{-1}\mathbf{P} = [\mathbf{R}|\mathbf{t}]$ is known as a *normalized matrix*, where the effect of the known calibration matrix is removed. The essential matrix is defined by the following equation:

$$\hat{\mathbf{x}}'^{T}\mathbf{E}\hat{\mathbf{x}} = 0; \tag{2.16}$$

By substituting for $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}'}$, we get $\hat{\mathbf{x}}'^{T}\mathbf{K}'^{-T}\mathbf{E}\mathbf{K}^{-1}\hat{\mathbf{x}} = 0$; Now the relation between the fundamental and essential matrix can be written as

$$\mathbf{E} = \mathbf{K}'\mathbf{F}\mathbf{K}; \tag{2.17}$$

2.3.4 Multi-view Reconstruction

In this section, we describe how to reconstruct the scene and to compute camera parameters, first from two views, and later for more views. The prerequisite for the reconstruction is a set of 2D correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ originating from a set of unknown 3D points $\mathbf{X}_i$. Our task is to find two camera matrices $\mathbf{P}$ and $\mathbf{P}'$ and a set of 3D points $\mathbf{X}_i$, such that for each i the following equations are fulfilled:

$$\mathbf{x}_i = \mathbf{P}\mathbf{X}_i \;\; \mathbf{x}'_i = \mathbf{P}'\mathbf{X}_i \tag{2.18}$$

For stable results, a sufficient number of points is required. The reconstruction is done in three steps:

1. Compute the fundamental matrix from the point correspondences.
2. Compute the camera matrices from the fundamental matrix.
3. For each correspondence, compute the 3D point, that projects to the two image points.

The approach presented here is very general, and there exist many variants of this method. The details can be found in [HZ04].

Computation of the Fundamental Matrix

When the correspondences are known, and fulfill the condition $\mathbf{x}_i'\mathbf{F}\mathbf{x}_i = 0$, an equation set can be build and solved for the entries of the fundamental matrix. At least 7 points are required to solve the equation set. One equation for the corresponding points $(x, y, 1)$ and $(x', y', 1)$ can be written as:

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0 \quad (2.19)$$

If the entries of matrix $\mathbf{F}$ are written as a vector $\mathbf{f}$ in row-major order, then Eq. 2.19 can be written in a form

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\mathbf{f} = 0 \quad (2.20)$$

The whole equation system for a set of n point matches can be written as

$$\mathbf{Af} = \begin{pmatrix} x_1'x_1 & x_1'y_1 & x_1' & y_1'x_1 & y_1'y_1 & y_1' & x_1 & y_1 & 1 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x_n'x_n & x_n'y_n & x_n' & y_n'x_n & y_n'y_n & y_n' & x_n & y_n & 1 \end{pmatrix} \mathbf{f} = 0 \quad (2.21)$$

This system of equations is usually solved with SVD. The least-squares solution for the fundamental matrix entries is the singular vector corresponding to the smallest singular value of $\mathbf{A}$. In this case the last column of $\mathbf{V}$ in the SVD $\mathbf{A} = \mathbf{UDV}^T$. This solution minimizes $||\mathbf{Af}||$ subject to the condition $||\mathbf{f}|| = 1$.

Enforcing the Singularity Constraint

Many applications rely on the fact that the fundamental matrix is of rank 2. If this matrix is not singular, then the epipolar lines are not coincident. The solution to the linear equations generally does not produce a rank 2 fundamental matrix, so this constraint should be enforced. The convenient way of doing that is by correcting the fundamental matrix $\mathbf{F}$ obtained by the SVD solution from $\mathbf{A}$. Matrix $\mathbf{F}$ is replaced by another matrix $\mathbf{F}'$ that minimized the Frobenius norm $||\mathbf{F} - \mathbf{F}'||$, while holding the condition $\det \mathbf{F} = 0$. An easy way for calculating $\mathbf{F}'$ is to use the SVD again. When $\mathbf{F} = \mathbf{UDV}^T$ is the SVD of $\mathbf{F}$, where $\mathbf{D} = \mathrm{diag}(r, s, t)$ is a diagonal matrix with $r \geq s \geq t$. Then $\mathbf{F}' = \mathbf{U}\mathrm{diag}(r, s, 0)\mathbf{V}^T$ is the matrix minimizing the Frobenius norm.

The 8-point Algorithm

Here we introduce the normalized 8-point algorithm, that computes the fundamental matrix from a set of at least 8 corresponding points.

1. **Normalization:** First the image coordinates are transformed with formulas $\hat{\mathbf{x}}_i = \mathbf{T}\mathbf{x}_i$ and $\hat{\mathbf{x}}_i' = \mathbf{T}'\mathbf{x}_i'$. The matrices $\mathbf{T}$ and $\mathbf{T}'$ are normalizing transformations consisting of translation and scaling. The suggested translation is that the centroid of points is at the origin of the coordinates. The scaling is chosen in such a way, that root mean square (RMS) of the points from the origin is equal to $\sqrt{2}$.

2. Find the fundamental matrix $\hat{\mathbf{F}}'$ corresponding to matches $\hat{\mathrm{x}} \leftrightarrow \hat{\mathrm{x}}'$ by:
    - **Linear solution:** Compute $\hat{\mathbf{F}}$ from the singular vector corresponding to the smallest singular value of $\hat{\mathbf{A}}$, where $\hat{\mathbf{A}}$ is composed from the matching points $\hat{\mathrm{x}} \leftrightarrow \hat{\mathrm{x}}'$.
    - **Constraint enforcement:** Replace $\hat{\mathbf{F}}$ by $\hat{\mathbf{F}}'$ in such a way that $\det \hat{\mathbf{F}}' = 0$ using SVD.
3. **Denormalization:** Output the fundamental matrix as $\mathbf{F} = \mathbf{T}'^T \hat{\mathbf{F}}' \mathbf{T}$. The matrix $\mathbf{F}$ corresponds to the original set of matching points.

Handling Outliers

The normalized 8-point algorithm assumes, that the correspondences are true. This is not always the case. As described in Chapter 4, the most widely used SIFT algorithm [Low04] can output correspondences, that may not reflect the 3D geometry of a scene, as the matching is done by considering only local image descriptors. In this case a RANSAC approach is used, where a random subset of correspondences (usually 8) is selected as a hypothesis and the reconstructed fundamental matrix is used to verify this hypothesis by computing how many points fulfill the epipolar line condition. The procedure is repeated, and the best hypothesis is selected as a solution.

Computation of Camera Matrices

The cameras $\mathbf{P}$ and $\mathbf{P}'$ can be extracted from the essential matrix. The essential matrix can be computed directly from the normalized image coordinates, or it can be computed from the fundamental matrix. The solution that can be computed from the essential matrix, can be determined up to a scale. If we assume that the first camera is $\mathbf{P} = [\mathbf{I}|\mathbf{0}]$, then in order to compute the second matrix $\mathbf{P}'$, we can factor $\mathbf{E}$ into the product $\mathbf{SR}$ of a skew-symmetric matrix and a rotation matrix. If the SVD of $\mathbf{E}$ is $\mathbf{U}\mathrm{diag}(1, 1, 0)\mathbf{V}^T$, then there are two possible factorizations $\mathbf{E} = \mathbf{SR}$:

$$\mathbf{S} = \mathbf{UZU}^T \ \mathbf{R} = \mathbf{UWV}^T \text{ or } \mathbf{UW}^T\mathbf{V}^T$$

Having the essential matrix $\mathbf{E} = \mathbf{U}\mathrm{diag}(1, 1, 0)\mathbf{V}^T$ and the first camera matrix $\mathbf{P} = [\mathbf{I}|\mathbf{0}]$ we can compute the second camera matrix with the following formula:

$$\begin{gathered}\mathbf{P}' = [\mathbf{UWV}^T| + \mathbf{u}_3] \ \text{ or } \ [\mathbf{UWV}^T| - \mathbf{u}_3] \\ \text{or } \ [\mathbf{UW}^T\mathbf{V}^T| + \mathbf{u}_3] \ \text{ or } \ [\mathbf{UW}^T\mathbf{V}^T| - \mathbf{u}_3]\end{gathered}$$

where $\mathbf{u}_3 = \mathbf{U}(0, 0, 1)^T$.

Computation of 3D Points

When we have $\mathbf{P}$ and $\mathbf{P}'$, the next step is to compute 3D coordinates of points corresponding to the matches. This step has a geometric interpretation. As $\mathbf{x}'$ lies on the epipolar line $\mathbf{Fx}$, the two rays back-projected from points $\mathbf{x}$ and $\mathbf{x}'$ lie in a common epipolar plane. This plane is passing the camera centers. Because the two rays are located on the same plane, they intersect, in this case in a point X. Often the elements of the $\mathbf{X}$ vector are treated as unknowns in a set of equations $\mathbf{x} = \mathbf{PX}$ and $\mathbf{x}' = \mathbf{P}'\mathbf{X}$ and the equation set is solved numerically.

Extension to Multiple Cameras

We have described the simple two view reconstruction algorithm. [HZ04] also covers three and more view reconstruction in detail. The theory used in the two view reconstruction can be extended to multiple views in a straight forward way. Starting with two views, we reconstruct cameras and 3D points. Then we can add a third camera by selecting one of the previously computed cameras, and perform two-view reconstruction. Then the newly computed camera and 3D points should be transformed to the coordinate system of the two original cameras. In this manner, we can add new views one-by-one. As the reconstruction error may accumulate, this reconstruction approach should be considered only as an initial solution for Bundle Adjustment described in Section 2.4.

2.4 Bundle Adjustment

In previous sections, the basic theory of camera model, projections, and the algorithm for reconstruction of camera paramters were presented. In this section, we present the basic Bundle Adjustment (BA) algorithm from [HZ04]. More details can be found in this book. A software library called Sparse Bundle Adjustment is presented in [LA09].

Imagine a situation, where a set of 3D points $\mathbf{X}_j$ is observed by a set of camera with projection matrices $\mathbf{P}^i$. The 2D coordinates $\mathbf{x}^i_j$ denote point $\mathbf{X}_j$ seen by i-th camera. We want to solve the problem of finding the set of camera matrices $\mathbf{P}^i$ and 3D points $\mathbf{X}_j$ such that the following equation is fulfilled:

$$\mathbf{P}^i\mathbf{X}_j = \mathbf{x}^i_j \tag{2.22}$$

The image measurements may be noisy, so the Equation 2.22 may not be fulfilled precisely. In this case we are interested in the Maximum Likelihood (ML) solution under the assumption of a Gaussian noise in the measurements. This means, that we are really interested in estimating projection matrices $\hat{P}^i$ and 3D points $\hat{\mathbf{X}}_j$ which project exactly to the image points $\hat{\mathbf{x}}^i_j$ using the projection matrices $\hat{\mathbf{P}}^i$ and minimize the image distances between reprojected and detected points:

$$\underset{\hat{\mathbf{P}}^i,\hat{\mathbf{X}}_j}{argmin} \sum_{ij} d(\hat{\mathbf{P}}^i\hat{\mathbf{X}}_j, \mathbf{x}^i_j)^2 \tag{2.23}$$

In the above equation, the $d(\mathbf{x}, \mathbf{y})$ is the Euclidean distance between 2D image points $\mathbf{x}$ and $\mathbf{y}$. The procedure of solving the Equation 2.23 is known as *bundle adjustment*. In can be seen as adjusting a bundle of rays between each camera center and a set of 3D points. The bundle adjustment is used as a final step of a reconstruction algorithm.

The pre-requisite for this algorithm is a good initial solution. Additionally, the problem may be the huge number of parameters in the Equation 2.23. For n points and m views, we have $3n + 11m$ parameters - 11 degrees of freedom for each camera, and 3 parameters for a point. When using the Levenberg-Marquardt algorithm to minimize Equation 2.23, then factoring (or sometimes inverting) of matrices of size $(3n+11m)^2$ is required. For large m and n this becomes very slow or impossible. There are a few solutions for this problem:

- Reduction of the number of parameters: It is possible to exclude some views or points or partition the data to subsets, and merge them later.

- Interleaving the camera and points estimations. First optimize the points while keeping cameras fixed, then optimize cameras while keeping the points fixed, and repeat it in a loop.
- Sparse methods. Ready to use open source software library is provided [LA09].

Chapter 3

Review of the State of the Art Reconstruction Methods

In order to make this thesis complete, in this chapter we present the most interesting state-of-the-art methods of MVS reconstruction. At first we present a general taxonomy of MVS methods. Later we focus on two selected reconstruction algorithms in details. One of them produces dense point clouds, and the other - triangle meshes.

3.1 Overview and Comparison of MVS algorithms

As the number of different multi-view reconstruction methods is growing and algorithms are being improved, it was necessary to provide benchmark datasets in order to compare different methods. The first attempt to benchmark MVS methods was made in [SCD$^+$06], where the authors have provided a set of high quality calibrated images, that were registered with ground truth 3D models, and proposed an evalutation methodology for comparing those methods. They have established the Middlebury Ranking for MVS and optical flow methods, where other authors can submit their results for comparison. The MVS methods were categorized according to the following fundamental properites:

- **Scene representation.** Reconstruction algorithms use voxels, level-sets, polygon meshes, depth maps. Sometimes different representations are used in different steps of a reconstruction algorithm.
- **Photoconsistency measure.** There are several methods for measuring a visual quality of reconstructed surface. Many methods compare pixels in different images to check the level of correlation. The photoconsistency methods can be defined in *screen space* and *image space*. The first type project 3D points, patches or volume of geometry to two or more images, and check the agreement between the projections. The second type of photoconsistency measure wraps image from one viewpoint using the estimated geometry, and predicts a different view. The predicted view is compared to the measured view.
- **Visibility model.** There are different approaches to handling visibility. The *geometric* approach models the image formation process and uses the shape of a scene in order to determine which parts of the scene are visible in which images.

Quasi-geometric techniques use approximate geometric reasoning to estimate visibility relations. The *outlier-based* methods treat occlusions as outliers. Some methods use outlier rejection techniques to select good views.

- **Shape prior.** In regions, where high frequency textures are not available, the photoconsistency itself is not sufficient to recover precise geometry. Often the assumption on the smoothness of a surface is made. Some methods are trying to reconstruct surfaces with minimal area. Some space carving techniques extract the maximum area surface by removing non-photoconsistent voxels from an initial volume.

- **Reconstruction algorithm.** There are four main classes of reconstruction algorithms. (1) Algorithms from the first class compute a cost function on a 3D volume, and then extract the surface from this volume. Voxel coloring and its variants perform a single sweep through the volume, compute a cost function, and reconstruct voxels with a cost below a threshold. (2) Other algorithms iteratively evolve the surface in order to decrease a cost function. Methods of this class operate on voxels, level sets and surface meshes. (3) There are image-space methods, that compute depth maps. The depth maps are merged at a later stage of the algorithm or consistency constraints are enforced during execution of the algorithm. (4) Algorithms from the last class extract and match a set of features. After the features are reconstructed in 3D, a surface fitting is performed.

- **Initialization requirements.** Many space carving variants and level set algorithms depend on a rough bounding box to be provided with a set of calibrated images. Other algorithms require background separation masks for the input images in order to reconstruct a visual hull. A disparity range or allowed disparity values are used in image-space algorithms. In this case, the scene is constrained to be located in between near and far plane of each camera.

The references to particular methods mentioned above can be found in the original paper [SCD$^+$06]. In the further sections we first describe a simple reconstruction algorithm. Then we show two of the most interesting methods, one of them being under the top performing methods from the Middlebury database.

3.2 Simple Reconstruction Algorithm

In Chapter 2 we have introduced a basic reconstruction algorithm. In Section 2.3.4 we have shown, how to reconstruct a set of 3D points from image correspondences and camera matrices. The same approach we can apply for a reconstruction of a dense 3D point cloud. The key problem is to find a large set of image correspondences. After this step, it is enough to apply a triangulation, in order to obtain points in 3D space. One of the possibilities of calculating a dense set of correspondences is to use an *optical flow* algorithm. The dense optical flow algorithms output a vector field, where for each pixel in the source image, the corresponding pixel is located in the target image. The implementation of this reconstruction algorithm is very straight-forward, as many implementations of optical flow algorithms are available, for example in the OpenCV library [Bra00]. An example reconstruction from two views using optical flow is shown in Figure 3.1.

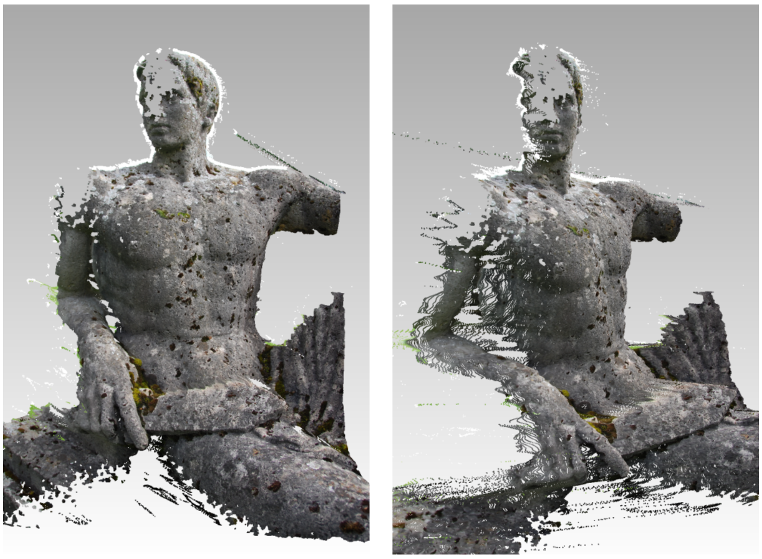


Figure 3.1: Simple reconstruction with optical flow.

Often optical flow algorithms cannot output correct correspondences. In low-frequency areas of images, the number of possibly correct solutions is very large. This leads to a number of outliers in the 3D point cloud.

There are methods that are using more images, in order to verify and improve the optical flow solution to some extent. An interesting algorithm, *depth map fusion* is introduced in [HZL+11].

3.3 Patch-Based Multi View Stereo

One of the top performers on the Middlebury Database [SCD+06] is the method described in [FP10]. The algorithm presented in this paper outputs a dense patch cloud. In this section, we briefly present this method. The algorithm takes a set of calibrated images as input, and as output it produces a set of points, normal vectors at that points, images in which the points are visible, and the image coordinates of the patches. There are three basic steps of this algorithm:

1. Generation of an initial point cloud.
2. Expansion of the point cloud.
3. Outlier filtering.

The two last steps are repeated three times in order to obtain a dense reconstruction. The author can recover a triangle surface for the patch cloud, however the reader is advised to consult the original paper in this matter. In this thesis, we present another approach for triangle mesh reconstruction in Section 3.4.

Generation of an Initial Point Cloud

In the first step of the algorithm, all images are divided into a coarse regular grid of 32×32 pixels. In each cell, a set of Harris corners and Difference of Gaussians (DoG) blobs is detected. This is done to obtain an uniform coverage of the image with features. Detected features are matched along the epipolar lines, and triangulated in order to obtain a (sparse) set of 3D points. The matching is done in the following way. For each feature f, a set of corresponding features of the same type is found. This creates several potential patches in 3D. The potential patches are considered in order of increasing distance from the optical center of the feature f. The first patch that is photoconsistent in at least k images is kept, and the rest is discarded. Each patch obtained in this way is subjected to the optimization routine in order to improve the photoconsistency. The variables optimized in this process are a patch position and a normal vector. This procedure gives a stable sparse 3D point cloud. Next, the algorithm iterates over the expansion and filtering steps described in the next paragraphs.

Expansion of the Point Cloud

For each image, the algorithm creates a regular grid C with cells of size 1×1 or 2×2 pixels. Some of the cells are already occupied with patches reconstructed from features, that project to those cells. The goal of the algorithm is to reconstruct patches that project to a large number of the cells in order to create a dense model. Some patches will be incorrect and removed in the filtering step.

The expansion step is used to add new neighbors to existing patches. Two patches are considered as neighbors when they are stored in adjecent cells of the grid C of the same image, and have similar tangent planes. If we already have a patch p in the grid cell, new patches are created in the neighboring cells using the following procedure. Rays from the optical center passing through empty neighboring cells are shot, and intersected with a plane defined by the normal vector of the patch p already associated with the grid. This intersection point is a location of a new patch p'. The normal vector for that patch is copied from p. Then the new patch is optimized in order to improve the photoconsistency.

Outlier Filtering

The filtering step is used to enforce visibility consistency and remove wrong matches. Two fiters are applied. The first one removes patches laying outside of the original surface. This is done by checking if the patch is occluded by stable patches or is occluding stable patches. The second filter removes wrong patches that are laying on the surface, but the photoconsistency cannot be verified in at least a certain number of images. Iterating over the expansion and filtering steps create dense point clouds of a high quality.

3.4 Reconstruction with Graph Cuts

In this section, we describe an alternative MVS method presented in [LPK07] and [HKLP09], that can produce triangle meshes by design. There are four main steps of the algorithm: (1) semi-dense point cloud generation, (2) match aggregation and Delaunay triangulation, (3) surface extraction, (4) surface optimization. These steps are described in the following subsections.

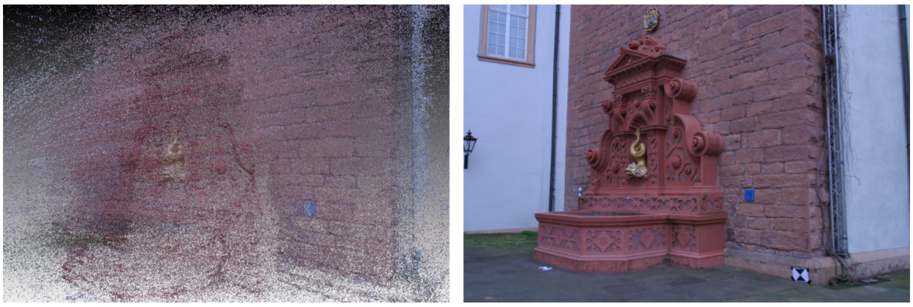


Figure 3.2: Semi-dense point cloud (left) and one of the input images (right).

Semi-dense Point Cloud Generation

In this method, first the SIFT and DoG features are extracted. The features are matched along epipolar lines in all matching pairs of images. The false positive matches are also included, as the method can deal with outliers very well, as it is shown later. The matches are used to compute a semi-dense point cloud by triangulation. The semi-dense 3D point produced by our implementation of this algorithm together with one of the input images is illustrated in Figure 3.2

Match Aggregation and Delaunay Triangulation

In this step, a 3D Delaunay triangulation of a point cloud is performed. It is done in an incremental way. The authors start by inserting one point into an empty structure. Points are instrted one by one. Before inserting the next point, the nearest neighbor is found in the structure. If the differences between the 2D coordinates of the two points after projection to the respective images are small, the 3D points are merged. In the other case, the new point is inserted to the Delaunay tetrahedralization. Each point in the triangulation structure has a list of images it appears in. The lists from aggregated points are merged together. Usually outliers does not aggregate, so in this case, the image lists are very short. At the end of this process, the full 3D structure consisting of tetrahedra (formed by triangles) is prepared for the next step - surface extraction.

Surface Extraction

The surface extraction is done by labeling each tetrahedron as inside or outside. If two neighboring tetrahedra sharing a triangle have different labels, then the shared triangle is a part of the surface. The surface extraction algorithm extracts all triangles between tetrahedra with different labels, as the final mesh.

A globally optimal labeling can be found with a graph cut algorithm. The tetrahedral structure is converted into a graph in the following way. Each tetrahedron becomes a graph node. The nodes corresponding to tetrahedra that are sharing a triangle are joined with edges. The graph cut algorithm will operate directly on this graph.

The surface S that is to be reconstructed is an union of triangles between neighboring tetrahedra with different labels. The reconstructed surface should minimize the energy function consisting of three components - visibility, photoconsistency and smoothness terms:

$$E(S) = E_{\text{vis}}(S) + \lambda_{\text{photo}} E_{\text{photo}}(S) + \lambda_{\text{area}} E_{\text{area}}(S) \tag{3.1}$$

where λ_{photo} and λ_{area} are positive weights. Each energy term is implemented in the graph cut framework.

Graph Cut

In the graph theory, a *graph cut* is a partition of graph nodes into two disjoint sets. The *size* of a cut is the number of edges connecting nodes from different sets (in case of an unweighted graph) or the sum of weights assigned to those edges.

In flow networks, often the term *s-t cut* is used. A flow network is a directed and weighted graph. Each weight describes a capacity of an edge that receives a flow. The flow network has two special nodes - the *source* s and the *sink* t. The flow in the network originates from the source and is distributed through the edges to the sink. The s-t cut $C = (S, T)$ of a network $N = (V, E)$ is a cut of N such that $s \in S$ and $t \in T$, where s and t are the source and the sink.

The minimum cut problem for graphs is to find the cut of a graph of the smallest size. A similar problem for flow networks - *the maximum flow* - is to find flow values for the edges, that are not exceeding the capacities, and fulfill the flow conservation principle. The flow conservation principle states that the sum of incoming flows is equal to the sum of outgoing flows for each node, not counting the source and the sink.

It has been proven already in 1956, that in a flow network, the maximum amount of flow passing from the source to the sink is equal to the size of the minimum cut of the flow network. The details about this theorem can be found in [Wik12c]. This is an interesting finding, because the minimum s-t cut problem can be transformed into the maximum flow problem. In the paper, the minimum s-t cut problem is solved by a maximum flow library.

The surface extraction is done by assigning proper weights to the egdes in the tetrahedra graph and solving the minimum s-t cut problem. Nodes (tetrahedra) in the source set are labeled as inside, and nodes in the sink set are labeled as outside. After labeling, extracting the surface is straightforward. In the following subsections, we describe the weight assigment corresponding to the energy function components.

Visibility Term

Each 3D point in the Delaunay triangulation has the information, from which images it has been reconstructed. Some close 3D points reconstructed from different views are merged, together with the visibility information. The authors use the point visibility information to establish the global visibility error function.

If a point belongs to a final surface, it should be visible in the corresponding image. As a consequence, all the tetrahedra intersected by a ray from the point to the corresponding camera center should be labeled as outside.

This observation leads to the following weight assignment. A source node and a tetrahedron p_0 containing a camera center gets a large weight $w_{s,p_0} = \lambda_\infty$. If a ray emmited from a 3D point, exiting tetrahedron q_1 enters its neighboring tetrahedron p_1 through a shared triangle, the weight $w_{p_1 q_1}$ is set to λ_{out}. The tetrahedron q_2 located behind the 3D point (in the opposite ray direction) is connected to the source node with a weight $w_{q_2 t} = \lambda_{\text{in}}$

The weights for all the rays from all the points to the corresponding cameras are accumulated in the edges of the tetrahedra graph. Additionally, the edges accumulate photoconsistency and smoothness terms described below.

Photoconsistency Term

The photoconsistency term $E_{\text{photo}}(S)$ is a measure how well the surface S matches different images, in which it can be seen. The surface photoconsistency is an integral of a photoconsistency measure $\rho \geq 0$ over the surface S and is defined in the following way:

$$E_{\text{photo}}(S) = \int_S \rho \, \mathrm{d}S = \sum_{T \in S} \rho(T) A(T) \tag{3.2}$$

where T is a triangle, $A(T)$ is the triangle area. The photoconsistency is computed only in images, from which all three vertices were reconstructed.

The mapping of the surface photoconsistency to the graph cut framework is done as follows. For each pair of tetrahedra (p, q) which share a triangle T, with normal $\mathbf{n}$ pointing from tetrahedron p to tetrahedron q, an edge (p, q) in the graph receives a weight w_{pq} equal to the photoconsistency ρ of the triangle T.

Smoothness Term

The smoothness is enforced by minimizing the area of a surface:

$$E_{\text{area}}(S) = \int_S \mathrm{d}S = \sum_{T \in S} A(T) \tag{3.3}$$

Translation of this energy term to the graph cut framework is done by adding a weight to each edge, equal to the area of a triangle between a corresponding pair of neighboring tetrahedra. The contribution of the triangle area is done for the corresponing edges in both directions.

Surface Optimization

Several improvements over the initial graph cut based method [LPK07] were proposed in [HKLP09]. A denser initial point cloud is computed by using also different types of features. A photoconsistency criterion based on sum of normalized cross correlations for different fixed window sizes is used to reduce the number of false matches. The most important improvement over the previous work is the mesh optimization procedure.

The 3D points of an initial dense point cloud are often not precise, as it would require to reconstruct them from very precise sub-pixel coordinates of the features. Additionally the vertex merging can introduce some disturbances to the point positions. As a consequence, the output mesh is noisy.

The energy function used to optimize the surface is based on the *reprojection error*. The reprojection error is the difference between the original photo k and the image obtained by projecting image k' to the surface and rendering the surface to the camera k. This error is summed for all the close camera pairs. Additionally a regularization term is added in order to enforce surface smoothness.

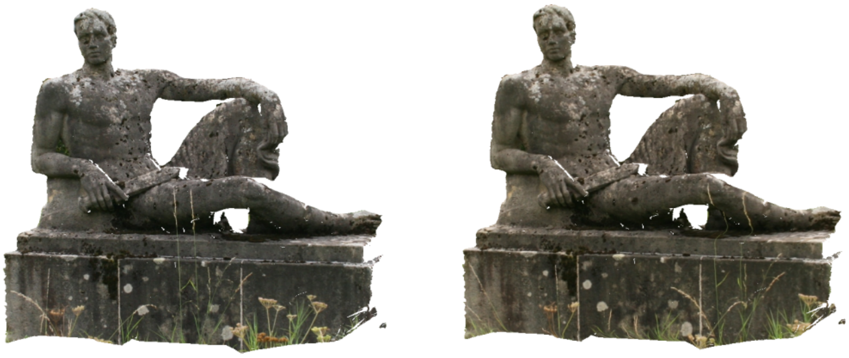


Figure 3.3: Comparison of the original and artificial image. The left image is a photograph with an alpha mask applied, and the right image is an artificial rendering of a reconstructed mesh with a reprojected texture from another camera.

The graph cut based surface reconstruction is described in detail in previously mentioned publications. The author of this thesis made his own implementation of this algorithm, and the results are presented in Figure 3.4. This figure shows two example images, initial meshes obtained by graph cut, and meshes after the optimization. The original image with an alpha mask applied and an artificial rendering of the final model with a nearby camera's image projected onto the surface is shown in Figure 3.3.

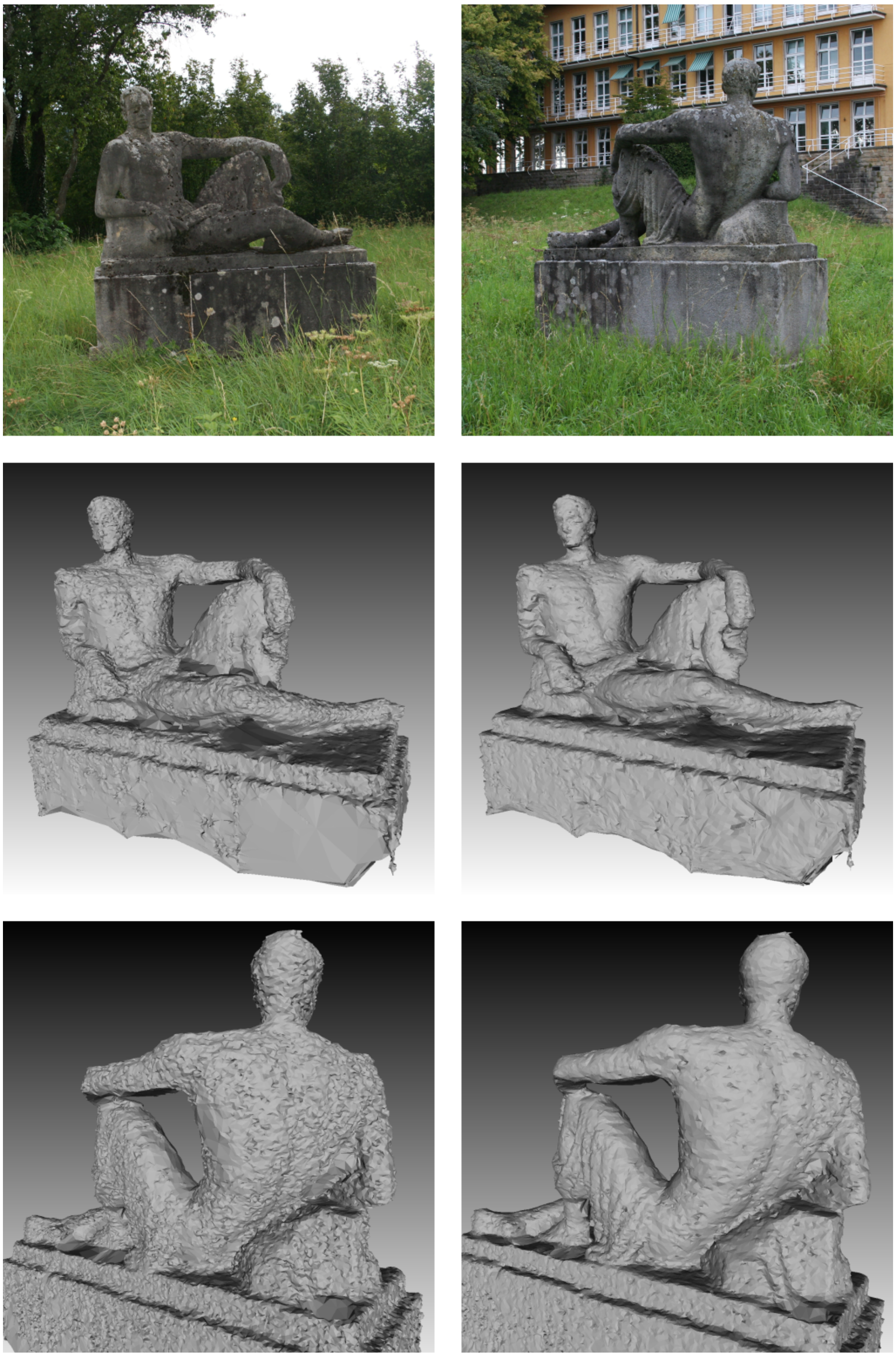


Figure 3.4: Surface reconstruction. Some of the input images (top row), the reconstructed mesh (two bottom rows), mesh before optimization (left column), mesh after optimization (right column).

Chapter 4

Compact Descriptors for Video Sequence Matching

4.1 Introduction

The challenge in camera calibration is computing camera parameters in large scenes, where images are unorganized. As the database of images grows, the computational complexity of finding similar images is becoming a problem. Camera calibration algorithms often consider images as nodes of a graph, where edges encode geometric similarity between images. This is known as a similarity graph. A naive method of the similarity graph construction would be to consider all possible pairs. The basic operation in this case is computing similarity between two images, and a number of those operations is $O(n^2)$. No matter how fast this basic operation is, the large number of images makes it practically impossible to process the whole set. Efforts of decrease the number of comparisons exist, but very often there are limitations, like an image database has to be known in advance, and extending the database with new data is often not practical. An overview of these efforts is given in Section 4.2.

In this chapter, we present our *Compact Similarity Descriptor*, that is used for a quick construction of the similarity graph, and exploits partial organization of the input data. A basic building block used in our method is a set of images, or a short video sequence. In this case the image relations within sets are known, or can be computed efficiently due to limited data size. The work described in this chapter has been published in [PLS12].

For Internet photo collections of cities, we usually do not have the assumption of known image relations within small sets, as images often are completely unorganized. In most of the cases, the coverage of a scene is limited to the most prominent sites, what is not sufficient to build the whole scene. In this case, the data acquisition for a full scene must be performed, and during the acquisition, a partial organization of data can be easily obtained.

During the acquisition of the image data for a city, recording time stamps and GPS coordinates is sufficient for data pre-organization. With this information, we can partition the whole data set into smaller subsets depicting the same geometry. The Compact Similarity Descriptor is used to quickly compute a similarity measure and individual image relations between different subsets of images. The speed of our algorithm is comparable to the simple two frame matching algorithm presented in the classical

SIFT paper ([Low04]).

In this chapter, we often use the term *image sequence*, however the algorithm does not assume any ordering of images within one subset. In the case of video sequences, we work on a subset of key frames with a good contrast and sufficient baseline.

4.2 Related Work

In this section, we present the work that is related to problems commonly encountered in the preliminary stage of MVS reconstruction - finding matching images. In case of a large scale reconstruction the number of images required for the scene coverage is excessive, so using a camcorder to produce video sequences may be desired. In case of video sequences, most often consecutive frames depict the same geometry, so a lot of computation can be saved when exploiting this information. However when more video sequences cover the same geometry, finding similar frames between two different sequences is no longer given for free. This leads us to research on matching algorithms for video sequences.

4.2.1 Video Sequence Matching

There have been many approaches used for finding similar video streams ([HHB02], [KC05], [KP02], [CS08], and others), however in these cases, the frame ordering is a key factor. This does not fit into our scenario, where a sequence often is an unordered set of digital photographs mainly used for multi-view stereo reconstruction.

Another approach for finding similar videos, proposed in [YC09], is using an adaptive vocabulary tree to index all video frames in a database, and each sequence is treated as a ”bag of frames”. The authors use global image features in order to reduce memory and computational requirements. This approach works well in a context of copy detection, however in our scenario, global descriptors are not desired, as they often do not detect matching images useful for reconstruction. Additionally, the problem of matching videos is formulated as local alignment problem, what is not suited to our scenario.

4.2.2 State-of-the-Art Image Matching Algorithms

SIFT Features

One of the most important approaches for image matching is Scale Invariant Feature Transform (SIFT) described in [Low04]. The author comes up with a selection of distinctive image features, that can be robustly matched with features of another image, under different scales and rotations. The features are also partially invariant to illumination changes and changes of 3D camera viewing angles. This is very important for the task of MVS reconstruction, as we need different camera positions by definition, to solve the stereo problem. An additional desired property of SIFT features is high distinctiveness - a single feature can be correctly matched against a large feature database with a high probability.

The algorithm creates a pyramid of filtered images. In this multi-scale approach, more expensive operations are executed only in image locations, that pass an initial test. There are four major steps in the extraction of SIFT features from images:

1. **Scale space extrema detection.** In this stage, a difference of Gaussians (DoG) is computed between all image levels. Potential points of interest are defined as minima or maxima of the DoG function.
2. **Keypoint localization.** At each initial location, a model is fit in order to determine location and scale. Locations that are not stable are disregarded from further computation.
3. **Orientation assignment.** Local image gradients are used to determine one or more orientations for each keypoint location. In later steps, all operation on features are performed relative to the orientation, scale, and location of the feature.
4. **Computation of the keypoint descriptor.** First the local image gradients are computed at selected scales around each keypoint. Then they are transformed into a robust representation, that allows for local illumination and shape distortions.

The SIFT feature matching algorithm can be thought of as a nearest neighbor search algorithm. Each SIFT feature is represented as a 128-dimensional vector. Finding a matching feature turns down to finding the nearest feature in terms of Euclidean distance. In practice, in order to guarantee an uniqueness of the matching features and limit the number of false positives, also the second nearest feature is found. The two nearest neighbors of the query feature have distances d_1 and d_2. The feature match acceptance or rejection is based on a ratio $\frac{d_1}{d_2}$. The rejection is made, if the ratio is larger than 0.8.

In order to match two images, it is necessary to extract image features first. Two feature sets F_1 and F_2 are obtained. The matching algorithm takes each feature $f \in F_1$, and searches for the nearest and the second nearest feature in the set F_2. If the distance ratio criterion is fulfilled, then the match is accepted. A sufficient number of matches means that the images are potentially matching, however the geometric verification of feature locations is desirable. The most commonly used approach for nearest neighbor search uses kd-trees. The Approximate Nearest Neighbor Library (ANN) is used in different implementations very often.

SURF Features

There were approaches to optimize the speed of SIFT. The Speeded-up Robust Features (SURF) algorithm presented in [BETVG08] is an successful attempt to improve the speed of feature extraction. The SUFT feature extraction algorithm has been also implemented on Graphics Processing Units (GPUs).

The detector is based on the Hessian matrix, in order to detect a blob-like structures in places, where the determinant of the matrix reaches the maximum. In the computation of the Hessian matrix, the second order Gaussian derivatives are approximated with box filters. Box filters can be computed very fast using integral images. Each entry of the integral image $I_{\sum}(\mathbf{x})$ for image I, at the location (x, y) is defined by:

$$I_{\sum}(\mathbf{x}) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{i \leq y} I(i, j) \tag{4.1}$$

The integral image entry for a pixel $\mathbf{x} = (x, y)$ is a sum of all pixel intensities in the rectangle $[0, 0, x, y]$. This representation allows for a quick computation of intensity sums of any rectangular areas, independent of their size.

There are many open source implementations of SURF, the list can be found in [Wik12d]. For further details about the algorithm itself, the reader is referred to the original paper [BETVG08].

Casting Features to Lower Dimensions

SIFT features have been found to be effective and stable in many applications. Unfortunately the descriptor is 128 dimensional. In case of large image databases, this can account for high storage requirements. For example, storing 10000 SIFT features, requires around 1MB of space per image (8 bits per dimension). Additionally the cost of nearest neighbor searches is increasing with the number of dimensions [Yia93]. Therefore, there are several approaches to reduce the feature descriptor dimensionality.

One of the approaches for dimensionality reduction called *spectral hashing* is described in [WTF08]. In the paper, the authors design a compact coding for feature descriptors, that preserve distance ratios between highly dimensional vectors in space of reduced dimensionality.

Improvements of the matching speed for high dimensional vectors by casting them to a space of lower dimension were presented in [RL09]. The authors of this publication a design mapping from high dimensional vectors to short binary strings. The coding scheme is mapping close vectors to binary strings with small Hamming distance.

A modification to the original SIFT algorithm termed PCA-SIFT is presented in [KS04]. The authors use the Principal Component Analysis, a standard technique for dimensionality reduction ([Jol86]), to reduce the descriptor size of SIFT features. However, it is not the final descriptor, that is subjected to PCA. The authors modify the last step of the SIFT algorithm - they replace the SIFT's smoothed weighted histograms with vectors obtained by PCA applied to the normalized gradient patches. The original descriptor dimension (128) is reduced to 36, but instead of using 8 bits per vector entry, they use 16.

4.2.3 Bag-of-Words Approaches

In Bag-of-Words techniques, the problem of finding similar objects or images in sets of other images can be recast as a text retrieval problem. It is approached by creating a visual analogy of a word. In most of the publications, visual words are obtained by vector quantization of the descriptor vectors.

In text search engines [ZM06], a document is represented as a set of words. The words are contained in a dictionary. Let's consider a classical example from Wikipedia [Wik12a]. We have two text documents, and we want to compute the similarity between them:

- ”John likes to watch movies. Mary likes too.”
- ”John also likes to watch football games.”

We define a dictionary over those two documents:

- dictionary={1:'John', 2:'likes', 3: 'to', 4:'watch', 5:'movies', 6: 'also', 7: 'football', 8:'games', 9:'Mary', 10:'too'}

This dictionary has 10 words. We can represent the two documents as 10-dimensional vectors:

- $[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]$
- $[1, 1, 1, 1, 0, 1, 1, 1, 0, 0]$

Each entry in those vectors is the count of corresponding words from the dictionary. This is called the histogram representation. The similarity measure between two documents in this representation can be computed using an Euclidean distance.

This representation is employed in Computer Vision, where an image can be treated as a document. In this case, an image requires a 'word'-based description. Usually this is done by feature detection, computation of feature descriptors, and creation of a code book. Then each image is described as a set of code book vectors.

In the next paragraphs, we present state-of-the art methods, on which other image retrieval methods are based.

The work presented in [SZ03], makes it possible to search and localize user-marked objects in video sequences. The word-based description relies on two types of features representing regions of a video.

- Shape Adapted (SA) regions. They are based on ellipses parametrized by a center, a shape and a scale. Their scale is computed from a local extremum of a Laplacian, and shape is computed by maximizing an intensity gradient isotropy over elliptical regions.
- Maximally Stable (MS) regions. They are constructed by selecting areas from a watershed image segmentation and represent regions with stationary areas of variable intensity.

Both types of regions are represented with ellipses. The two types of regions are used, because they detect two different image areas. The SA regions are centered around corner-like structures, while the MS regions correspond to blobs of high contrast with respect to their surrounding.

For each elliptical region, a 128 dimensional SIFT descriptor is used. This provides invariance to small shifts in the region positions. Regions are tracked over several frames, and those that cannot be tracked across at least three frames are discarded.

In order to build a dictionary, about 10% of all video frames were carefully selected. From those frames, regions were extracted and clustered by the k-means algorithm. Two types of regions are clustered separately, as they cover different and independent areas in the video frames. For the clustering, the Mahalanobis distance is used, because it allows to assign less weight to the noisy components of the SIFT descriptor and to decorrelate the components.

In the text retrieval methods, each document is represented by a vector of word frequencies, however a weighting of the components of this vector is often preferred. The standard weighting is called *Term Frequency - Inverse Document Frequency* (TF-IDF), which works as follows. If we have a dictionary of k words, then each vector is described as $\mathbf{V}_d = (t_1, ...t_k)^T$ of weighted word frequencies:

$$t_i = \frac{n_{id}}{n_d} \log \frac{N}{n_i} \tag{4.2}$$

In this equation, n_{id} is the number of occurrences of word i in document d, n_d is the total number of words in document d, n_i is the number of occurrences of term i in the whole database, and N is the total number of document in the whole database. This weighting scheme is a product of two terms: the *word frequency* n_{id}/n_d and the *inverse* *document frequency* $\log \frac{N}{n_i}$. The first part gives more weight to words occurring more often in the document, while the second part puts lower weights on words occurring often in the database.

The retrieval is done by computing a normalized scalar product between a query vector V_q and all other documents V_d in the database. In the paper [SZ03], the query vector is defined by visual words contained in a user-selected part of a frame, and other frames are ranked according to the similarity to this query vector. In order to speed up the querying process, all visual words are kept in a structure called the *inverted file*. In this structure each word has an entry storing all the documents, where it occurs.

Another approach to image search in large databases is presented in [NS06]. This work is inspired by the work described in [SZ03], however the database organization is different, as well as an application of the algorithm. Instead of a plain visual word database, the authors use a vocabulary tree. The local region descriptors are quantized in a hierarchical manner. This enables to use a larger and more discriminatory vocabulary in an efficient way. As the experiments have shown, a much better retrieval quality was obtained, when compared to standard methods.

In this approach, a vocabulary tree is constructed in the following way. First, the whole set of visual words is clustered to k clusters with the traditional k-means algorithm. Then the clustering is recursively applied to each of the visual word clusters. This procedure is repeated in order to obtain L levels in the hierarchy.

The search is done in the following way. At each level k dot products are performed, giving kL dot products in total. In the experiments, the authors use 6 levels and 10 clusters at each level. When a query for a single visual word is performed, the search path down the tree is recorded. The relevance of the query image and a database image is evaluated by comparing how similar the paths for the feature descriptors are. The weights assigned to a cluster center on each level are based on the TF-IDF scheme. The authors demonstrated real time query speeds on an image database of 50 000 CD covers, and timings around one second were demonstrated on a 1M image database.

4.3 Compact Descriptor and Matching

The key observation is that an image sequence showing the same 3D scene has many common image features visible in different photographs of the sequence. Due to the robustness of SIFT, different viewing angles and distances to the scene have a small influence on SIFT feature descriptors. This fact can be immediately exploited, by using features occurring multiple times in different images of a sequence, just once.

In the following subsections, we describe our similarity descriptor, the method of descriptor matching, and the search algorithm for finding individual matching images among two different image subsets.

4.3.1 Computation of Compact Descriptor

In order to build a descriptor for an image sequence, first, we extract SIFT features from all images. It is possible to limit the number of features by adjusting the contrast threshold. This step is optional, however for plain similarity measurement, it may be worth to limit the input data size for later processing, as it can save computation time for longer sequences.

All features from all images or key frames in a sequence are collected, and then clustered with the standard k-means algorithm. With this approach, we exploit the fact of having very similar features in many images belonging to the same sequence.

The *compact descriptor* $\mathbf{C}$ is a set of cluster centers $\mathbf{C}_j \in \mathbb{R}^{128}$ of all features from the sequence:

$$\mathbf{C} = \{\mathbf{C}_j\}_{j=1}^{k} \tag{4.3}$$

The size of the compact descriptor is $k * 128 *$ sizeof(float), what in most of our tests gives 2048 kilobytes for 4096 cluster centers.

4.3.2 Computation of Similarity Measure

The similarity measure computation can be reduced to the standard SIFT feature matching algorithm, as presented in the original publication [Low04]. We can do this, because a cluster center can be considered as a single SIFT feature, that is an average vector of all features that it represents, in an image subset.

In the original paper, matching of two sets of features is done as follows: for each feature in the first set, search for the nearest and the second nearest neighbor in the second set. If the ratio of the first nearest neighbor to the distance to the second nearest neighbor is less than a threshold t, then the match is accepted. The choice of $t = 0.8$ was able to eliminate 90 percent of false matches, while discarding less than 5 percent of correct matches. Directly from the original algorithm, we can deliver the following similarity measure between two sequences S_a and S_b:

$$\sigma(S_a, S_b) = \#\{c \in \mathbf{C}_a | \frac{dist(c, NN_1(c, \mathbf{C}_b))}{dist(c, NN_2(c, \mathbf{C}_b))} < t\} \tag{4.4}$$

what is just a number of matches between cluster centers of two descriptors.

This simple algorithm has proven to be robust in the case of SIFT image feature descriptors, however in case of matching k-means centers, the following problem, illustrated in Figure 4.1, may occur. Due to an insufficient amount of centers used in k-means, one center can represent more than one real cluster of features. A natural consequence of this fact is, that the k-means center is shifted to a location influenced by other centers of gravity of real feature clusters. When matching with the k-means centers from another sequence, the ratio of the first nearest neighbor to the second one may exceed the threshold, with the consequence of being rejected, despite the fact, that the real centers of gravity for clusters of features should fulfill the acceptance condition. In this case, for long sequences with relatively small descriptors, it may be necessary to relax the original condition in order not to throw away close clusters that may contain matching features. We have analyzed plots of the distance between cluster centers together with associated ratios in case of matching and non-matching

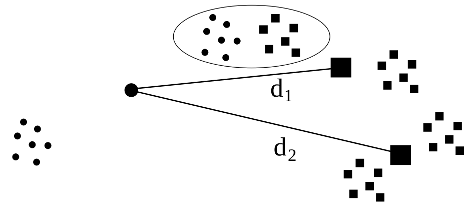


Figure 4.1: Example configuration of real feature clusters and k-means centers for two image sequences, where the original condition for SIFT feature matching (involving the ratio of d_1 and d_2) may fail.

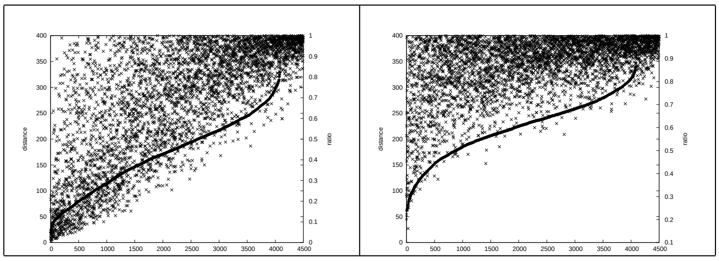


Figure 4.2: Plots of distances and ratios for a matching sequence (left) and non-matching sequence (right). The X axis shows match numbers, the continous line represents match distances, and points spread throughout the plot are ratios associated with the matches. In the matching case, a large number of ratios is below selected threshold value 0.5, while in the non-matching case, the majority of ratios is above 0.5.

sequences. It can be clearly seen, that even for close distances, some of the matches have high ratios. We have obtained a good similarity response with the threshold of 0.5 for the ratio of squared distances to the nearest and the second nearest cluster center. Plots for exemplary matching and non-matching sequences, are shown in Figure 4.2.

4.3.3 Computation of Image Occurrence Statistics

As the descriptor itself is sufficient for computing the similarity measure, we are also interested in identifying individual frames that match between image sequences. This is essential for the following purposes:

- Adding single frames to image sequences for obtaining camera calibration parameters, with respect to an already calibrated sequence.
- Recognizing 3D points corresponding to 2D feature points for computing transformations between reconstructed 3D sub-models.
- Adding additional constraint for 3D model alignment and loop closure.

In order to efficiently select pairs of matching frames from different subsets of images, we extend each cluster center $\mathbf{C}_j$ of the compact descriptor with a set D_j:

$$D_j = \{I_i\}_{i=1}^{k} \tag{4.5}$$

where I_i is the image number that fulfills the following condition:

$$\exists_{f \in E_j} f \in F_i \tag{4.6}$$

where E_j is a set of image features associated with $\mathbf{C}_j$, and F_i is a set of features associated with image I_i.

In practice, the computation of sets D_j is done by gathering features associated with center $\mathbf{C}_j$ and constructing the set of images, from where the features came from. This can be done in the last iteration of the k-means algorithm.

4.3.4 Computation of Matching Frames

Matching frames are computed from the image statistics described in the previous subsection. During the similarity computation, for each k-means center c_i^1 in the

Algorithm 1 Computation of Matching Frames

```
function ComputeImageMatches($M$, $S^1$, $D^1$, $S^2$, $D^2$, cnt)
    $n \leftarrow$ number of images in $S^1$
    $m \leftarrow$ number of images in $S^2$
    allocate array $A$ of size $n \times m$, set to zeros
    for $i = 1 \rightarrow |M|$ do
        $(c^1, c^2, d, r) \leftarrow i$-th 4-tuple of $M$
        for all $d_1 \in D^1_{c^1}$ do
            for all $d_2 \in D^2_{c^2}$ do
                $p_{ok} \leftarrow P(\text{match}(d_1, d_2)|d, r)$
                $p_{nok} \leftarrow P(\text{no-match}(d_1, d_2)|d, r)$
                if $p_{ok} > p_{nok}$ then
                    $A[d_1][d_2] \leftarrow A[d_1][d_2] + 1$
                end if
            end for
        end for
    end for
    return $\{(u^1_i, u^2_i)|A[u^1_i][u^2_i]\text{is } i\text{-th largest element of } A\}^{\text{cnt}}_{i=1}$
end function
```

first sequence, we have computed the nearest (c^2_i) and second nearest corresponding cluster center from another image sequence, obtaining a set of matches $M = \{(c^1_i, c^2_i, d_i, r_i)\}^u_{i=1}$, where d_i is the distance to the nearest center, and r_i is the ratio of squared distances to the nearest and the second nearest cluster center.

The computation of matching images between sequences S^1 and S^2 with corresponding descriptors D^1, D^2, whose elements are defined in Equation 4.5, is shown in Algorithm 1.

The idea behind this algorithm is as follows. When we have two matching cluster centers from different image sequences, there is a probability that the images associated with the first center are matching to the images associated with the second center, because they may have at least one common feature. Therefore, we consider all possible image pairs from the two matching centers, and apply a voting mechanism using the array A. Each pair increases an entry in this array depending on probability calculations detailed below. When we consider all matching cluster centers, the voting scheme will cause the most probable matching image pairs to emerge.

The probability function P, describes how probable it is that two images from corresponding cluster centers can be matched together. According to the Bayesian theory, this function can be expressed as

$$P(\text{match}(I_1, I_2)|d, r) = \frac{p(d, r|\text{match}(I_1, I_2)) \cdot P(\text{match}(I_1, I_2))}{p(d, r)} \tag{4.7}$$

where the evidence $p(d, r)$ is just a scaling factor, that can be omitted in the Bayesian decision rule. As the prior probability $P(\text{match}(d_1, d_2))$ is unknown, at this point we need to assume it equal with $P(\text{no-match}(d_1, d_2))$. The simplified decision rule now depends only on the likelihoods $p(d, r|\omega)$, where ω is the matching and non-matching class. In order to estimate those two density functions $p(d, r|\omega)$, the idea that immediately comes to a mind is to use a standard technique from statistics - kernel density estimation. It would be required to gather a lot of data about distances and ratios of matching and non-matching pairs, and estimate the probability density directly from this data. However, we used instead an extremely simplifying but effective approximation:

$$p(d, r|\text{match}(I_1, I_2)) = \begin{cases} 2, & r \in (0, \frac{1}{2}] \\ 0, & r \notin (0, \frac{1}{2}] \end{cases} \tag{4.8}$$

According to the above formula, the decision about considering a possible image pair as matching, in practice turns down to checking, if the ratio associated with matching k-means centers falls into the interval $(0, \frac{1}{2}]$. The choice of the interval is directly connected to the sequence matching threshold for k-means centers, as described in Section 4.3.2. This simple approximation would not be sufficient for a correct decision when a single image pair associated with two matching clusters is taken into consideration, however when used in the voting scheme, it provides stable results.

We remove the most frequently occurring image pair from the array A. This helps to filter out false positives from the query response in the context of image search databases, as it is suggested in [CPS$^+$07] and supported by our experiments. We do not check the geometric configuration of features within the images. This simplification can limit the accuracy of the algorithm, however any false positives are detected in the later stage of feature matching between proposed matching image pairs.

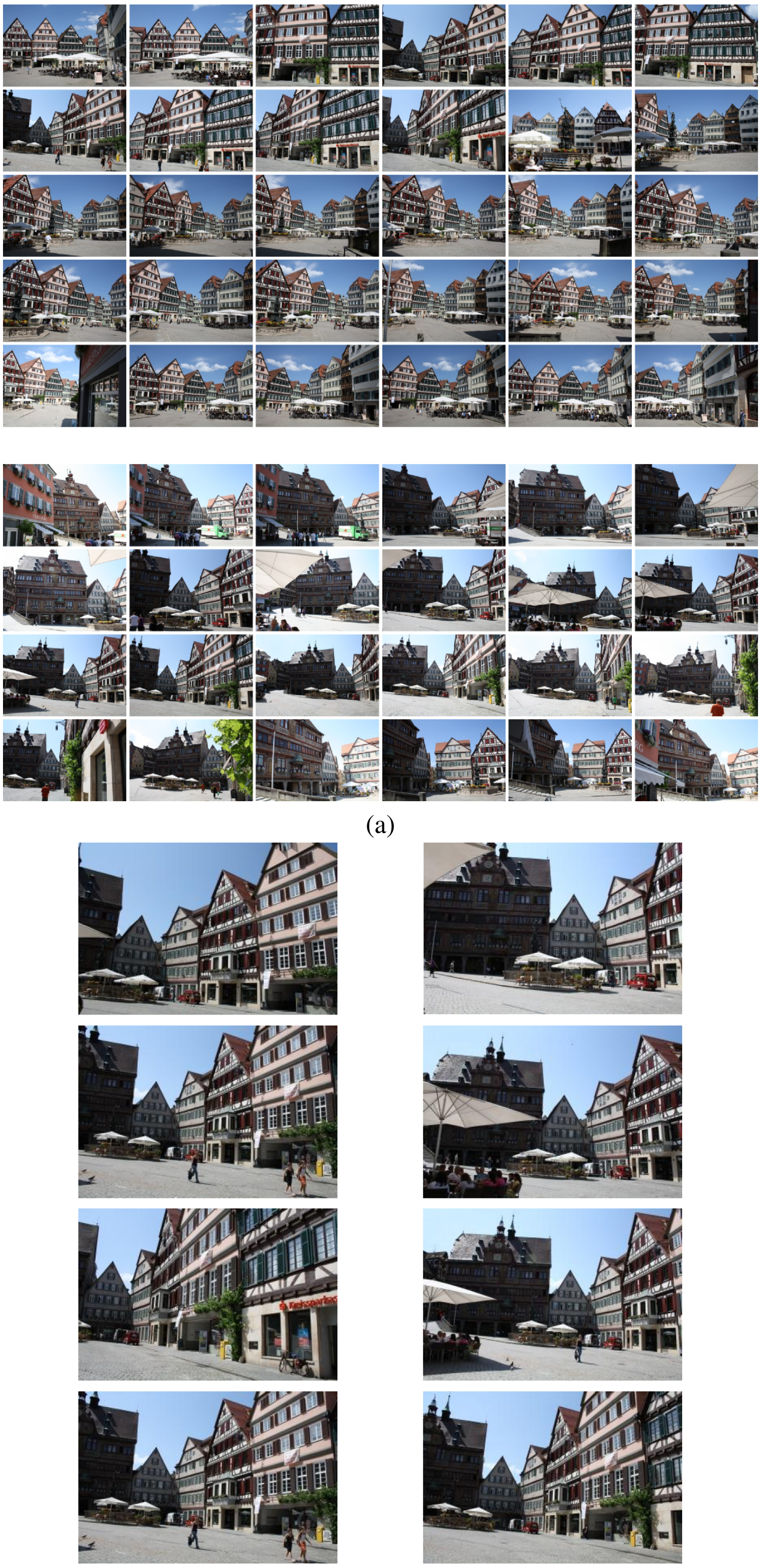
4.4 Results

We have tested our compact descriptor on many datasets. Example datasets are shown in Figures 4.3, 4.4 and 4.5 (a). In the same Figures (b), we show examples of matching pairs. Pairs containing repeating images from two sequences have been omitted for space saving reasons, and the most highly ranked pairs are shown. The resolution of images used in experiments is 10 megapixels, From each image, we use 1000 features with the highest contrast. Features are gathered and clustered to the descriptor size of 4096 k-means centers.

The time required for image sequence matching is comparable to SIFT feature matching between two images, and on a single core of an Intel Core Quad Q9300 running at 2.5 GHz, is approximately 5 seconds. No optimization or parallelization of the image sequence matching code has been performed yet. The creation of compact descriptors can be done nearly as fast as SIFT feature extraction, for the tested sequences of 45 images, it takes approximately 223 seconds (4 threads). In our experiments, we use the maximum of 32 k-means iterations. The descriptor creation time is much longer than the matching time, however it is created just once, and it is reused many times in the course of large scale reconstruction.

False positive matches are ranked low, and they have never appeared in the first 16 matching pairs used for assembling a final 3D model. However, if any false positive matches appeared, they would be discarded in the later stage of geometric verification of the reconstruction algorithm.

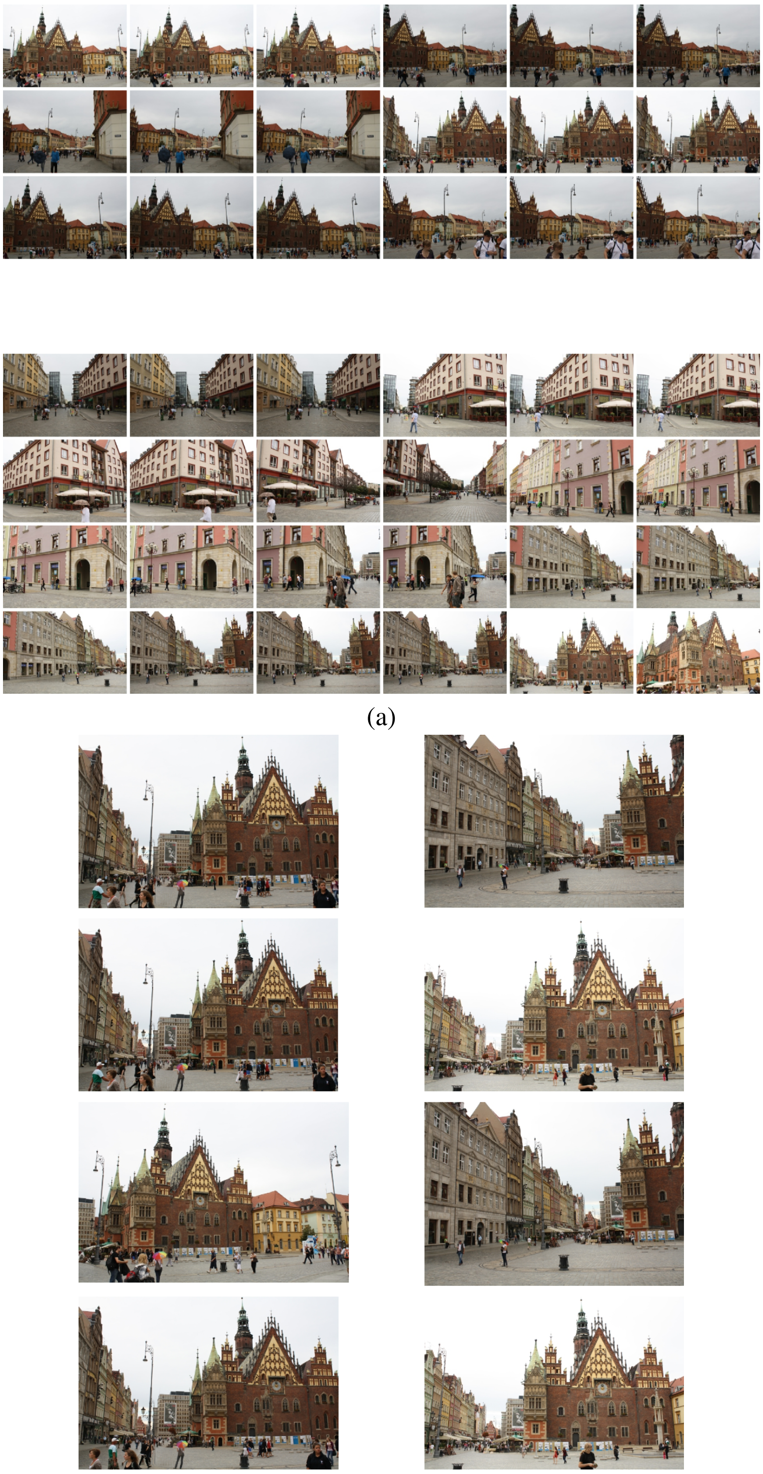
The algorithm for similarity descriptor matching has been proven in creating correct similarity graphs between image sequences used to merge reconstructed 3D submodels.



(a)

(b)

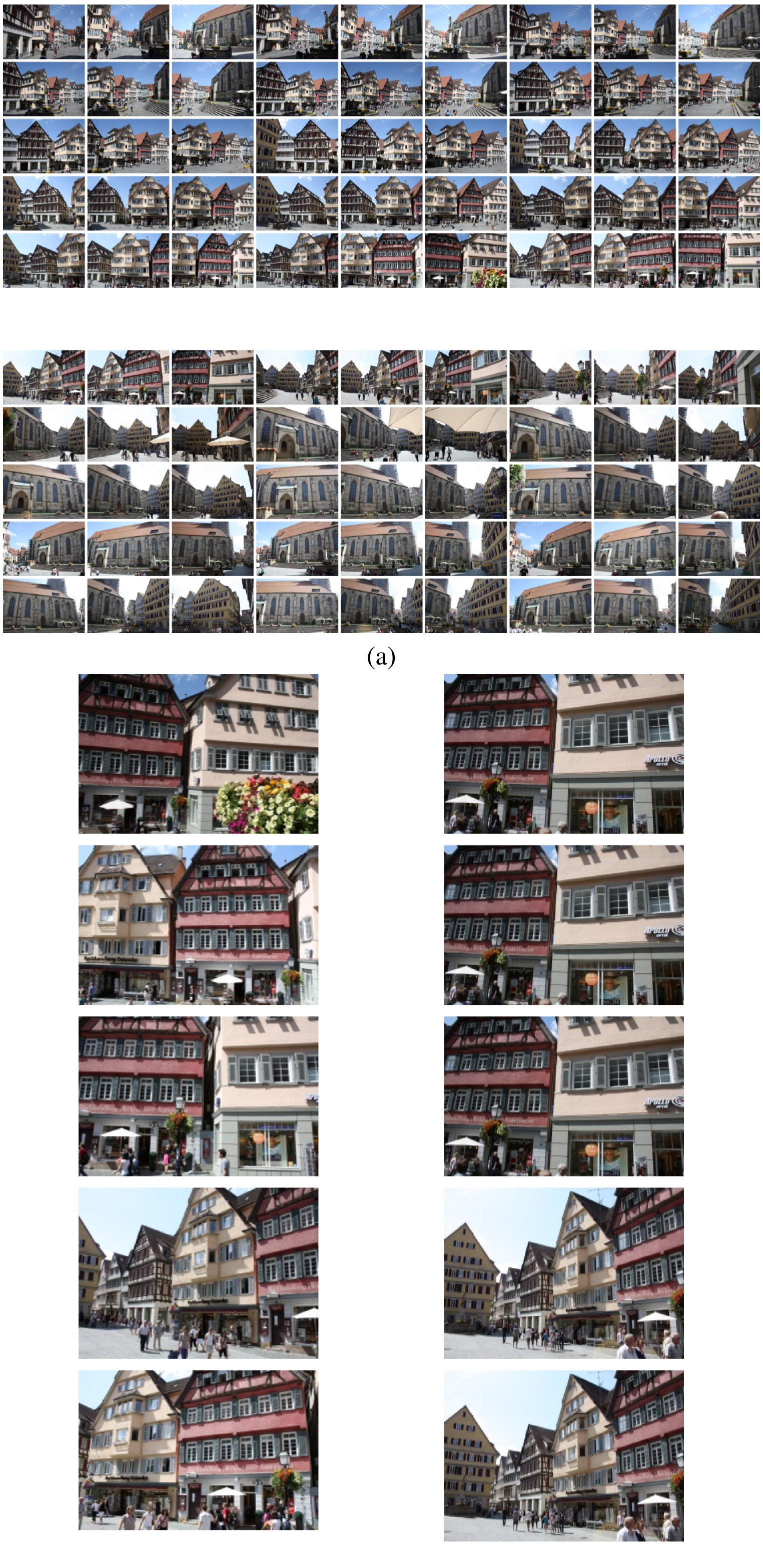
Figure 4.3: Image matching results for Tue1 dataset. Two sequences are shown on (a), and example matching pairs on (b).



(a)

(b)

Figure 4.4: Image matching results for Wro3 dataset. Two sequences are shown on (a), and example matching pairs on (b).



(a)

(b)

Figure 4.5: Image matching results for Tue2 dataset consisting of two 45 image sequences. Two sequences are shown on (a), and example matching pairs on (b).

4.5 Conclusions and Future Work

We have presented a novel approach for video sequence matching using compact descriptors. The speed of our algorithm in matching two video sequences can be compared to the speed of the standard SIFT feature algorithm for two images. We have successfully used the described descriptors in the context of large scale reconstruction.

In the future work, we would like to address the following issues:

- In addition to the selection of individual matching images, we will work on extending the algorithm to output individual features that match in the selected images. This would require checking individual features contained in corresponding k-means clusters.
- We will work on an improved and still fast probability density function and probability integration instead of the simple voting scheme. However the estimate in Equation 4.8 provides stable results, and does practically not contribute to the total matching time.

Chapter 5

Large Scale Reconstruction

5.1 Overview

In this chapter, we describe our large scale reconstruction approach. This work has been published in [PS12].

The input for the algorithm is a set of sub-models, created from smaller sets of calibrated images, together with compact descriptors described in Chapter 4. Additionally, the GPS information can be used in order to speed up the computation. As output, the algorithm computes a set of transformation matrices, that are used to bring sub-models to the common coordinate system. All sub-models transformed by their corresponding matrices form a single large model.

The interesting property of the proposed approach is the possibility of incremental construction of the final composite model. It is not necessary to collect all the data at the beginning. The data collection, camera calibration, and computation of 3D sub-models can be performed by many users in an independent way. Furthermore, adding new sub-models can improve overall reconstruction quality by providing more stable paths in the similarity graph.

Since the quality of 3D sub-models may vary depending on the quality and quantity of input images, photographed scenes, and the algorithm used in small scale reconstruction, the approach is constructed in a way, that operations in 3D space are postponed to the latest stages of the algorithm. The overview of the method is as follows:

1. Computation of compact descriptors for sub-models. The descriptors are computed from input image sequences of sub-models. They are described and evaluated in Chapter 4.

2. Camera calibration for image sequences. This step produces a set of camera parameters, and a sparse 3D point cloud, representing the structure of the sub-reconstruction. This is done with a standard open source software - Bundler - that is capable of calibration of relatively small image sequences.

3. Building a similarity graph for sub-models. The nodes of the similarity graph are image sub-sequences. Edges of the graph are connecting nodes, that have images of common scene geometry. Edges are constructed by executing a compact descriptor comparison function between pairs of sub-sequences with overlapping GPS bounding boxes.

4. Computation of transformation matrices between sparse point clouds of sub-models that have a connection in the similarity graph. We use a RANSAC approach, and try to select a subset of matching 3D points that minimizes the overall sum of distances of all matching points.
5. Global-error driven minimum spanning tree construction (GED-MST). The GED-MST is used to compute transformations of sub-models to the global coordinate system. The tree construction algorithm is aware of a global fitting error, and tries to compute the best initial solution, that is improved during the course of the optimization in the next stage.
6. Large scale global bundle adjustment. Local camera calibration (within sub-sequences) has already been performed, but as more sequences are available, some adjustments of camera parameters and sparse point clouds may be required in order to obtain better photo-consistency in a final model. We show, how this can be done on a large scale, in the global coordinate system, by optimization of transformation matrices.
7. Reconstruction of sub-models in order to obtain dense point clouds. In the previous steps, we have computed and optimized transformations to the global coordinate system. When dense point clouds are available, then they can be transformed with these matrices, as the dense reconstruction is matching the sparse point cloud geometry.

In the following subsection, we present related work in the area of large scale reconstruction, and we describe the steps of our large scale reconstruction algorithm in detail.

5.2 Related Work

The pre-requisite for large scale reconstruction of scenes from images, is camera calibration, that for large amounts of data, has to be solved efficiently. In this section, we present the state-of-the-art methods of camera calibration and multi-view stereo reconstruction of large scenes.

5.2.1 Photo Tourism

One of the first large scale approaches to camera calibration is described in [SSS06] and [SSS08], where the authors present the system called *Photo Tourism*, that allows to navigate through large photo collections in 3D space. The system also renders a sparse point cloud representing the scene geometry, and the user can explore the environment by clicking the image frames of nearby views. Smooth transitions between photographs are achieved with morphing techniques. The browsing capabilities of the system are rather a byproduct of the core functionality - camera calibration for large image sets.

The overview of the method presented in this sub-section is as follows. First image features are extracted. The features are then matched between image pairs. Then they run a robust, iterative Structure-from-Motion (SfM) procedure to compute the camera parameters. As the SfM can compute only relative positions for each camera, the absolute coordinates (latitude and longitude) are determined using an interactive registration procedure with an overhead map.

First from each photograph a set of SIFT features is extracted. For each pair of images I,J, a feature matching algorithm is executed. The matching is done using the approximate nearest neighbor library (ANN), where a kd-tree is built for the image J. Each feature descriptor from image I is queried for the nearest and the second nearest neighbor using the kd-tree constructed for features of J. The two nearest neighbors have distances d_1 and d_2. The authors accept the match, if $\frac{d1}{d2} < 0.6$.

When a sufficient number of matches is found between image pairs I and J, then the fundamental matrix is estimated. The authors use a RANSAC based algorithm, where in each iteration they compute a candidate fundamental matrix using the 8-point algorithm [HZ04]. The fundamental matrix is then refined using the Levenberg-Marquardt algorithm. Matches that are outliers with respect to the fundamental matrix, are removed. If the number of remaining matches is less than twenty, the image pair (I, J) is considered as non-matching. Additionally, matching features are connected into *tracks* across multiple images. If a track contains a keypoint in the same image, it is discarded. The next step is to create an *image connectivity graph*, where images are treated as nodes, and an edge is connecting matching images.

The camera calibration procedure operates on the image connectivity graph. First, just a single pair of images is used to compute camera parameters with the five point algorithm. Tracks are triangulated in order to obtain 3D points from 2D keypoints. A bundle adjustment is done to improve the 3D points and the camera parameters. In the next step, a new camera is added to the optimization. The model is kept fixed, and only the new camera with its points is optimized. After this step a global bundle adjustment is performed. The procedure is repeated by adding new cameras, one by one. Several speed improvements are described for this procedure, for details, the reader is referred to the original publication [SSS08].

The downside of this algorithm is the computational complexity of the image relation graph construction. If we consider matching of two images as a basic operation, the algorithm is running in $O(n^2)$ time for n images. Fortunately, the patience at the preprocessing step is rewarded with a great browsing experience.

This system works for sets of unorganized images, however in case of video sequences, it is possible to build the image relation graph in much better time, as it is not necessary to consider all possible pairs.

5.2.2 Skeletal Graphs

In the paper [SSS], an improvement over the approach described in [SSS06] and [SSS08] is presented. As the global optimization of all camera parameters together with a large number of points is time consuming, in the proposed approach, only a meaningful subset of all images is reconstructed and optimized. This subset is called a *skeletal graph*, and it is selected carefully to capture the whole scene, and provide stable reconstruction. Once, the cameras and 3D points for the skeletal set are computed, the rest of the images is added to the model. In the rest of this sub-section, we will describe the procedure in more detail.

The quality of reconstruction can be described by *completness* and *accuracy*. If only the two properties are of consideration, then all the images in the data set should be used. However in case of very large image collections, the *efficiency* is an important factor. At sufficiently large data sets, the computational complexity of reconstruction algorithms may cause the whole reconstruction operation to be impractical. In Internet photo collections, there are many redundant images that do not contribute additional meaningful information to the reconstruction, but also there can exist some rare images, that are important. The identification of a small set of important images can help in much quicker reconstruction of the scene. The skeletal set can be used for a reconstruction that is an approximation of the full solution. This approximation can be used as a starting point for full bundle adjustment in order to improve the quality.

The approach presented in this paper starts with a full image relation graph G_I, where each pair of matching images is connected, and removes edges in order to reduce the number of parameters to optimize in the bundle adjustment step. There is a trade-off between speed and quality of reconstruction. The more edges are removed, the faster the algorithm becomes, but the quality can be degraded. This trade-off is expressed by a parameter t, called the *stretch factor*. The skeletal graph problem can be solved by finding the subgraph G_S of G_I with the maximum number of leaves, with the constraint, that the shortest distance between a pair of cameras (P, Q) in G_S is at most t times longer that the shortest distance between the same images in G_I. The subgraph with this property is called *t-spanner*, and the task is to find a t-spanner with a maximum number of leaves. The details of this work are out of scope of this thesis, and they can be found in the original paper [SSS].

The speed-ups achieved by this approach, are from two to around fifty times faster (depending on the data set) than the original full bundle adjustment. The assumption in this work is, that the initial relation image graph has been already computed, what as mentioned before can be time consuming.

5.2.3 Parallel Approach to Camera Calibration

One of the first approaches for the full reconstruction pipeline - from image matching to camera calibration and bundle adjustment - is presented in [AFS$^+$10]. The work is based on previous attempts [SSS06], [SSS08], [SSS] of the large scale camera calibration. In this section, we describe how the authors parallelize the state-of-the-art algorithms in order to perform the reconstruction in much shorter time. In this way, it was possible to reconstruct a model consisting of 150K images in less than 24 hours using a cluster with 500 computing cores. The major parts of the processing pipeline are: (1) pre-processing, (2) matching, and (3) geometric estimation. In the following paragraphs, we briefly describe this approach.

The images are located on a central storage. All nodes request and process the images. The processing includes extracting EXIF tags, recording the focal length, downsampling to 2MP, converting to gray-scale, and finally extracting the SIFT features. This task is automatically load-balanced. The full set of processed images is partitioned into disjoint subsets, one for each node. Each node operates on its own subset in the later processing stages.

In the image matching stage, the previous approach from [SSS06] and [SSS08] is no longer followed. This is motivated by the fact, that the computational complexity of the exhaustive matching algorithm is not practical for a big collection of images, and additionally, the majority of pairs do not match. The chosen solution is based on recent work on object retrieval [SZ03], [NS06], [CPS$^+$07], [SBS07]. For a query image, first a set of possible matching images is determined with a fast algorithm based on vocabulary trees, and then detailed feature matching is performed in order to discard any false positives. Each node builds a k-means tree in order to quantize all the features of images associated with this node. This quantizations are aggregated over all features contained in an image to compute the term frequency (TF) for the image, and the document frequency (DF) vector for a set of images. The document frequencies are broadcasted, and each node builds a single vector of DFs from all computing nodes. Each node normalizes its TF vectors in order to obtain the TFIDF (term frequency inverse document frequency) matrix. Per-node TFIDF matrices are broadcasted in the cluster. In this way, each node can calculate the inner product between its own TFIDF vectors and a global TFIDF vectors. In can be seen as a distributed product of the matrix of TFIDF vectors by itself, where each node computes the block of rows corresponding to the set of images associated with this node. The query set of images is the same as the database, and it is not available in the stage, where features are being encoded. Thus, another matrix multiplication stage is required in order to find the best matching set of images. Vocabulary trees are described in more detail in the related work section of Chapter 4.

As the vocabulary tree methods return sometimes false positives, and do not check the geometric configuration of the features, the next step is to verify the match proposals precisely. The goal is to perform parallel image matching while minimizing the number of network transfers. The master node has a list of images assigned to each node. When a node requests for a job package, the master node runs through the image list, and adds image pairs to the job package, if the pairs do not require network transfers. When the job package is full, it is transferred to the node requesting a task. A node performs two step verification. First, standard feature matching is done, and when it succeeds, the essential or fundamental matrix is computed. When this step is completed successfully, then images get connected in the image relation graph. At the end of the stage, the graph may not be dense enough to provide a good reconstruction, the next step is to merge the connected components. Additionally query expansion is performed to find even more potential matching candidates.

The last step of the image matching procedure is to combine all the information in order to generate consistent feature tracks across images. Each node generates tracks from the locally available data. Local feature graphs are then broadcasted to all the nodes, and merged using shared features. At this stage, all required data for camera calibration and bundle adjustment is present.

The skeletal graph algorithm [SSS] is then performed in order to reduce the graph, and improve the bundle adjustment speed. One of the best publicly available software packages for sparse bundle adjustment is [LA09]. The performance is achieved with so called Schur complement trick to reduce the size of the system of linear equations [TMHF00] that has to be solved in each optimization step. The work described in [AFS$^+$10] further improves the performance of the bundle adjustment algorithm.

The parallel approach described in this section can perform sparse reconstruction and camera calibration on the cluster with 500 processors in 21 hours for the Rome dataset (150 000 images) and in 65 hours for the Venice dataset (250 000 images).

5.2.4 Efficient Image Relation Graph Construction

Further improvements over the work [AFS$^+$10] have been done in the paper 'Building Rome on a Cloudless Day' ([FFGG$^+$10]). The approach presented in this work aims at 3D reconstruction from unorganized photo collections with about 3 millions images, within one day, on a single PC. The computer used in the reconstruction is however quite powerful - dual Intel quad core Xeon 3.33 GHz, four NVidia 295GTX graphics cards, 48 GB RAM, and 1 TB SSD hard disk. This approach is 100 times faster on 10 times bigger datasets, when compared to the method described in [AFS$^+$10]. In this section, we describe this work in more details.

Appearance-based clustering with small codes

In this step, for each image, a GIST feature [OT01] is extracted, and concatenated with extremely downsampled versions of this image. Descriptors obtained in this way, are casted to compact binary strings using a locality sensitive scheme ([RL09], [TFW08]). Then the clustering based on the Hamming distance of binary features is performed. The clustering is done on the GPU using the k-medoid algorithm [KR90]. In the k-medoid algorithm an existing feature is used as the cluster center. In the standard k-means, usually the cluster center is represented as the center of gravity of the cluster. With this approach, the initial relations between images can be computed in a short time, however this is not yet sufficient for the 3D reconstruction.

Geometric cluster verification

In this step, in each cluster, a 'core' set of images with a mutually consistent epipolar geometry is computed. This is done with a fast RANSAC method [RFP08]. All other cluster images are checked for matching to the 'core' set, and removed, in case of an inconsistency. From each cluster, a single representative image is selected and used as an iconic view. As sometimes the GPS coordinates are recorded in the images, it was possible to geo-locate more that 50% of the clusters for a tested data set.

Local iconic scene graph reconstruction

The vocabulary tree algorithm is used to find similar iconic views [NS06]. The GPS data is used to exclude distant images from consideration. In order to prune false positives from the image similarity graph, a geometric verification is performed. Geometric verification is based on matching SIFT features on the GPU. Then the fundamental matrix is computed for each pair. The fundamental matrix is used for further verification of matching images. This step leads to a set of local iconic graphs, which are extended with additional views from the iconic clusters. In each local iconic graph, a sparse bundle adjustment is performed. After this step the image relation graph together with camera calibration and sparse point cloud is obtained.

Computation of dense model

Once the image similarity graph is created, the cameras are calibrated, a dense 3D reconstruction is performed. The GPU implementation of 2.5D surface reconstruction algorithm is used for this purpose [GFP10].

5.2.5 Other Approaches for Camera Calibration

In the paper [SKZ99], a hierarchical bundle adjustment for video sequences is proposed. First, a long sequence is divided into shorter segments, where feature points can be tracked reliably. The 3D structure is reconstructed for each segment. Then from all segments, a small number of 'virtual key frames' is extracted, and the global bundle adjustment is done on those frames. This method provides a significant speed-up over standard bundle adjustment methods.

In the paper [MLD$^+$06], the authors claim, that matching a new frame to a number of previous frames and performing a bundle adjustment over a 'window' of number of frames is sufficient to obtain results, that are comparable with the standard hierarchical bundle adjustment described in [SKZ99].

5.3 Similarity Graph for Sub-models

In order to proceed with the large scale algorithm description we need a formal definition of a sub-model:

Definition 1. *We define a sub-model as a set*

$$S = (I, C, P) \tag{5.1}$$

where $I = \{i_1, ..., i_k\}$ *is the set of* k *images,* $C = \{c_1, ..., c_k\}$ *is a set of* k *cameras,* $P = \{p_1, ..., p_n\}$ *is a sparse point cloud. Each sparse point cloud element* p_i *is defined as a set*

$$\{x, y, z, (i_1, f_1), ..., (i_u, f_u)\}$$

containing 3D coordinates x, y, z*, and pointers* (i_j, f_j) *to SIFT features, to which it can be projected to. Each pointer to a SIFT feature contains image and feature number.*

The large scale reconstruction algorithm relies on an overlap between sub-models. In order to enforce a good fitting between sub-models, we need to know their degrees of overlap. As the number of sub-models can be very large, even with a quick similarity evaluation between compact descriptors, the total number of pairs grows quadratically. For faster processing, we use GPS bounding boxes in order to consider sub-models located within certain proximity.

We construct a weighted graph $G = (V, E, w)$, where V is the set of sub-models defined by image sequences, E is the set of edges, and w is the set of edge weights. An edge belongs to the graph, only when there is an overlap in geometry captured by some images of two sub-models. The edge weights are the similarity values resulting from the compact descriptor comparison. This graph can be constructed incrementally, and is sparse by definition. For each node, we limit the number of neighbors to three. If a new node is inserted to the graph, some edges may be replaced with new ones, with a better similarity value. Limiting the number of edges has a great impact on the performance of the next steps of the algorithm (described in Section 5.4, 5.5 and 5.6), while keeping enough constraints to enforce a good global fitting. If the loop closure is considered as important, the edge selection may prefer to keep weak edges that are parts of cycles in the graph over stronger ones that have some degree of redundancy.

5.4 RANSAC-based Model Matching

In this section, we describe our algorithm for computing a transformation matrix between two sparse point clouds of sub-models.

There are many methods for computing a transformation matrix between two surface models. Some of the methods most widely used for similar problems are the Iterative Closest Point algorithms ([RL01]). The algorithms of this type can align two geometric models provided that an initial estimate of the transformation is known. The initial transformation can be specified manually or computed by a feature matching algorithm. A general idea is to define an error function, that depends on the parameters of the transformation, and use it in an optimization algorithm. In this case, the error function depends on a *distance of a point to a surface*, and this quantity is minimized for all points. This method is robust and used mainly in registration of laser scans. Unfortunately, in our case, we have no notion of a surface at this stage of the large scale reconstruction algorithm. Of course, we could use dense reconstructions of sub-models, however the ICP algorithm will not be as robust, as in the case of laser scans, because reconstructions may be imprecise and incomplete. The only data we use at this stage, are the sparse point clouds of the sub-models, where often it is not possible to define any surface at all. However, the points can be considered stable, because the camera reconstruction algorithm effectively filters outliers and optimizes point positions.

RANSAC (Random Sample Consensus) is an iterative method of parameter estimation of mathematical models, where a small sample of data is assumed to contain inliers only, and is used to compute a hypothesis for a model. The hypothesis is then verified with all the data points, returning a value (known as a score) that describes, how well the hypothesis explains the data. Iterating over small random samples of data helps to select the hypothesis with the best score, that explains the data in the most precise way. We have developed a RANSAC based algorithm, that can compute the transformation matrix between sub-models, and is robust with regard to false positive matches.

We need an algorithm that can estimate the transformation between sub-models and can handle some false positive matches. As we know sparse point clouds of two sub-models, and we can select matching images between sub-models with our compact descriptors, we can perform feature matching on the selected images. Since image features are linked to 3D points, we have correspondences between 3D points, however some of them may be false positive matches, as we rely on the SIFT matching algorithm.

When overlapping of two sub-models S_i and S_j has been confirmed by our compact descriptor matching routine, we can proceed with the computation of the transformation matrix using Algorithm 2. As input, the algorithm takes two sub-models and a structure

$$M_{ij} = \{(i_1, f_1, p_1), (i_2, f_2, p_2)\}_{k=1}^{n} \tag{5.2}$$

containing probable matches between the sub-model image features. Each element of M contains references to matching SIFT features (image i and feature number f) and 3D points p in the local coordinate systems of S_i and S_j associated with these features. The algorithm computes transformation matrix $\mathbf{T}_{i,j}$, used to transform the sparse point cloud geometry of S_i to the coordinate system of S_j.

In the algorithm, we draw random matches from the set of matches M_{ij}. As the speed of the algorithm depends on the number of iterations, we need to make sure, that we select random subsets of matching 3D points, that do not contain outliers. Then, we can avoid computing a score for a wrong transformation. A quick heuristic used to eliminate potential outliers is presented in the function 'RandomPoints' in Algorithm 2. The function selects point matches in groups of three. Three selected points form two matching triangles in two sub-models. If there is a wrong match, then the two triangles will not have a similar shape. We check this by computing ratios of edges for each triangle. If the ratios are not similar, then the three selected matching points are discarded, and the algorithm proceeds to select the next three random elements. As the three matching points are not sufficient to compute a robust transformation, we select a larger number of points, usually in order of 100-200.

When we have selected a set of 100-200 matching points, we need to find a transformation matrix, that transforms those points from the first sub-model to the corresponding points in the second model. In order to obtain the matrix $\mathbf{T}$, we need to solve the following equation set:

$$\begin{cases} \mathbf{T}(p_1) = q_1 \\ ... \\ \mathbf{T}(p_n) = q_n \end{cases} \tag{5.3}$$

where (p_i, q_i) is a pair of matching 3D points. As the 3D points come from camera calibration, the positions of the points can be slightly distorted. In practice, this turns down to the solution of the following minimization problem:

$$\underset{t \in \mathbb{R}^{12}}{argmin}(|\mathbf{At} - b|) \tag{5.4}$$

where $\mathbf{t} \in \mathbb{R}^{12}$ is a vector of transformation coefficients from matrix $\mathbf{T}$, $\mathbf{A}$ and $\mathbf{b}$ are matrix and vector created on basis of equation set 5.3. In the full matrix form, it can be written as:

$$\underset{t \in \mathbb{R}^{12}}{argmin\ abs} \left(\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{k+1} & p_{k+2} & p_{k+3} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{k+1} & p_{k+2} & p_{k+3} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{k+1} & p_{k+2} & p_{k+3} & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} t_{00} \\ t_{01} \\ t_{02} \\ t_{03} \\ t_{10} \\ t_{11} \\ t_{12} \\ t_{13} \\ t_{20} \\ t_{21} \\ t_{22} \\ t_{23} \end{pmatrix} - \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ q_{k+1} \\ q_{k+2} \\ q_{k+3} \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} \right) \tag{5.5}$$

The minimization of the above function can be performed with Singular Value Decomposition (SVD) method.

One important issue worth mentioning is the 'ComputeTransformationScore' function of Algorithm 2, where the set of matching 3D points is sorted, but only half of it is contributing to the final score. The idea behind this is that the set M_{ij} may contain wrong matches. This is caused by the SIFT matching algorithm, that does not take into account the geometry of matched features, just their local descriptors. False matches may cause the corresponding 3D points to be separated with long distances. We do not want these false matches to be included in the scoring function. Considering only the first half of the sorted set M_{ij} has turned out to be an effective method for removing outliers from further consideration, while considering a sufficient number of distances to provide stable scoring results. The alternative method used in our system is to compute the first quartile q_1 of distances, and include in the final score only the distances that are not greater than $3q_1$.

We apply Algorithm 2 to each pair of sub-models S_i and S_j, that are connected in the graph G, and associate the computed matrix $\mathbf{T}_{i,j}$ with the corresponding edge (i, j). We also compute $\mathbf{T}_{i,j}^{-1}$, and associate it with edge (j, i).

5.5 Large Scale Calibration of Extrinsic Parameters

In the previous section, we have shown how to compute transformation matrices between adjacent nodes in the similarity graph. This allows us to transform one sub-model to the local coordinate system of the other model. However we would like to find a global coordinate system that is common for all the sub-models. This will allow

Algorithm 2 Transformation Matrix Computation with RANSAC

```
function ComputeTransformation(Submodel $S_i$, Submodel $S_j$, Matches $M$)
    $s_{best} \leftarrow$ big_number
    $T_{best} \leftarrow \emptyset$
    for $i = 1 \rightarrow$ max_iters do
        index_set $\leftarrow$ RandomPoints($M$,$n$);
        $T \leftarrow$ ComputeTransformationMatrix($M$,index_set)
        $s \leftarrow$ ComputeTransformationScore($M$,index_set);
        if $s < s_{best}$ then
            $\mathbf{T}_{best} \leftarrow \mathbf{T}$
        end if
    end for
    return $\mathbf{T}_{best}$
end function
function RandomPoints(Matches $M$, count)
    index_set $\leftarrow \emptyset$
    while |index_set| $<$ count do
        $a \leftarrow$ random_index $mod$ $|M|$
        $b \leftarrow$ random_index $mod$ $|M|$
        $c \leftarrow$ random_index $mod$ $|M|$
        $d_{11} \leftarrow \|M[a].p_1 - M[b].p_1\|$        $d_{12} \leftarrow \|M[b].p_1 - M[c].p_1\|$
        $d_{13} \leftarrow \|M[c].p_1 - M[a].p_1\|$
        $d_{21} \leftarrow \|M[a].p_2 - M[b].p_2\|$        $d_{22} \leftarrow \|M[b].p_2 - M[c].p_2\|$
        $d_{22} \leftarrow \|M[c].p_2 - M[a].p_2\|$
        $r_{11} \leftarrow d_{11}/d_{12}$        $r_{12} \leftarrow d_{12}/d_{13}$        $r_{13} \leftarrow d_{13}/d_{11}$
        $r_{21} \leftarrow d_{21}/d_{22}$        $r_{22} \leftarrow d_{22}/d_{23}$        $r_{23} \leftarrow d_{23}/d_{21}$
        if $r_{11} \approx r_{21} \wedge r_{12} \approx r_{22} \wedge r_{13} \approx r_{23}$ then
            index_set $\leftarrow$ index_set $\cup\{a, b, c\}$
        end if
    end while
    return index_set;
end function
function ComputeTransformationMatrix(Matches $M$, index_set)
    $n \leftarrow$ |index_set|
    Solve the following set of equations to compute transformation $\mathbf{T}$:
    $\begin{cases} \mathbf{T}(M[\text{index\_set}[1]].p_1) = M[\text{index\_set}[1]].p_2 \\ \ldots \\ \mathbf{T}(M[\text{index\_set}[n]].p_1) = M[\text{index\_set}[n]].p_2 \end{cases}$
    return $T$
end function
function ComputeTransformationScore(Matches $M$, Matrix $\mathbf{T}$, index_set)
    for $i = 1 \rightarrow$ |index_set| do
        $p_1 \leftarrow M$[index_set[$i$].$p_1$]
        $p_2 \leftarrow \mathbf{T}(M$[index_set[$i$].$p_2$])
        $d \leftarrow \|p_1 - p_2\|$
        distances[$i$] $\leftarrow d$
    end for
    sort(distances)
    score $\leftarrow 0$
    for $i = 1 \rightarrow$ |index_set|$/2$ do
        score $=$ score $+$ distances[$i$]
    end for
    return score
end function
```

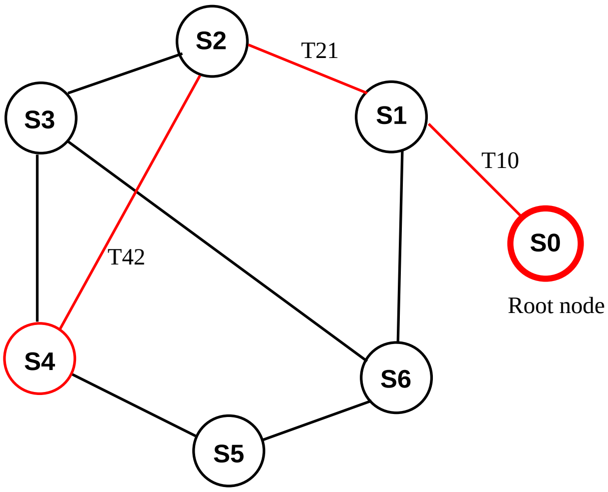


Figure 5.1: The Similarity Graph. Shortest path from the node S_4 to the root node S_0 is drawn in red color.

us not only to merge sub-models to a single, large model, but also to recover global extrinsic camera parameters - positions and rotations in the global coordinate system.

In this section we describe a number of approaches, that can be used for computing initial transformation matrices for sub-models in order to assemble a larger model in the global coordinate system. Most of the methods presented here rely on graph algorithms.

In this chapter, we assume to work on the largest connected component $G = (V, E)$ of the similarity graph.

5.5.1 Dijkstra's Algorithm

The most simple solution for this problem is to select one node of the graph, and use the coordinate system of the sub-model associated with this node as the global coordinate system. In order to compute transformation matrices for other sub-models, it is required to compute paths to the selected node, and multiply transformation matrices along the paths.

The simple approach is based on the classical Dijkstra's algorithm described in [Wik12b]. The purpose of the algorithm is to find a shortest path from each node to the selected node. The path length is defined as sum of the edge weights on the path. As the edge weights, we use the output of the error function 5.10. The root node is chosen randomly.

The computation of the global coordinate system transformation matrices for the example similarity graph is shown in Figure 5.1. The sub-model S_0 is selected as the root node. The global coordinate system is the coordinate system of the root node. In order to compute the global transformation matrix for the node S_4, the shortest path to the root node must be found (drawn in red color). We compute $\mathbf{T}_4$, the global coordinate system transformation matrix for sub-model S_4, by multiplying matrices on the shortest path to the root node:

$$\mathbf{T}_4 = (\mathbf{T}_{42} * \mathbf{T}_{21}) * \mathbf{T}_{10} \tag{5.6}$$

We repeat this process for every node in the similarity graph. The set of global coordinate transformation matrices is the initial solution for building a large composite model from sub-reconstructions.

This simple approach however very often will not give the optimal solution. It is possible that an edge containing a transformation matrix, that could not be precisely estimated, will occur in many paths. During matrix multiplication, this matrix will influence the overall error, causing imprecise fitting of sub-models. The situation, that is common to any choice of the global coordinate system and paths in the graph, is an error accumulation along the path, during matrix multiplication. Unfortunately, in the case of small overlap of sub-models or an insufficient number of input photographs, the transformation matrix cannot be estimated with high precision.

The computational complexity of Dijkstra's algorithm is $O(|E|+|V|\log|V|)$, what is an advanage, since it can be used in larger scenarios. The theoretical result of the implementation of Dijkstra's algorithm running in $O(|E|+\frac{|V|\log|V|}{\log\log|V|})$ is presented in [FW94], however due to the large constant, the algorithm is not practical.

5.5.2 Minimum Spanning Tree

Our experiments have shown, that the final fitting error is proportional to the length of all paths to the root node. Dijkstra's algorithm does not guarantee that the tree consisting of shortest paths contains the minimum number of edges.

The *spanning tree* (ST) of a connected, undirected graph, is a subgraph that is a tree and connects all the vertices together. The *minimum spanning tree* (MST) is the tree with the smallest sum of edge weights. For computing transformation matrices, such a tree is desired, as the number of matrix multiplications is minimized, what helps to reduce the error accumulation along the paths. The fastest MST algorithm from the theoretical point of view is presented in [FW94]. It can run in $O(|E|)$, however it is impractical due to a large constant hidden in the O notation. There are many algorithms that can be used to compute the MST. The survey [BH01] presents an overview of these methods. Some of the methods reviewed in their work can build MSTs in times $O(|E|\log|V|)$, $O(|E|+|V|\log|V|)$, and $O(|E|\log\log|V|)$.

The MST algorithms can minimize the total number of matrix multiplications along the paths, however it may often happen, that two sub-models connected in the similarity graph, but not in the spanning tree, may not fit well. This happens, because there is no notion of global error, and only the weights in the tree are summed up. An approach to improve the initial model creation by introducing the global fitting error of the sub-models is described in the next sub-section.

5.5.3 Global-error Driven Minimum Spanning Tree Construction

The graph theory does not solve our problem completely, as the graph algorithms assume constant weights for edges, while the measure of fitting for a single edge in the global coordinate system may depend on the order of matrix multiplications for sub-reconstructions associated with the similarity graph nodes. This order depends on the chosen paths to the root node. In other words, the fitting error between two sub-models can be computed only if two corresponding paths are known. In this section we propose a simple algorithm that is aware of a global-error fitting function.

The error in the matrix estimation cannot be avoided, however it is still possible to minimize it by selecting a good starting node, referred later as the *root node*. A good starting node, and a set of shortest paths from the starting node to all other nodes will help to minimize the global fitting error. Global fitting error functions used in this algorithm are described in more detail in Section 5.6.

We propose the following algorithm: We execute Dijkstra's algorithm for each node selected as a root. For each node, the global error function is evaluated. The starting node with a spanning tree that yields the smallest error function response is selected. The disadvantage of this algorithm is the running time. In practice, the complexity is $O(|V|(|E| + |V| \log |V|))$, what under a reasonable assumption that $k|V| \approx |E|$ gives the complexity of $O(|V|^2 \log |V|)$. In further sections, we call this algorithm the Global-error Driven Minimum Spanning Tree construction (GED-MST).

The visual comparison of the the simple Dijkstra (SD) shortest paths algorithm, with the Global-error Driven Minimum Spanning Tree construction algorithm is presented in Figure 5.2. The error function value for the SD algorithm was 1.744675 (max. 2.772635), while for the GED-MST algorithm 0.675194.

5.5.4 Improving the Computational Complexity

The assumption on the sparsity of the similarity graph is supported by the spatial distribution of sub-models in the case of a city reconstruction. Buildings overlap only with neighboring buildings most of the time. For the reconstruction we may assume a constant number of neighbors in the graph.

After making the assumption on the sparsity of the similarity graph, We come up with the computational complexity for the GED-MST construction of $O(|V|^2 \log |V|)$, what for simplicity may be written as $O(n^2 \log n)$.

In this subsection we show how to improve this complexity, and additionally how to parallelize the GED-MST construction.

We apply a standard divide and conquer approach. We partition the similarity graph into $\sqrt{n}$ subgraphs, and recursively execute (maybe in parallel) the GED-MST construction algorithm. We merge models, and each subgraph becomes a node in a new similarity graph. We execute the GED-MST algorithm on the new similarity graph in order to merge a set of models, that were already merged in the previous step. The recursion stops at a subgraph of a fixed size. This divide and conquer approach does not give the same final result as the original algorithm from Section 5.5.3, as it computes sub-problems independently. However as we show in the next paragraph, it has a much better computational complexity.

The complexity function of our GED-MST can be expressed as:

$$F(n) = n^2 \log n \tag{5.7}$$

When using divide and conquer strategy by subdividing the problem to $\sqrt{n}$ subproblems, we can write the computational complexity reduction in this way:

$$\begin{aligned} O(F(n)) \rightarrow \sqrt{n}O(F(\sqrt{n})) + O(F(\sqrt{n})) = \\ O(n^{\frac{1}{2}})O(F(n^{\frac{1}{2}})) + O(F(n^{\frac{1}{2}})) = \\ O(n^{\frac{1}{2}}F(n^{\frac{1}{2}}) + F(n^{\frac{1}{2}})) = \\ O((n^{\frac{1}{2}} + 1)F(n^{\frac{1}{2}})) = \\ O((n^{\frac{1}{2}} + 1))O(F(n^{\frac{1}{2}})) = \\ O(n^{\frac{1}{2}})O(F(n^{\frac{1}{2}})) \end{aligned} \tag{5.8}$$

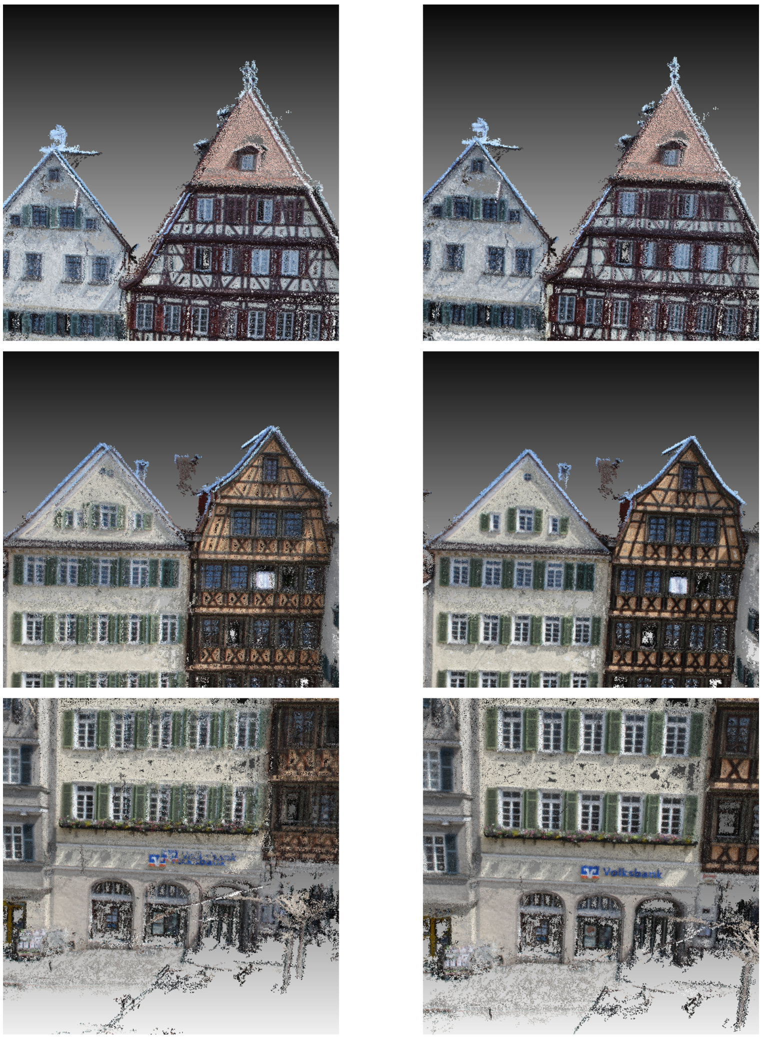


Figure 5.2: Visual comparison of the Simple Dijkstra's (left) and the GED-MST algorithm (right).

The meaning of the above equation is, that in case of the subdivided problem, we need to execute $\sqrt{n}$ times the algorithm of complexity $O(F(\sqrt{n}))$ for a sub-problem of size $\sqrt{n}$. Additionally, we need to apply a merging step for the computed sub-problems - the GED-MST algorithm for the composite graph of size $\sqrt{n}$.

If we consider only the single recursion step to compute sub-problems, Equation 5.8 gives us the complexity of $O(n^{\frac{3}{2}} \log \sqrt{n})$. We apply the recursion step multiple times:

$$
\begin{aligned}
O(n^2 \log n) = \\
O(F(n)) \rightarrow \\
O(n^{\frac{1}{2}})O(F(n^{\frac{1}{2}})) \rightarrow \\
O(n^{\frac{1}{2}})O(n^{\frac{1}{4}})O(F(n^{\frac{1}{4}})) \rightarrow \\
O(n^{\frac{1}{2}})O(n^{\frac{1}{4}})O(n^{\frac{1}{8}})O(F(n^{\frac{1}{8}})) \rightarrow \\
... \\
O(n^{\frac{1}{2}})O(n^{\frac{1}{4}})...O(n^{\frac{1}{2^k}})O(F(n^{\frac{1}{2^k}})) = \\
O(\prod_{i=1}^{k} n^{\frac{1}{2^i}})O(F(n^{\frac{1}{2^k}})) = \\
O(n^{\sum_{i=1}^{k} \frac{1}{2^i}})O(F(n^{\frac{1}{2^k}})) = \\
O(n^1)O(C) = \\
O(n)
\end{aligned}
\tag{5.9}
$$

The above equation shows, that finding initial transformation matrices required to bring sub-models to the common coordinate system, can be done in a linear time. As the recursion stops at a fixed subgraph size, the fitting error will be optimized, by trying different spanning trees within this subgraph. The subdivision scheme should break the similarity graph into subgraphs that are connected with a reliable transformation matrices. Additionally, the sub-problems can be computed in parallel.

Apart from the theoretical analysis described above, in practice, we need to perform the subdivision into sub-problems carefully. As the composite graphs are merged together, is is necessary to guarantee, that there exist stable links between subgraphs. In the case of city reconstruction, we have an additional information about GPS bounding boxes of photos that were used to reconstruct sub-models. One of the possible subdivision schemes is to overlay a 2D grid over the whole scene, and consider the grid cells as sub-problems, on which the GED-MST is executed. As a further optimization, neighboring grid cells with 'weak links' could be merged together, and treated as a single sub-problem. In this context, a 'weak link' between two cells occurs when there are no reliable transformation matrices between sub-models located in different cells.

5.6 Large Scale Bundle Adjustment

The initial large model can be computed from sub-models, using the GED-MST algorithm over the similarity graph, as described in Section 5.5. This solution however is approximate, what can be caused by insufficient overlap of sub-models, small number of overlapping features, or imprecise sparse 3D point clouds obtained from the camera calibration. Therefore, the solution should be treated as a starting point for further refinement.

In order to improve the global fitting of submodels, we need to define functions describing, how good the fitting is. These functions, in the scope of optimization methods, named *objective functions*, can measure the global fitting error, and can be defined in two ways.

1. **Global geometric error function.** This error function, when used in the optimization, takes care of minimizing Euclidean distances between surfaces or sparse 3D point clouds of sub-models. This type of error function is used in the initial creation of the large, composite 3D model, as it can be computed pair-wise between different sub-models, without any notion of the global coordinate system.
2. **Photoconsistency-based error function.** This is a standard error function used in multiview reconstruction problems. The pre-requisite for using this function in the context of improving the global fitting, is the selection of the global coordinate system. This function measures the *visual fitting*, as the error measure is done from the view perspective of the cameras.

In the next subsections, we define the two types of previously mentioned error functions, and give an argumentation, that both types are required in our large scale reconstruction algorithm.

5.6.1 Global Geometric Fitting Error Function

The pair-wise transformation matrices are computed directly on the sparse point clouds (as described in Section 5.4). An optimization procedure computes a matrix, that brings one model to the other, while minimizing the distances between points. This is done by solving a system of equations. As there are no camera parameters (radial distortion) used in this step, the system of equations does not have any non-linear terms. For the definition of the global fitting error function, it is desired to extend the local fitting error function to the whole model. This allows us to be consistent with the original RANSAC based approach. For the sub-models S_i, S_j, the set of matches M_{ij} defined by Formula 5.2, the formal definition of the error function is:

$$Err(\mathbf{T}_i, \mathbf{T}_j) = \frac{1}{|M_{ij}|} \sum_{m \in M_{ij}} ||\mathbf{T}_i(m.p_1) - \mathbf{T}_i(m.p_2)|| \tag{5.10}$$

The difference between this error function and the function 'ComputeTransformationScore' from Algorithm 2 is that the set of matches M_{ij} has now incorrect matches removed. This has been done by setting an acceptance threshold for distances between transformed points of S_i and points of S_j.

As we are interested in the error measure of the global fitting, we need to extend the local geometric error (Equation 5.10) to the whole similarity graph G:

$$Err(G) = \sum_{(i,j) \in E(G)} Err(\mathbf{T}_i, \mathbf{T}_j) \tag{5.11}$$

what is the sum of error functions defined by Equation 5.10 over all edges of the similarity graph G.

This error function is used only for computing a good initial fitting, however there are no contraindications to use it in the scope of matrix optimization in order to improve the global fitting. After the initial fitting solution is obtained with the spanning tree construction and the objective function defined in Equation 5.11, the set of global coordinate system transformation matrices is refined later with the photoconsistency-based fitting measure (described in detail in the next subsection).

5.6.2 Photoconsistency-based Error Function

In the bundle adjustment problem, it is required to refine initial camera parameters together with 3D points. The solution is based on optimization techniques. As we precisely know the projections of 3D points to 2D images - they are usually SIFT feature locations, computed with sub-pixel precision, we build an error function computing the reprojection error. The reprojection error is the average distance of projected points to the corresponding feature locations. The difference between the standard bundle adjustment and our approach is as follows:

- In standard bundle adjustment, the error function has an enormous number of dimensions - 3 for each point, and even up to 14 for each camera (3×4 projection matrix and two radial distortion coefficients.
- In our approach, as standard bundle adjustment has been already performed within sub-models, we optimize only 3×4 sub-model transformation matrices, that have influence on cameras and points contained within them.

The number of variables to optimize differs a lot. For example, in the Tuebingen Markt dataset, containing around 121 000 points and 200 cameras, we have about 365 400 variables in the standard bundle adjustment error function, while in our case, there are 108 variables (9 sub-models) to optimize. For optimization methods that compute partial derivatives at each iteration, the difference is significant. Of course, the standard bundle adjustment influences each point and camera individually, what can lead to a better quality, however, we rely on precise results of bundle adjustment performed within sub-models, that can be done in an independent way, so it can be easily parallelized.

In order to proceed with the formal definition of the global photoconsistency-based error function, first we need to define the local photoconsistency-based error function, that describes visual fitting between two overlapping sub-reconstructions S_i and S_j. For our large scale bundle adjustment, for each overlapping pair of sub-models, we need cameras that observe the same geometry. This information has already been computed in the first stage of sub-reconstruction matching and is contained in the structure M_{ij} (see eq. 5.2).

Now we can proceed with the formal definition of the photoconsistency error measure for two sub-models:

$$\begin{aligned} Err(\mathbf{T}_i,\mathbf{T}_j) = {} & \frac{1}{|M_{ij}|}\sum_{m\in M_{ij}} ||\Pi_{m.i_2}(\mathbf{T}_j^{-1}(\mathbf{T}_i(m.p_1))) - m.f_2.xy|| + \\ & \frac{1}{|M_{ij}|}\sum_{m\in M_{ij}} ||\Pi_{m.i_1}(\mathbf{T}_i^{-1}(\mathbf{T}_j(m.p_2))) - m.f_1.xy|| \end{aligned} \tag{5.12}$$

where Π_k is the camera projection matrix for an image k, $\mathbf{T}_i$ and $\mathbf{T}_j$ are the matrices, that transform sub-models S_i and S_j to the global coordinate system. The term $m.f_1.xy$ is a 2D vector of the feature position. This error function works as follows:

1. Start with a structure of matching 3D points and features (Equation 5.2) and the cumulative distance $D = 0$.
2. For each entry $m \in M_{ij}$ do the following:
    (a) Transform $m.p_1$ to the global coordinate system with the matrix $\mathbf{T}_i$ in order to obtain point $\mathbf{p}'$.
    (b) Transform $\mathbf{p}'$ to the local coordinate system of S_j by the matrix $\mathbf{T}_j^{-1}$ to get the point $\mathbf{p}''$.
    (c) Project the point $\mathbf{p}''$ to the image $m.i_2$ to get the 2D point $\mathbf{p}''_{i_2}$.
    (d) Compute the distance d of $\mathbf{p}''_{i_2}$ to the location of the corresponding feature $m.f_2$.
    (e) Add the distance d to the cumulative distance D.
3. Repeat step 2 in the opposite direction - by reprojecting points from S_j to S_i.
4. Compute the average distance from D, and return it as the error measure between sub-models S_i and S_j.

This error function is then used in the global scenario - we sum its outputs for each matching pair of sub-models in the similarity graph, exactly like in the Equation 5.11.

5.7 Similarity Graph Optimization

Once we have defined the error functions, we can describe the optimization process of the similarity graph. In order to improve the sub-model fitting, we optimize the global coordinate system transformation matrices. As previously mentioned, the two objective functions can be used to improve the geometric fitting or the visual fitting (from the perspective of the cameras). Our optimization framework can use both objective functions.

In the similarity graph, each model has a transformation matrix that brings it to the global coordinate system. We concatenate all the matrices, we want to optimize to form a single long parameter vector of the size 12 times the number of sub-models, and use this as a parameter to the error function defined by Equation 5.11 (or the photoconsistency based error function).

We have used two optimization schemes: Levenberg-Marquardt nonlinear least squares optimization and a simple gradient descent approach. For the first one, we use the open source library [Lou04], while the second scheme is implemented from scratch. In both cases, the important step is to compute the derivatives of the error function. In this case, they have to be computed with numerical methods. The partial derivatives can be computed in an optimized way. As the single change in the parameter vector influences only one sub-model, we do not have to recompute the reprojection error between all sub-models in the similarity graph. Changing a single parameter would require to recompute the local error function from Equation 5.12 only for the sub-models connected to the affected sub-model in the similarity graph.

As we optimize the transformation matrices of size 3×4 directly, also non-rigid deformations are possible. This is an advantage, since a sub-model build from insufficient number of images can be deformed. The 3×4 matrix allows for non-rigid deformation, so there are more degrees of freedom in obtaining a better global fitting.

	TownSquare	Courtyard	Statue
N	9	14	15
T_q	110.67	356.32	508.98
T_f	2898.92	1754.31	3830.28
T_r	409.43	832.98	259.18
K_p	288	72	114
T_p	472.4	31.7	92.92
E_p^s	75.67	114.57	62.03
E_p^e	46.43	113.09	52.15
α_p	38.6%	1.3%	16%
K_g	310	56	32
T_g	538.8	18.9	21.63
E_g^s	0.678	0.0025886	0.002687
E_g^e	0.614	0.0025875	0.002599
α_g	9.4%	0%	3.3%

Table 5.1: Performance summary.

5.8 Single Camera Optimization

After the global coordinate system matrices have been adjusted for sub-models, it is possible to further improve camera positions within the sub-models. During the initial camera calibration for sub-models, only the 3D features within sub-model were taken into account. However, after the construction of the similarity graph and transformation estimation for all sub-models, more data is available. The camera positions can be adjusted not only by using 3D features of the sub-model, from which the cameras were reconstructed, but also by using the 3D features of overlapping sub-models.

5.9 Results

In this section we present results of our large scale reconstruction approach. All the time measurements were done on a single core of Intel(R) Core(TM)2 Quad Q9300 2.5 GHz CPU, 4GB of RAM, and Windows 7 x64 OS (unless stated otherwise).

Figure 5.3 shows sub-models in local coordinate systems (top), that were merged together with our algorithm into a single composite model (bottom). The result shown is after matrix optimization. Each color shows a different sub-model. Other examples of our reconstruction algorithm can be seen in Figures 5.9, and 5.8. The performance of our algorithm is summarized in Table 5.1.

The following times are measured: quick matching time based on Compact Descriptors (T_q), full image matching for image pairs selected by Compact Descriptors (T_f), the time required for 5000 iterations of the RANSAC algorithm (T_r), optimization times for geometric and photoconsistency error functions (T_g and T_p). The number of iterations for geometric and photoconsistency optimization is given as K_g and K_p. Starting and ending error function values and improvement factor for geometric optimization are E_g^s, E_g^e, α_g. The same times are given for photoconsistency optimization (indexed with 'p'). The number of sub-models is denoted as N. All times are given in seconds.

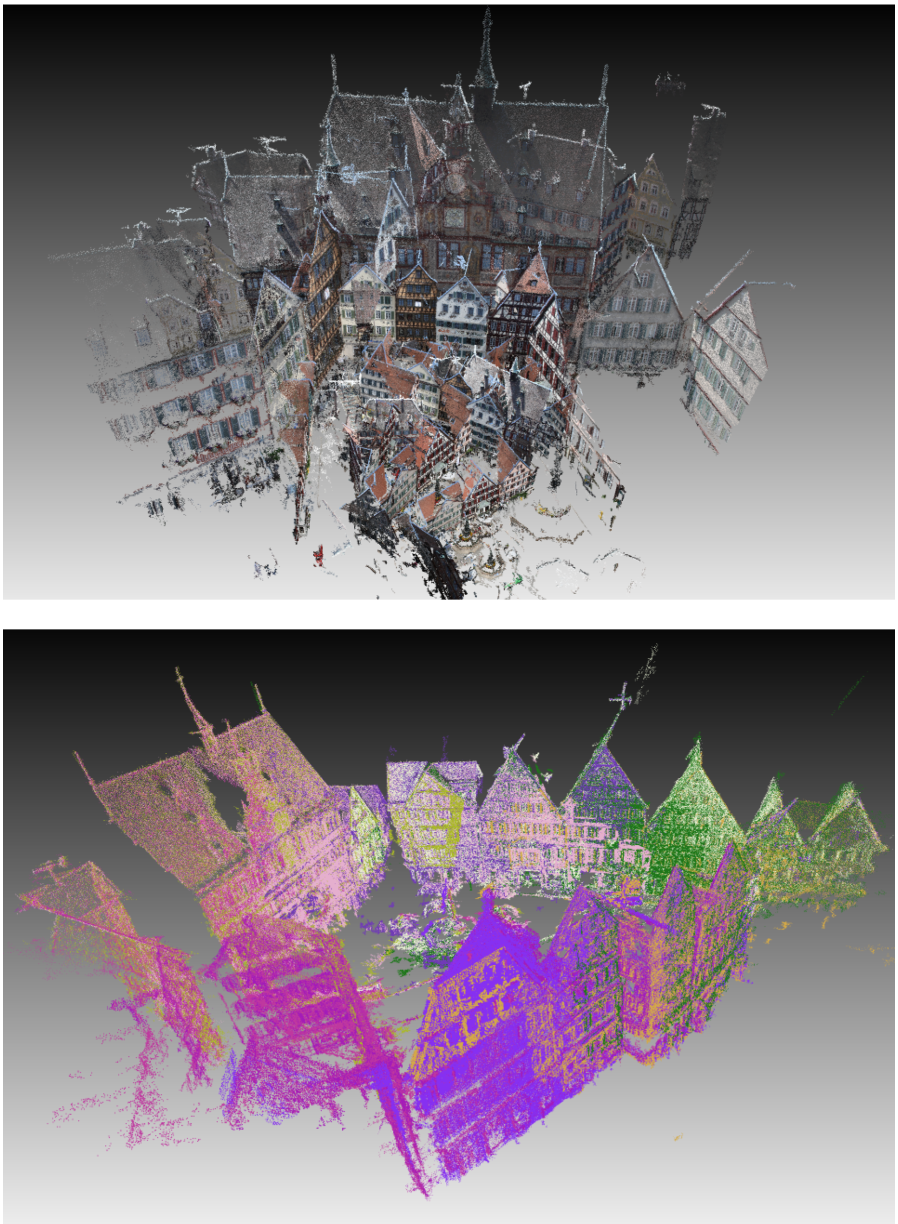


Figure 5.3: Sub-models in local coordinate systems (top) and the composite model (bottom) after optimization.

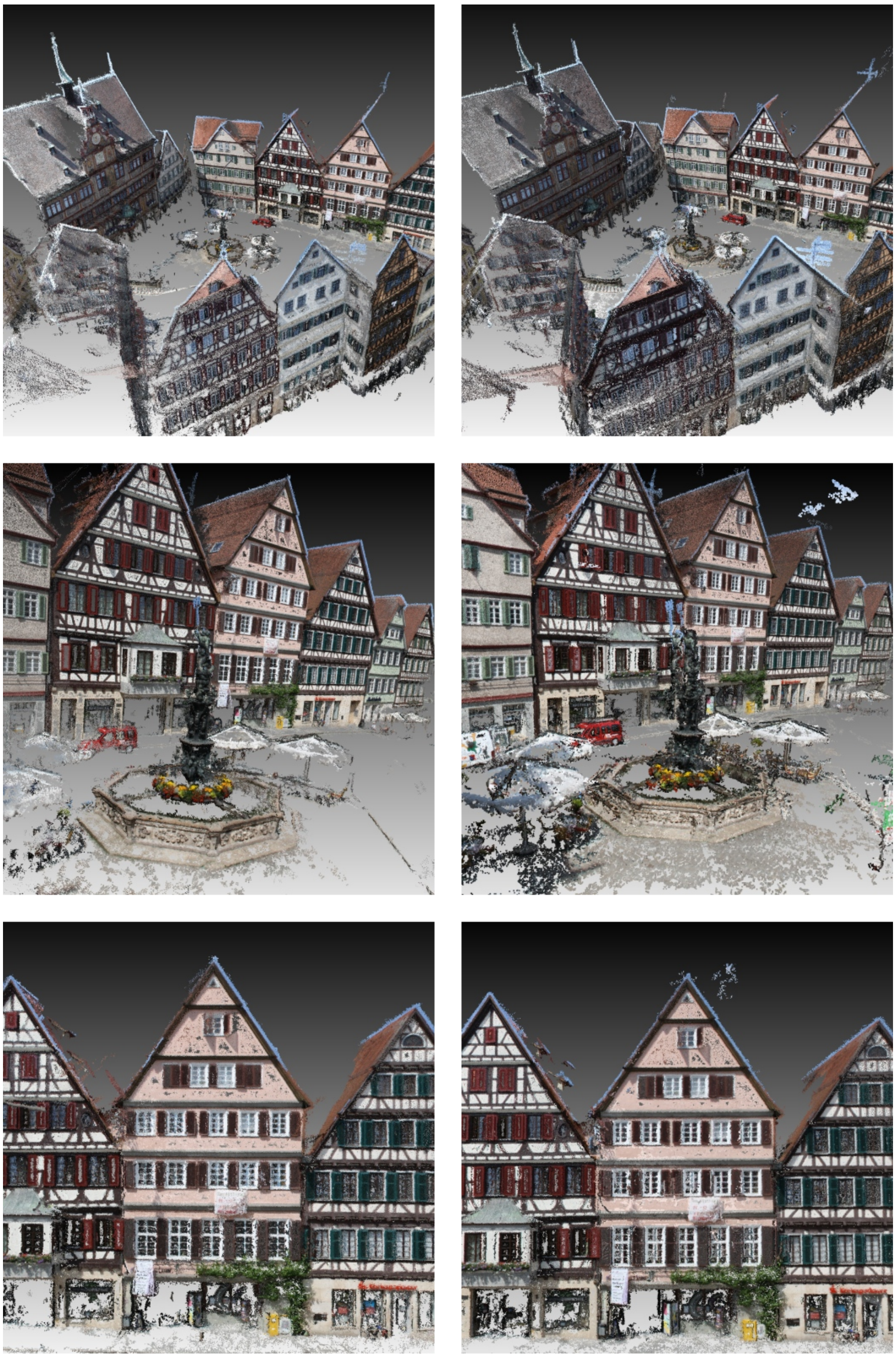


Figure 5.4: Comparison of full PMVS reconstruction (left column) and the result of merging 9 sub-models with our algorithm (right column).

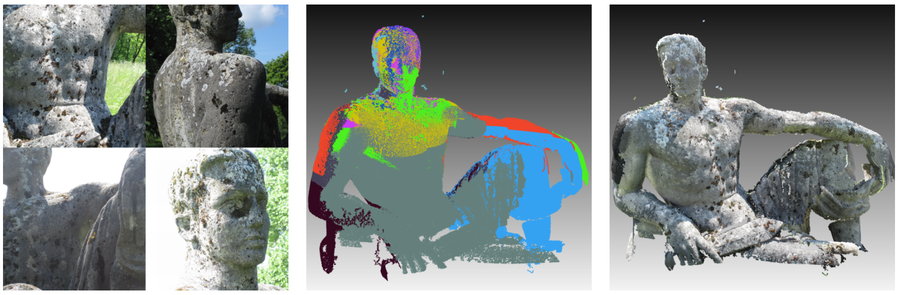


Figure 5.5: Example of a composite reconstruction. Sample input images (left), the final reconstruction (middle, right).

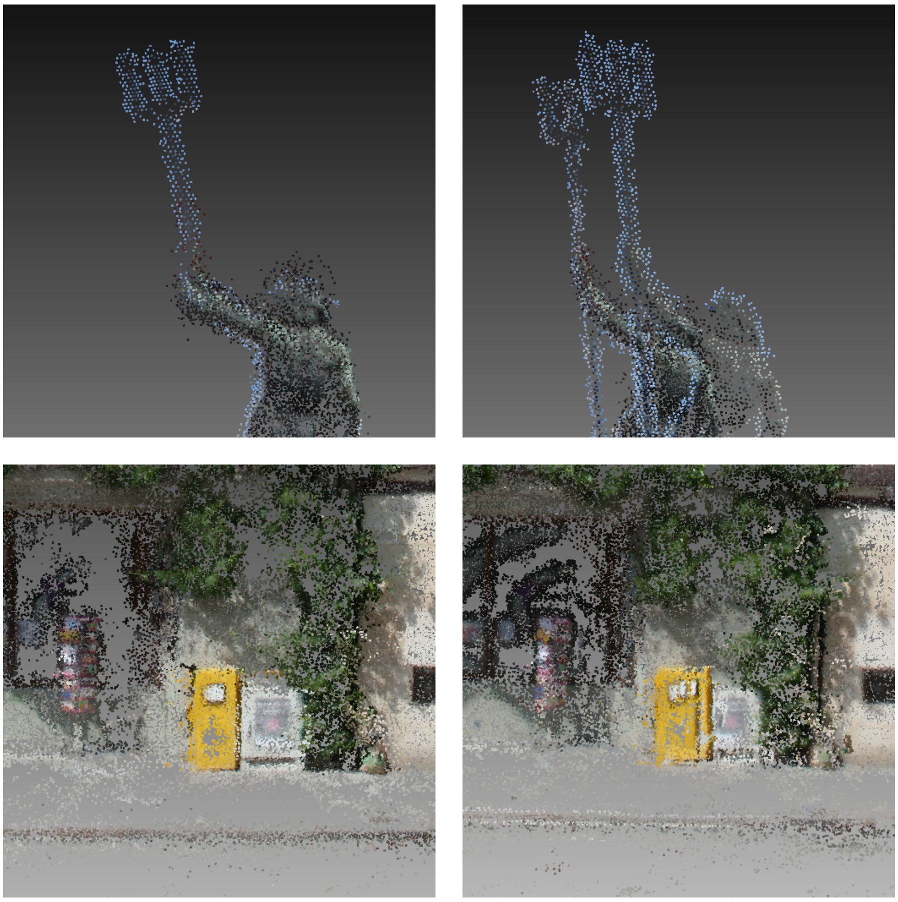
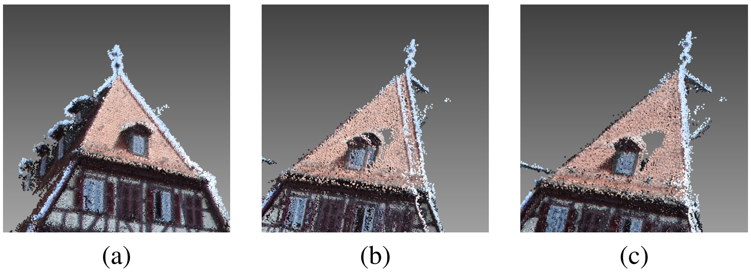


Figure 5.6: Artifacts ocurring due to imprecise reconstruction of sub-models. Full PMVS reconstruction (left column) and the result of merging (right column).



(a) (b) (c)

Figure 5.7: Optimization result: (a) photoconsistency optimization - view from the area of camera locations, (b) photoconsistency optimization - view from top region, (c) geometric optimization - view from the top region.

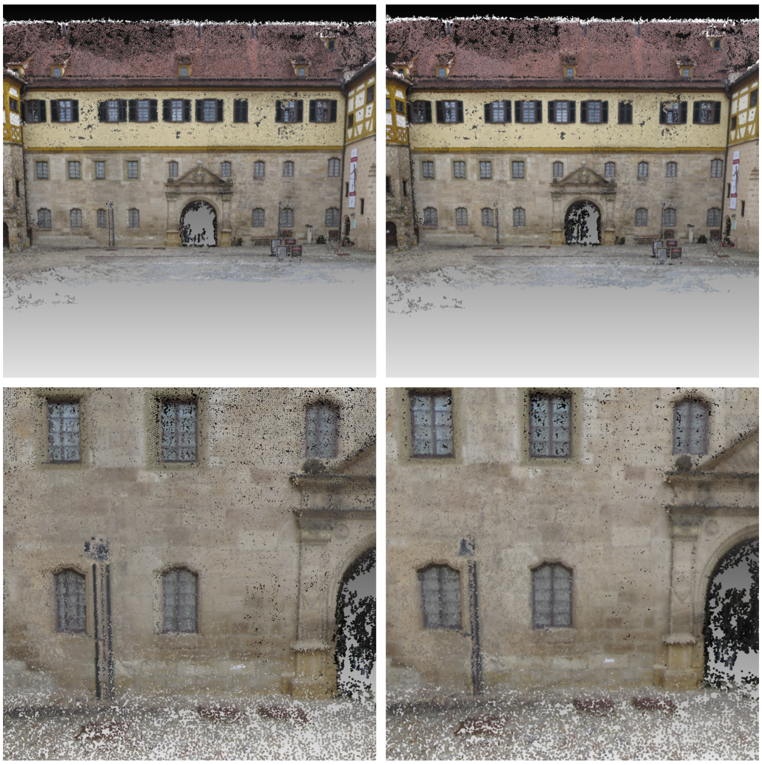


Figure 5.8: A view of the Tuebingen Castle reconstruction. Geometric fitting optimization (left column) and photoconsistency optimization (right column).

Chapter 6

Compression Algorithm for Efficient Storage and Rendering

6.1 Overview

The storage requirements for massive point clouds is very high. As the size of point clouds exceeds system memory capacity, the rendering algorithm cannot operate in-core. For out-of-core rendering, the bottleneck is the disk bandwidth. An effective compression can reduce disk bandwidth, speeding up the rendering and lowering the storage requirements.

In this chapter, we introduce an effective compression scheme based on Residual Vector Quantization (RVQ). The best data representation from the perspective of the GPU is a textured triangle mesh, as GPUs were build directly to process this kind of data. Another interesting representation, when a mesh is not available, is the point cloud itself. In this case, storing colors with each point may be inefficient, however using compressed textures with a Surfel (Surface Element) representation allows to render the data set efficiently without reconstructing the mesh. This work has been done together with Guenter Knittel, who suggested to use RVQ in texture and volume compression, and who has invented an improved seeding scheme for k-means based image compression [KP09]. This chapter is a literal citation of the work presented previously in [PK08], [PK09a] and [PK09b].

This chapter consists of two parts. The first part presents an application of RVQ to large textures, and shows an effective rendering algorithm for compressed textures. As it may be desired to consider a volumetric representation for reconstructed models, the second part of this chapter describes the application of RVQ to volumetric models. Also a volume rendering algorithm that operates directly on a compressed data is presented. A good example of volumetric models of trees reconstructed from photographs can be found in [RMD04].

6.2 Rendering with Compressed Textures

6.2.1 Introduction

Computer-graphics applications, such as computer games, aim at generating realistic images at very high frame rates. Typically, the objects in a synthetic scene are described by a relatively coarse surface model, such as a triangle mesh, to limit computation costs. The (photo-)realistic appearance is to the largest part achieved by mapping textures (images) onto the objects.

Such texture mapping, however, poses high demands on the computing device, or the graphics system in particular. Textures tend to be large in size, and need to be accessed very frequently. Applications aiming at a high realism or photo-realistic rendering might even map multiple textures onto each generated pixel. A high image quality will also require the use of expensive texture filter operations using kernels with large spatial extent.

Therefore we can find large and extremely fast memory systems on modern graphics cards, providing peak bandwidths in excess of 100GByte/s and capacities approaching and surpassing 1GByte. This, of course, comes at a high cost for the consumer, including high retail prices, high power consumption, expensive cooling, and large unit dimensions.

A high performance, however, can only be achieved if all textures an application will ever need reside in the fast video memory close to the graphics processing unit (GPU). Clearly, this sets an upper limit to the visual richness of graphics applications. More severely, applications such as geographic information systems (for example fly-through simulators) or location-based services (like navigation in urban environments) typically have to process huge images, far too big to fit into typical video memory systems. Even worse, quite many of such applications are supposed to run on portable devices.

Consequently, many efforts have been made to render from compressed textures, and thus to lower storage and bandwidth requirements. A prominent example is S3TC, also known as DXTC. It is part of both the DirectX and OpenGL graphics APIs, and therefore probably implemented in most if not all current graphics accelerators. This clearly demonstrates the economical relevance of a powerful texture compression scheme, and in this work we aim at further improvements over existing methods.

Texture compression, however, has its own set of requirements:

- fixed code length for random access to the compressed pixels,
- fast and inexpensive decompression,
- high compression rate at still acceptable image quality.

Thus, any compression scheme using entropy coding such as JPEG is not a good candidate for texture compression, despite other desirable properties like very high compression rates. The method we have chosen as basis is called Residual Vector Quantization (RVQ), which is an extension to standard vector quantization and which fulfills the above requirements. We aim at a compression rate close to 24:1 (1bpp from 24-bit RGB pixels).

Our application and test case is the rendering of landscapes using aerial photos of gigantic size. Thus, the compression is a very compute-intensive task. Adequately, we run the (offline) image compression on a NEC SX-8 supercomputer at the High Performance Computing Center Stuttgart (HLRS).

Our presentation system in turn consists of a display wall made from 16 LCDs, each with the resolution of 2560×1600 pixels. Each display is driven by a powerful PC, equipped with a modern graphics adapter with 1GByte video memory. Compressed textures, along with codebooks and other data structures, are loaded into the video memory and are processed in real-time by short programs running on the GPU. We are able to process images of size 80.000×80.000 (6.4G) pixels on a single graphics adapter at a peak decompression rate of 0.6Gpixel/s.

6.2.2 Related Work

This section briefly describes known methods and principles used for texture compression.

Block truncation coding (BTC), developed by Delp and Mitchell [DM79], is a simple scheme for image compression. The method compresses gray scale images by dividing them into tiles of size 4×4 pixels. The 16 gray values are replaced by two 8-bit values such that mean and variance are conserved. One bit per pixel selects which value to use. The Resulting compression ratio is therefore 2 bits per pixel.

An extension of BTC to color images was developed by Campbel et al. [CDF$^+$86]. The method is called color cell compression (CCC). They use a fixed size color palette of 256 RGBvalues. For each 4×4 pixel tile they store 16 decision bits as above, and two 8 bit indices into the color palette. This results in a 2bpp compression ratio.

The use of CCC for texture compression has first been proposed in [KSKS96]. Subsequently it was developed into the aforementioned S3TC [KI99]. In S3TC a block of 4×4 pixels is compressed into 64 bits. For each tile two base colors are stored in 16 bits each and each pixel has a 2 bit index into a local color palette. The local color palette contains two base colors and two additional colors that lie between them (in RGB color space). Compression rate for S3TC is 4bpp.

The fact of using only four colors has a negative influence on quality. Ivanov and Kuzmin [IK00] improve the quality by using colors from neighboring tiles.

The POOMA method presented by Akenine-Moeller and Stroem in [AMS03] is similar to the S3TC algorithm, but it compresses a tile of 3×2 pixels into 32 bits, which yields a compression rate of 4.5bpp. They use base colors with fewer bits and only one in-between color. The block size of 3×2 pixels is problematic for hardware implementations.

Perebrin describes another approach in [Per99]. Mipmapping and texture compression are combined under the assumption that box filtering is used for mipmaps. Each block of 4×4 pixels is converted into the YUV space. Wavelet methods are applied to luminance, while chrominance is subsampled before it is compressed. This method gives about 4.5bpp.

In [SAM04] Stroem and Akenine-Moeller present a compression scheme called PACKMAN. The method splits an image into 2×4 blocks and represents each block by 32 bits. For each block a base color is stored using 12 bits in RGB444 format. The next 4 bits are used to pick a table of four values from a codebook. For each pixel in a tile 2 bits are used as an index to components of a previously selected table. The selected component is added to each channel of the base color to produce the final color value for each pixel. The resulting compression rate is 6bpp. In [SAM05] the same authors present the iPACKMAN algorithm, which is an improved version of the original PACKMAN method. As a result, a 4bpp bit rate is obtained.

Lefebvre and Hoppe in [LH07] describe a quadtree based compression, but the method is not suitable for detailed color images and has a complicated and slow decoding algorithm. This method performs well for textures with smooth color variations. Compression bit rates range from 0.07bpp for binary images to 5bpp for high dynamic range images.

Most directly related to this work, Beers et al. proposed vector quantization (VQ) for texture compression [BAC96]. Codebook size was reduced by converting images into 4:1:1 YUV format. The compression ratio achieved by this method is as low as 1 or 2bpp. Residual vector quantization as a means to increase image quality was not considered in this work, though.

6.2.3 Residual Vector Quantization

Residual Vector Quantization (RVQ) has first been described in [JG82]. An excellent survey of RVQ and related techniques can be found in [BRN96].

RVQ is based on standard vector quantization (VQ). In VQ, a set of vectors is represented by a smaller set of vectors called *codevectors*, that is minimizing overall error. Usually, clustering methods like k-means [Llo82] are used to compute a set of codevectors called a *codebook*. K-means is starting from an initial set of random codevectors (*seeds*), and assigns each vector to its nearest codevector in order to create clusters. In the next step, the codevectors are moved to the center of gravity of their cluster. This step is repeated until the codevectors no longer move. Each vector is represented by a pointer to its codevector in the codebook. Decompression step for a vector is returning a codevector pointed by the pointer.

In case of the large or unknown (at the time of codebook construction) set of vectors, a subset of vectors (*training set*) can be used to create the codebook. Vectors from the outside of the training set are replaced by a pointer to its nearest neighbor vector.

For RVQ the data set is decompressed and for each original vector, the error vector is computed. The set of error vectors is compressed with VQ, to obtain the second set of pointers and the second codebook. This process is repeated for the desired number of levels. Compressed vector is represented by a set of pointers to codebooks. To decompress such a vector it is necessary to add the codebook vectors associated with a compressed vectors.

The code length for RVQ is defined by number of bits required to represent a set of pointers (indices) to the codebooks.

6.2.4 Texture Compression Using RVQ

For a high compression rate large vectors must be chosen. In our case, a vector is formed by a square image tile of 8×8 pixels. For RGB images, the vector dimension is 192. For our target compression rate of 1bpp, a tile is compressed into 64 bits. This gives a certain choice of number of levels and codebook sizes. We have found that larger codebooks should be favored over a high number of levels.

Commonly, the representative vectors in a codebook are found using some kind of clustering algorithm. We have examined kmeans (or Generalized Lloyd's Algorithm [Llo82]), kmeans++ [AV07], and neural gas [MBS93]. Results for a test image of 1024×768 pixels and 8 levels are shown in Table 6.1.

Codebook Size	kmeans Error [dB]	kmeans++ Error [dB]	Neural Gas Error [dB]
16	28.68	28.63	28.8
64	31.26	31.17	31.8
144	33.0	32.99	33.84
256	34.58	34.75	35.8

Table 6.1: Performance of different clustering methods.

As can be seen, neural gas performs better for increasing codebook sizes, however, compression times become excessively long. So we opted for using kmeans++ on 8 levels of 256 codevectors each for our 80k×80k image. In this way we achieved a PSNR of 29.16dB, using a cutout of 4k×4k (16M) pixels as training set. Compression rate is practically 1bpp, including codebooks using 2 bytes per color channel.

On this cutout image we performed a comparison to the S3TC industry standard. S3TC using 4bpp achieved 34.59dB. RVQ on this training set alone achieved 30.42dB, at a rate of 1.4bpp (including codebooks). Some detail images are shown in Fig. 6.2. Although there is a certain difference in visual appearance, fine details are still preserved and image quality is acceptable given the low bit rate.

The compression code has been optimized to run on the NEC SX-8 supercomputer. This is a cluster of multiprocessor nodes with vector processors. One node contains 8 vector CPUs and 128GByte of main memory. This particular installation contains 72 such nodes, of which typically 8-16 can be requested for a compute job. For code to run efficiently on such a machine it must be vectorized as well as parallelized on both the thread- and process-level. Image data is partitioned and distributed among the nodes, and only the small codebooks need to be exchanged after each kmeans-iteration using suitable MPI-routines (*Message Passing Interface*). In this way we achieve a very good scalability.

Clustering time is mainly consumed by nearest-neighbor searches. To speed up this operation, we use the VP-Tree algorithm [Yia93]. The VP-Tree searches have been modified to eliminate recursive function calls, which can reduce efficiency dramatically on this machine. In this way we achieved a ratio for vectorizable code vs. scalar code of 93%. Still compression times are in the order of days.

6.2.5 Real-Time Rendering From Compressed Textures

During rendering, geometric primitives forming the object surface are decomposed into the set of screen pixels they cover (scan conversion). The projection of a screen pixel on the surface defines the set of texture pixels (texels) which contribute to the screen pixel color. This set can be very large, depending on the distance and orientation of the geometric primitive. Thus, filter operations on the texture need to be done to avoid aliasing artefacts on a per-pixel basis.

Typically this is done by tri-linear interpolation in a mipmap (a pyramid of prefiltered versions of the texture). Although being relatively small in size, these downfiltered images still consume memory. Using RVQ, we can avoid this in an elegant way: instead of using prefiltered images, we can use a hierarchy of *prefiltered codebooks*. During scan conversion, the distance of a pixel to the observer can be used as a measure for the size of the pixel projection on the texture, and thus to select the proper codebook version.

However, if a pixel projection on the surface has a high aspect ratio, anisotropic filter kernels are required. Typically, such kernels are constructed by averaging multiple trilinear interpolations. These are expensive operations, for which most graphics accelerators offer dedicated hardware support.

Unfortunately, these hardware units do of course not support our compression format. Performing all these filter operations in software is too slow, even on today's high performance graphics chips.

As a solution we have implemented a two-pass approach. An auxiliary texture, much smaller in size than the compressed texture, is allocated in video memory. For each triangle to be rendered, we first decompress all texels which cover that triangle,

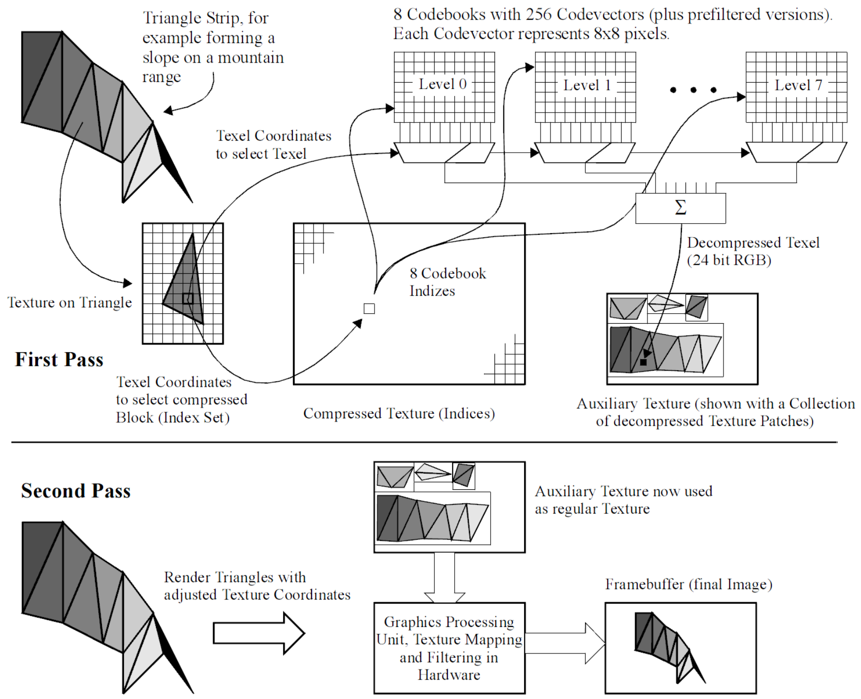

Triangle Strip, for example forming a slope on a mountain range
8 Codebooks with 256 Codevectors (plus prefiltered versions).
Each Codevector represents 8x8 pixels.
Level 0
Level 1
Level 7
Texel Coordinates to select Texel
Σ
Decompressed Texel (24 bit RGB)
8 Codebook Indizes
Texture on Triangle
First Pass
Texel Coordinates to select compressed Block (Index Set)
Compressed Texture (Indices)
Auxiliary Texture (shown with a Collection of decompressed Texture Patches)
Second Pass
Auxiliary Texture now used as regular Texture
Render Triangles with adjusted Texture Coordinates
Framebuffer (final Image)
Graphics Processing Unit, Texture Mapping and Filtering in Hardware

Figure 6.1: Two-Pass Rendering from Compressed Textures

and write them into the auxiliary texture. This is repeated until the intermediate texture buffer is full.

In a second rendering pass, we render the corresponding set of triangles, with adjusted texture coordinates, using the just constructed texture. When this second pass is finished, we construct the auxiliary texture anew for the next set of triangles and repeat until the scene is finished. For increased efficiency, the set of triangles can also include triangle strips.

Since the second pass is standard rendering procedure, we can use all the hardware support the GPU has to offer for texture filtering. So we can combine the advantages of both worlds: keeping huge texture in video memory, and applying expensive filter operations for high image quality. Rendering speed is still high, as will be detailed in Section 6.2.6.

A block diagram of our two-pass rendering algorithm is shown in Fig. 6.1. The codebooks are stored as floating point textures (16 bits per channel). The compressed image is stored as a texture in RGBA4 format (4 channels, 4 bits each), using a block of 2×2 texels for the eight codebook indices. For each texel to be decompressed, the GPU program reads the corresponding index set, and selects the proper element from each codevector. The sum of these elements gives the texel color. Although the GPU program needs to access video memory 12 times to decompress a single texel, decompression rate is still high (see Section 6.2.6). We take advantage of the fact that modern graphics systems are optimized for multiple texture accesses per pixel. Moreover, since neighboring texels have the same indices, texture reads are cached, which improves the performance further.

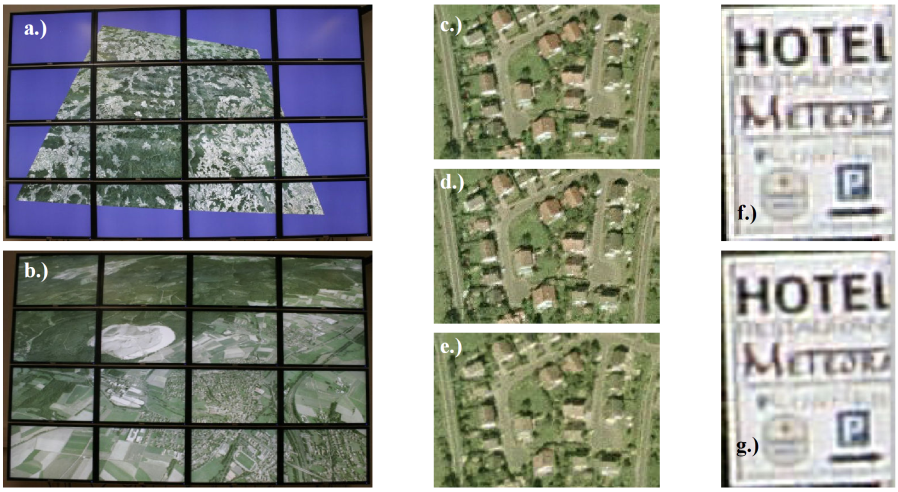


Figure 6.2: The display wall showing the aerial image of size 80k×80k. The texture is loaded redundantly on all graphics cards for fast zooming and panning (a and b). Close-ups of the original image (c), compressed using S3TC (d), RVQ on 8 levels of 256 codevectors (e), RVQ without filtering (f), RVQ including two-pass rendering with filtering (g).

6.2.6 Results

We measured the following performance figures with NVIDIA 8800 GT graphics cards with 1GByte of video memory. The host PCs are equipped with dual-core CPUs and 4GByte of main memory. The pure decompression speed into the auxiliary texture buffer (First Pass in Fig. 6.1) is 598MTexel/s. The real-life frame rate as seen on the display wall depends on the magnification, however, it never dropped below 36 frames/s in 4MPixel resolution per LCD.

6.3 Rendering of Compressed Volume Data Sets

6.3.1 Introduction

Nowadays, volume rendering has become a standard in medical applications and visualization of scientific simulations. Visualization requirements are increasing: data sets and display resolutions are getting larger. This leads to more complex rendering algorithms. In this work we present a quite extreme example: we render a very large data set (about 7.5G voxels) on a high-resolution display wall (65,536,000 pixels). Our display system (see Fig. 6) is built using sixteen LCDs of a resolution 2560x1600 each. A cluster of 16 PCs drives the display wall. Each PC is connected to one LCD, and is equipped with an Intel Core 2 Duo 2.4GHz processor, an NVidia Geforce 8800GT graphics card with 1GB of video memory and 4GB of system memory. The PCs are connected via GBit Ethernet.

Today's consumer graphics cards are approaching 150GB/s of peak memory bandwidth and a teraflop of computational performance, so they can outperform CPUs in many tasks. In order to achieve the highest performance in rendering, the most computation-intensive tasks are off-loaded to the GPUs by using NVidia's CUDA SDK. While the computing platform presented here is quite powerful, it has a few bottlenecks that can seriously limit the performance and should be avoided. One of the bootlenecks is the slow gigabit ethernet and the other is the PCIe 1.x bus. Our system is designed to minimize any data traffic sent over those mediums.

6.3.2 Related Work

Data Set Compression

The authors present lossless compression methods for volume data in [FY94]. The focus of this work is to reduce storage requirements, rather than improving rendering speed. Maximum reported data reduction is about 50% for selected data sets.

Vector quantization for volume rendering was first introduced in [NH92], with some improvements in [SW03]. The presented system renders directly from compressed data, but the nearest-neighbor interpolation limits rendering functionality.

In other work, a Block Truncation Coding is used in a space filling way to limit the memory bandwidth [Kni95].

In recent years Wavelet-based coding has received the most attention [IP99], [NS01], [Rod99], [RGW$^+$03]. Some extensions based on a hierarchical wavelet representation of large datasets are used in [GWGS02]. The claimed compression rate without noticeable artifacts in the image was 30:1. The authors minimize the number of voxels processed by deriving a quality measure from the wavelet representation and achieve interactive rendering speeds for large data sets on standard PCs. The decompression is done on the CPU however, sending uncompressed voxels to the graphics card over the bus can seriously limit performance (see Table 6.2).

GPU-based Volume Rendering

As our rendering algorithm uses the compressed volume directly, we shortly recall state-of-the-art work related to the volume rendering.

Graphics hardware is used for volumetric rendering in [Ake93], [CCF94] and [CN94]. The data is kept in a 3D texture and screen-aligned slices are drawn with properly computed mapping coordinates and blended together. Later the visual quality is improved with gradient shading [MHS99], multi-dimensional transfer functions [KKH01], pre-integrated transfer functions [EKE01], and the processing of presegmented data sets [HBH03]. We integrate these state of the art methods into our system, where it is possible and useful. Although, some features have lower priority, for example data segmentation is not included due to the large effort required for this task. This feature may be supported in later versions of our system.

Parallel Volume Rendering

As described in Section 6.4.7, parallel rendering can be implemented in two ways: object-space partitioning and screen space partitioning. Object-space partitioning usually is limited by alpha blending of the intermediate images. Solutions are proposed in [lMPH94], [SML$^+$03] and [SMW$^+$04]. In [SMW$^+$04], it is pointed out that the CPU is used that for alpha-blending, instead of the much better suited GPU. Although, GPU can be used to speed up the computation, large data streams are still difficult to handle in the network. Isosurface rendering on a display wall of about 63M pixels is described in [MT03]. Example is shown for isosurfaces built from 470M triangles, the rendering takes about 15 seconds.

6.4 The Giga-Voxel System

In our rendering system we off-load all time consuming computations to the GPU. This choice is supported by a few benchmark figures. Results obtained on a Dell XPS700 workstation, equipped with an Intel Core 2 Duo CPU at 2.13GHz and an NVidia GTX280 (optionally an 8800GT) are summarized in Table 1. To measure the bandwidth we have used 'bandwidthTest' from the NVidia CUDA SDK [Cor].

CPU ↔ Cache	98,520
CPU ↔ Memory, 16MB Blocks	2,100
CPU ↔ Graphics Card (PCIe 1.x)	1,500
GPU ↔ Video Memory GTX280	110,028
GPU ↔ Video Memory 8800GT	43,357

Table 6.2: Bandwidth Measurements [MB/s]

An interesting measurement is the internal CPU cache bandwidth and the bandwidth to the external video memory on the GTX280. As it can be seen, the latter one is better. Our design target was set to keep all necessary data locally in video memory and to avoid frequent transfers of big amounts of data between GPU and GPU. The GPU is used for all compute-intensive tasks like decompression, shading and ray casting. Each cluster node must have a copy of the data in order to reduce network traffic to minimum. When the size of typical data sets it taken into account, it becomes clear, that a compression scheme must be employed. There are contradicting requirements for the compression scheme: high compression rate at high quality and extremely fast decompression time. An interesting candidate is Residual Vector Quantization (RVQ).

The data needs to be compressed in an offline processing step. This step is performed only once. Later the compressed data is reused for rendering. Compression and rendering is described in the following subsections.

6.4.1 Residual Vector Quantization for Volume Data Sets

The general information about Residual Vector Quantication (RVQ) is described in Section 6.2.3. In this subsection we discuss the selection of codebook size and number of levels of RVQ for the compression of volume data sets.

Experiments with a large number of images have shown, that larger codebooks should be preferred over a high number of levels in order to achieve a higher PSNR. We have performed a number of tests showing that good quality can be achieved with 4 levels and 4096 codebook vectors per level, what gives a codelength of 48 bits.

We use the 'Visible Human Female' color (RGB) data set for our tests. Each voxel is defined by 3 channels, 8 bit each. We create vectors of dimension 192 from $4 \times 4 \times 4$ voxel cubes. The 64 voxels in a cube are represented with 48 bits, giving a compression rate of 32:1, or 0.75 bits per voxel. We store codebook vectors in higher precision to decrease the influence of rounding errors. 32-bit values are used to represent one codebook vector voxel. 11 bits are used for red and green, and 10 bits for blue. One codebook vector is 256 bytes, and the whole codebook for 4 levels takes 4MB of video memory.

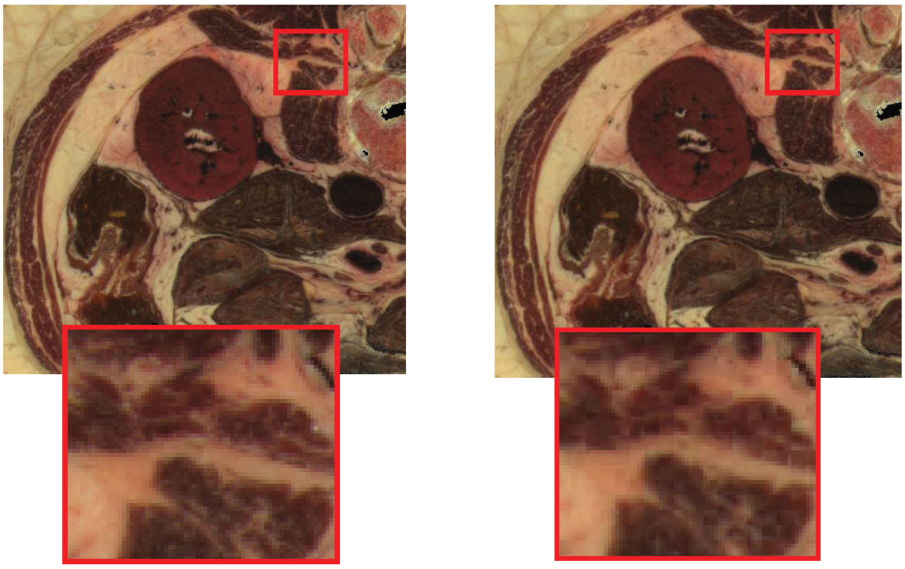


Figure 6.3: Compression example: original image (left) and decompressed image (right).

6.4.2 Compressing the Visible Human Female

The Visible Human data set is available for download as a set of images of 2048×1216 pixels [25]. There are 5189 images. The body was frozen in a blue gel. We treat this as an empty space, so we had to remove it from the data set. We have cropped the images to a final resolution of 1608×896 pixels, because there is too much of empty space. The total input data size is 20.9GByte. In order to limit the compression time, we have selected a training set equivalent in size to 300 images. Training phase to obtain four codebooks took about 21 hours on an eight-core machine. Codebook construction time can be improved significantly performing nearest neighbor searches in parallel. Reusing the codebook to compress the data set took additionally about 10 hours.

We have computed the quality of the decompressed data set in terms of PSNR. We didn't include 4×4×4 cubes that are forming the empty space. The overall PSNR is about 27dB. An example of an original image versus the decompressed image is shown in Figure 6.3.

The compressed data is an array of 402×224×1297 = 116,792,256 index sets of 48 bits each, with a total size of 700,753,536 Bytes. The entire compressed data set along with the codebooks fits on a graphics card with 1GByte of video memory. We only consider the case that the compressed dataset fits completely into the video memory. In the other case swapping from main memory or even hard disk would be required. This also would benefit from the high compression rate.

6.4.3 Rendering

The dataset is divided into subsets of subvolumes of $n \times n \times n$ voxels with the same Manhattan distance. The value of n depends on selected level of detail. We render each previously mentioned subset to an off-screen buffer and blend the resulting image

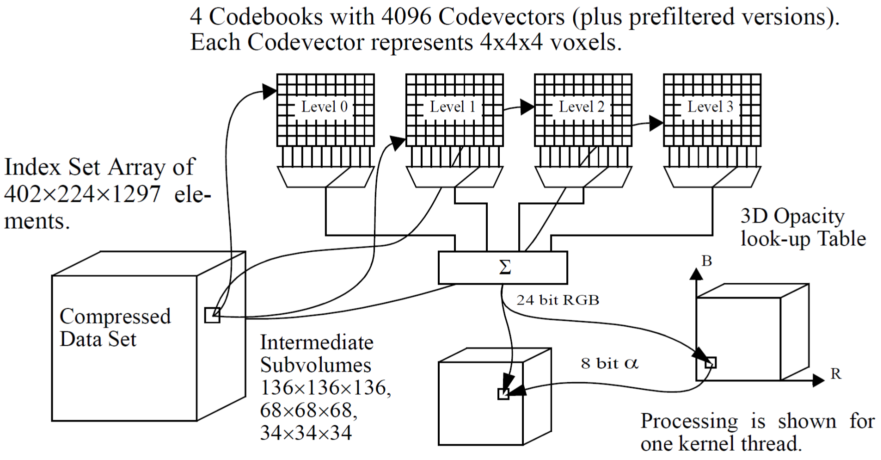

4 Codebooks with 4096 Codevectors (plus prefiltered versions).
Each Codevector represents 4x4x4 voxels.
Level 0
Level 1
Level 2
Level 3
Index Set Array of 402×224×1297 elements.
3D Opacity look-up Table
Σ
B
24 bit RGB
Compressed Data Set
Intermediate Subvolumes 136×136×136, 68×68×68, 34×34×34
8 bit α
R
Processing is shown for one kernel thread.

Figure 6.4: Decompression and classification.

with the framebuffer. To render a subset of subvolumes we repeat the steps of decompressing a subvolume into a 3D texture buffer and rendering the 3D texture using ray-casting. In the decompression step we integrate classification using a 3D lookup table. Optionally gradient extraction and shading is also integrated into the decompression step in order not to slow down the ray caster. We use early ray termination on a per-ray basis and occlusion culling for an early exit on a subvolume basis. We apply empty space skipping to subvolumes after classification. When a visible contribution of a subvolume is under a user-supplied threshold, the subvolume is discarded from rendering. This test is done according to the actual transfer function. Multi-resolution rendering is integrated in an elegant way - we downsample the codebooks to create two additional levels of detail. In the following subsections we present individual steps of the rendering pipeline in detail.

Decompression

The decompression code is implemented with the NVidia CUDA technology. We make use of the on-chip shared memory buffer in order to reduce transfers from video memory. Processing is as follows. We load 256 index sets (worth 16k voxels) into the shared memory. Each index set consists of 48 bits, which are unpacked into four 16-bit indices again into shared memory. For each voxel to be generated, there is one thread in the kernel. Each thread reads from memory those elements of the codebook vectors which it needs for its voxel. The codebook vector element is unpacked from the R11G11B10 format (see Section 6.4.1) and accumulated in the shared memory.

Unpacked RGB values of a voxel are used to access a 3D lookup-table with opacity (α) values. The α-value is again written into the shared memory, which completes the voxel generation. When a certain number of voxels is decompressed, they are written to an intermediate 3D texture. We take care that the memory transfers are mostly large bursts, to maximize the bandwidth. Decompression performance is 1.86G voxels/s on the GTX280, and 0.60G voxels/s on the 8800GT.

We partition the volume into subvolumes of $128\times128\times128$ voxels. We extend each subvolume in each direction by two layers of $4\times4\times4$ voxels, so the final subvolume

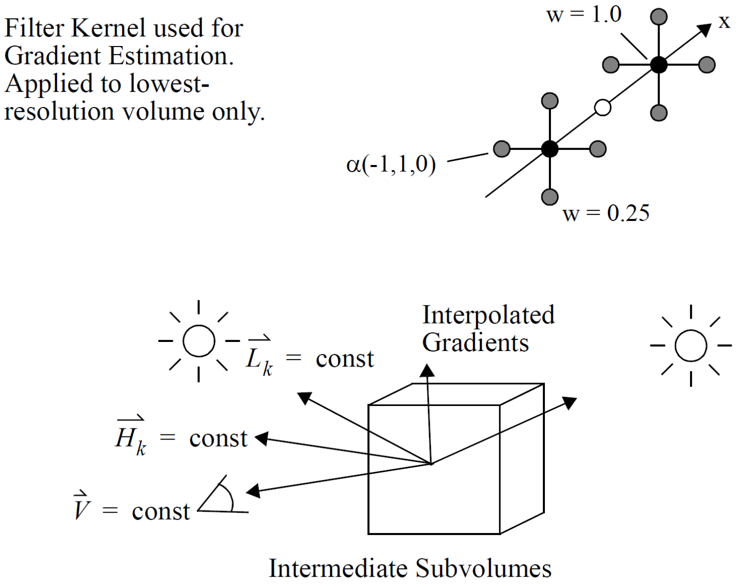


Figure 6.5: Gradient shading

size is 136×136×136. In this way we solve the problem of missing voxels at boundaries for the reconstruction filter in the ray caster (tri-linear interpolation) and during gradient extraction (see Section 6.4.3). In this way we an the overhead of about 20%. Decompression performance for different levels of detail is summarized in Table 6.3. The overall decompression and classification flow is presented in Figure 6.4.

Gradient Extraction and Shading

GPU	Level	Voxels/s	Subvolumes/s
8800GT	0	0.60G	254
GTX280	0	1.86G	776
8800GT	1	0.20G	716
GTX280	1	0.72G	2463
8800GT	2	0.03G	841
GTX280	2	0.06G	1667

Table 6.3: Decompression Performance.

After a subvolume has been decompressed, our system can perform gradient shading as an option. We compute the gradient from the opacity, because steep changes in opacity represent the surfaces. We use a variant of a 3×3×3 Sobel filter (see Figure 6.5). We need to address two problems:

1. high computation costs due to the large kernel,
2. a certain amount of noise in the volume.

Both problems are solved by using downsampled versions of the subvolume for gradient estimation (see also Section 6.4.6, Multi-Resolution Rendering). We generate two additional levels of detail, a 68^3, and a 34^3 subvolume. We compute the gradients only on the lowest-resolution grid, directly on the GPU, using a CUDA-kernel. The performance is given in Table 4.

We do not affect the raycaster performance by the shading operation, because gradient extraction and shading are done at the voxel positions, and the contributions from specular reflection are added to the just decompressed RGB-quantities. The decompression speed does not suffer too much because of the regular memory access pattern.

For gradient extraction the system can use a Central Difference (CD) operator. Gradient shading is again implemented as a CUDA-kernel in a way that each thread processes one voxel. Each thread reads a certain subset of the required voxel neighborhood, so that by the end of this step a large block of voxels resides in shared memory.

For the performance reasons we assume that light sources are located at infinity and a constant viewing direction throughout the subvolume (only for the shading, not for the raycasting). Thus, we do not need to recompute the halfway vectors for each voxel. This approximation does not produce disturbing effects caused by misplaced highlights. Exponentiation is done by a look-up in a precomputed table. Gradient shading speed for CD and one light source is summarized in Table 6.5. The gradient computation and shading is presented in Figure 6.4.

GPU	Gradients/s	Subvolumes/s
8800GT	17.8M	570
GTX280	63.9M	2045

Table 6.4: Gradient Estimation Performance.

GPU	Level	Voxels/s	Subvolumes/s
8800GT	0	0.227G	111
GTX280	0	0.411G	201
8800GT	1	0.309G	966
GTX280	1	0.546G	1712
8800GT	2	0.271G	7426
GTX280	2	0.482G	13158

Table 6.5: Gradient Shading Performance (CD).

6.4.4 The Raycaster

The basics of the ray casting are presented in Figure 6.6 (a). We use the raycaster from the NVidia SDK. No attempt to optimize this code was made, because it is not the scope of this work.

The average rendering time of this raycaster was measured 3.97ms per subvolume (on the GTX280 graphics card), with early ray termination disabled. It is about 4 times slower than the pure decompression. Recurring decompression can be tolerated fairly well, because most of the rendering time was spent in ray casting.

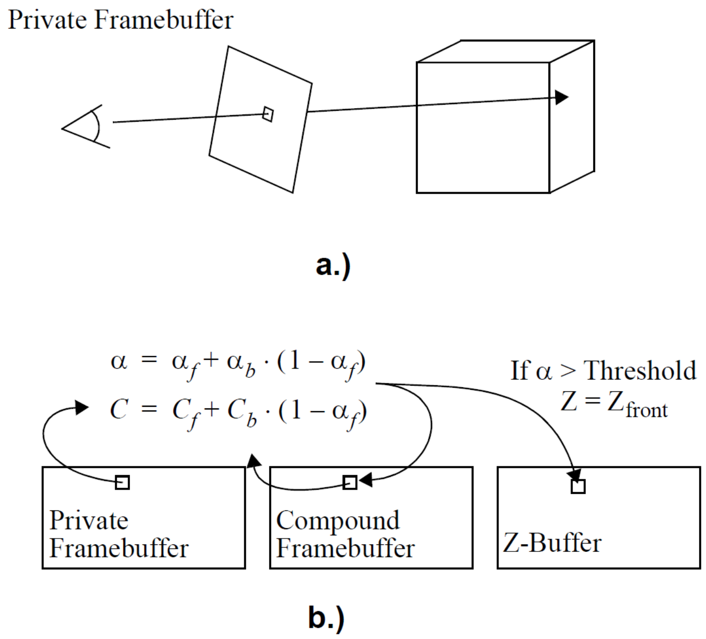


Figure 6.6: Ray casting (a) and alpha-blending (b).

6.4.5 Blending and Occlusion Culling

The subvolumes are rendered in front-to-back order. We divide subvolumes into non-overlapping subsets (in screen space) This is done by sorting subvolumes according to their Manhattan distance to the viewer. The result of the rendering of one subvolume subset is a private frame buffer of RGBα-values. This buffer is α-blended with the compound frame buffer, which in the end contains the final image.

During blending, the Z-buffer is also updated. Whenever the α-value of a pixel in the compound frame buffer exceeds a threshold, the corresponding entry in the Z-buffer is set to Z-front. This is then used to exclude subvolumes which are occluded by opaque structures in the data set from decompression and rendering. An OpenGL occlusion query is submitted with the bounding box of the subvolume, which returns the number of visible pixels. Depending on a user-defined parameter, the subvolume is rendered or discarded.

Occlusion queries can be accelerated by submitting a set of bounding boxes. In our system, all subvolumes with the same Manhattan distance could be rendered in parallel and in any order, so they are queried in one batch. We exclude subvolumes which are located at any of the visible faces of the entire volume from the occlusion query. The blending process is illustrated in Figure 6.6 (b).

6.4.6 Multi-Resolution Rendering

As mentioned before, a downsampled version of the subvolume can be generated from a *downsampled version of the codebooks*. Thanks to this fact there is no need to store separate index sets for each level of detail in video memory. We need only a small

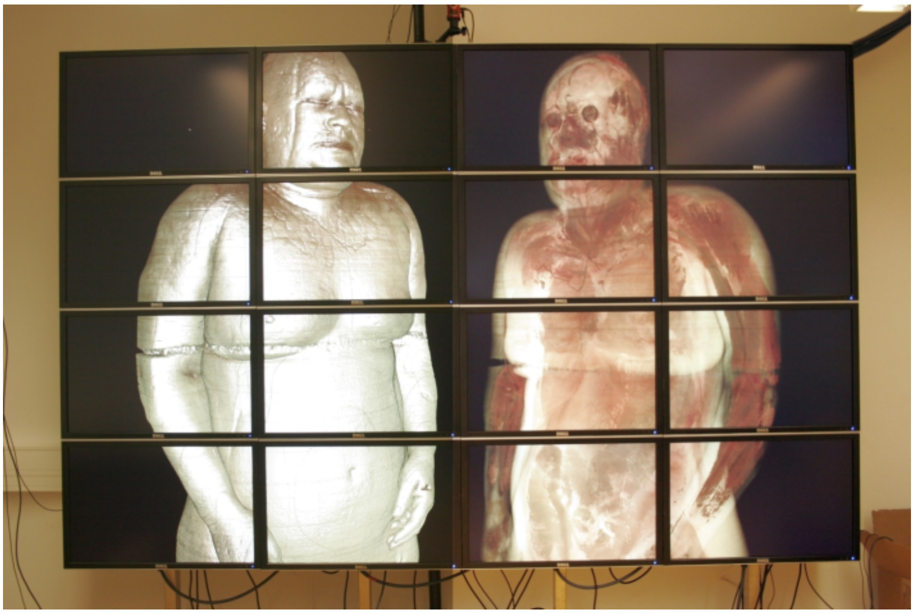


Figure 6.7: Volumetric rendering on the PowerWall (1).

amount of extra memory for storing downsampled codebooks.

Different resolutions are used to avoid subsampling of the data during ray casting and to speed up the rendering of distant subvolumes. The system can decompress subvolumes with 136, 68, and 34 voxels along each axis. The raycaster automatically performs proper voxel access and filtering by using normalized texture coordinates. Only the opacity must be adjusted, we accomplish this by using a separate 3D look-up table for each resolution.

6.4.7 Parallel Rendering on the Cluster

There are two possible ways of distributing work among rendering nodes: object-space partitioning (OSP) and screen-space partitioning (SSP). For the first method, a work-package is a subvolume that is rendered into a private frame buffer. A subvolume can be visible on more than one display, so it may be necessary to send a subset of pixels over the network to the receiving node. This subset of pixels is then blended into the local composite frame buffer on the target node. Since the contribution of many subvolumes may influence one pixel, multiple transfers over the network may be required to compute the final color. Many advanced algorithms have been designed to optimize this operation [14], [23], but for slow GBit Ethernet it may be still a bottleneck. The advantage of this approach is that each subvolume is processed only once.

In SSP, the screen is split into a set of tiles. A machine that is assigned to a certain tile, renders all subvolumes contained in the tile view frustum and sends the final pixels to the destination screen. We have selected this method, because in our setup it will give a higher frame rate. Often it may happen that a subvolume is visible on multiple tiles – in this case it has to be processed multiple times. In case of ray casting there is no redundant processing, but the decompression code can generate only complete subvolumes. Since the viewport can be in arbitrary location, each node needs to store

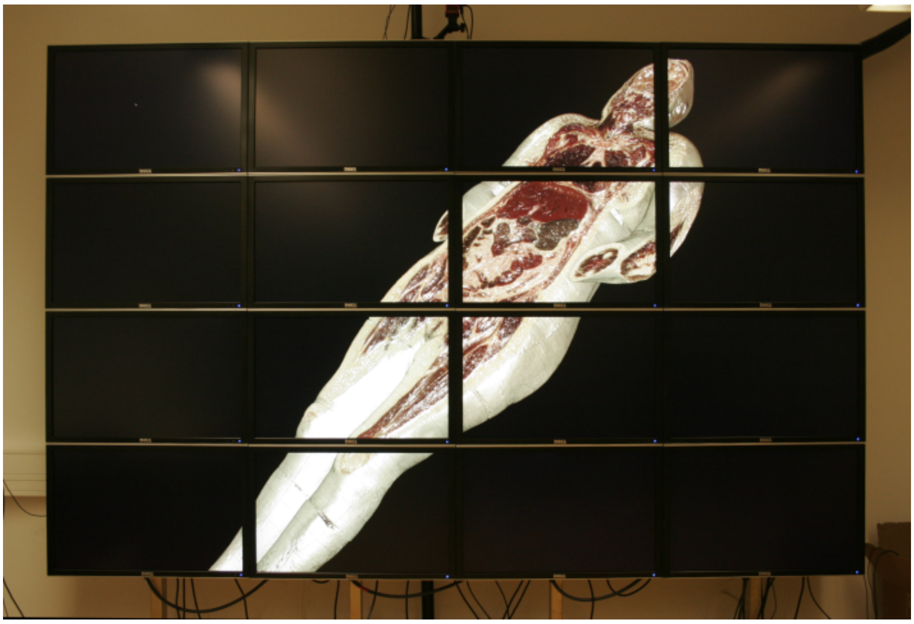


Figure 6.8: Volumetric rendering on the PowerWall (2).

a complete copy of the data set. In the cluster we have only 1GB of video memory per graphics card, the compressed data set takes about 700 MB. We use high resolution off-screen buffers for blending and a certain amount of video memory is allocated for internal OpenGL data structures.

Subvolume Caching

For a significant performance improvement, a caching scheme for the decompressed subvolumes is used. In order to reduce multiple processing of the same subvolumes by the decompressor, we cache the 128 most frequently used subvolumes. We use caching in a fast preview mode only, when the lowest level of detail is used. On the higher levels of detail, the decompressed subvolumes consume too much memory. Preview mode is used for camera motion, and we sacrifice rendering quality for speed. The caching scheme has proven to be very efficient also in case of camera motion, when temporal coherency is exploited. The effect of caching on the rendering speed in preview mode is summarized in Table 6.6. For a first frame of the rendering, when the temporal coherency cannot be exploited yet, we achieve an average hit ratio of 58%. For consequent frames, when temporal coherency can be exploited we achieve an average hit ratio of 89%.

Rendering method	Cache hit ratio	Time (s)
no caching	-	0.50
caching	58%	0.39
caching with temporal coherency	89%	0.25

Table 6.6: Impact of caching on the rendering speed.

The Tile Manager

Further performance optimization is achieved by a smart load balancing scheme based on dynamic scheduling and a set of work assignment rules. Assignment of tiles to the rendering nodes is done on demand. A tile management thread (*the tile manager*) is running on one of the cluster nodes. When a node needs to have a tile assigned, it sends a request to the tile manager and as a response gets a tile number to render. Since remote rendering causes network data transfers, the tile manager is using some heuristics during tile assignment to reduce network traffic and avoid network collisions. In the Ethernet, network collisions can happen when a few nodes send data to the same machine at the same time. It can cause large delays and it can limit target machine network bandwidth.

The tile manager uses the following rules for a tile assignment to a rendering node: (1) At first it tries to assign a local tile to a rendering node. (2) If it is not possible, it tries to assign a tile that was assigned to this node in the previous frame. (3) If it is not possible, it tries to assign a tile that is located on a different machine than the one of the previously assigned tile. The first rule tries to eliminate network traffic. Rules (1) and (2) improve cache hit ratio for local and remote rendering. Rule (3) reduces per-node network traffic and collisions for remote tiles, by trying to guarantee that tiles rendered in parallel on different machines are not sent to the same node. The tile manager runs 16 threads that are accessing a shared data structure holding information about the tile assignment. Threads are synchronized with critical sections. Only one thread at a time can access the shared structure, while other threads waiting to enter the critical section are descheduled. This approach does not overload the CPU.

Each machine is running one rendering thread and 15 threads are used for receiving pixel data from the network. Rendered tiles are received in the background of the rendering process and they do not affect the rendering speed on our dual core machines. Rules used by the tile manager try to guarantee, that the number of threads receiving data at the same time is minimized. At the end of a frame, each node checks if it has all tiles. After receiving a synchronization signal, each node plots its tiles to the screen. In the preview mode, the tiles are rendered in a lower resolution by shooting one ray per 4×4 pixel block, so the amount of data transferred is reduced 16 times. As an option, rendered tiles can be compressed before sending over them the network. The image compression is described in the following subsection.

Real-Time Image Compression and Decompression

We have implemented a variant of the S3TC compression algorithm. The built-in texture compression functionality of OpenGL is not suitable to be used in our case. While there is a dedicated S3TC decompression unit on the graphics hardware, the process of compression is implemented on the CPU and it is slow. In other applications it usually does not have an impact on the performance, because textures are compressed just once, and they are reused many times during the application run time. However in our case every frame we need to compress tiles before sending them off and we cannot afford an expensive compression algorithm. The other disadvantage of the original S3TC algorithm is a compression ratio of 4 bits per pixel.

We have implemented a variation of S3TC algorithm directly on the GPU using the NVidia CUDA technology. It has the advantage of massively parallel processing and access to the data stored on the GPU. The rendered tile is kept in video memory in a frame buffer object. With the current implementation of CUDA it is not possible

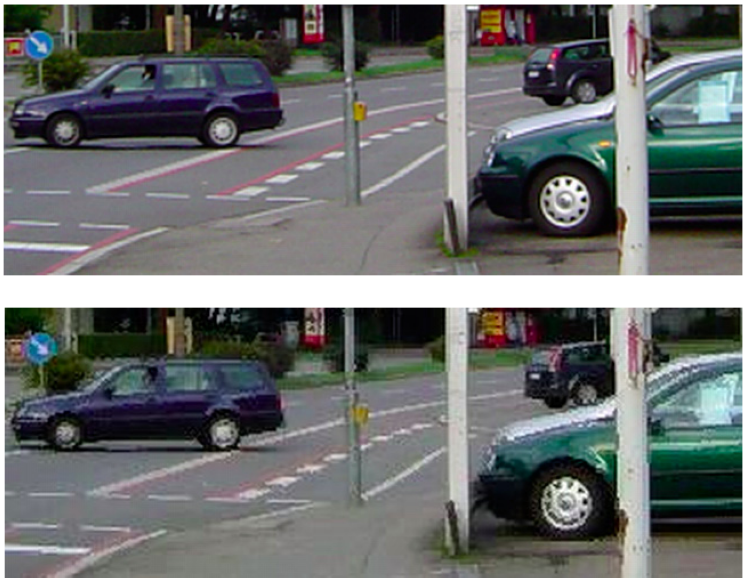


Figure 6.9: The origial (top) and the decompressed image (bottom)

to use a frame buffer object directly. A frame buffer has an object to be copied to a pixel buffer object before the CUDA kernel can process the rendered data. However it is much faster to copy data within the video memory than between the system and the video memory.

Our implementation of the S3TC algorithm is compressing blocks of 4×4 pixel in a lossy manner. An average luminance L of a pixel block is computed. Then the average luminance is used to divide the pixels into two sets. Pixels with higher luminance then L are assigned to the first set and the remaining pixels to the second set. For each set an average color is computed. In a compressed representation we store two average colors and for each pixel we store a value pointing to one of the colors in a lookup table. Each color is stored in R5G6B5 format, what gives 16 bits, and the lookup table stores 16 one-bit values. Finally 48 bits are needed to represent a tile of 4×4 pixels giving a ratio of 3 bits per pixel. Performance figures for the compression and the decompression are presented in Table 6.7. The GPU times shows only kernel times and do not include the memory copy time. A comparison between original and decompressed image is presented in Figure 6.9.

	CPU	GTX280	8800GT
compression	35.35 ms	0.18 ms	0.55 ms
decompression	4.65 ms	1.10 ms	0.70 ms

Table 6.7: Image compression and decompression speed.

6.4.8 Performance

A photo of the display wall showing a rendering of the Visible Human Female is shown in Figure 6.7 and 6.8. The average rendering speed for different levels of detail and without caching is summarized in Table 6.8. The column 'Res' shows the tile resolution level, for example f means full resolution, 'f/4' means 1 ray for a pixel block of 2×2, and 'f/16' means 1 ray for a pixel block of 4×4. In the preview mode, one ray was shot for each 4×4 pixel block and the data set was decompressed at the lowest level of detail.

LOG	Res	Shading	Time (s)
0	f	n	2.6
1	f	n	1.4
2	f	n	1.0
0	f/4	n	1.7
1	f/4	n	0.8
2	f/4	n	0.55
0	f/16	n	1.4
1	f/16	n	0.6
2	f/16	n	0.25
0	f	y	3.7
1	f	y	1.7
2	f	y	1.4
0	f/4	y	2.5
1	f/4	y	1.0
2	f/4	y	0.9
0	f/16	y	1.8
1	f/16	y	0.7
2	f/16	y	0.6

Table 6.8: The rendering speed.

Bibliography

[AFS$^+$10] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Brian Curless, Steven M. Seitz, and Richard Szeliski. Reconstructing rome. *Computer*, 43:40–47, 2010.

[Ake93] Kurt Akeley. Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 109–116, New York, NY, USA, 1993. ACM.

[AMS03] Tomas Akenine-Möller and Jacob Ström. Graphics for the masses: A hardware rasterization architecture for mobile phones. *ACM Transactions on Graphics*, 22:801–808, 2003.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[BAC96] Andrew Beers, Maneesh Agrawala, and Navin Chaddha. Rendering from compressed textures, 1996.

[BDH03] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, March 2003.

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[BH01] Cüneyt F. Bazlamaçci and Khalil S. Hindi. Minimum-weight spanning tree algorithms a survey and empirical study. *Comput. Oper. Res.*, 28(8):767–785, July 2001.

[Bra00] G. Bradski. The OpenCV Library, 2000.

[BRN96] Christopher F. Barnes, Syed A. Rizvi, and Nasser M. Nasrabadi. Advances in residual vector quantization: a review. *IEEE Transactions on Image Processing*, 5(2):226–262, 1996.

[BSL$^+$09] Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta, editors. *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information*

Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada. Curran Associates, Inc., 2009.

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*, VVS '94, pages 91–98, New York, NY, USA, 1994. ACM.

[CDF$^+$86] Graham Campbell, Thomas A. DeFanti, Jeff Frederiksen, Stephen A. Joyce, and Lawrence A. Leske. Two bit/pixel full color encoding. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 215–223, New York, NY, USA, 1986. ACM.

[CN94] Timothy J Cullip and Ulrich Neumann. Accelerating volume reconstruction with 3d texture hardware. *Radiation Oncology*, 61:1–6, 1994.

[Cor] NVIDIA Corporation. Cuda zone - http://www.nvidia.com/object/cuda_home.html#.

[CPS$^+$07] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *IEEE International Conference on Computer Vision*, 2007.

[CS08] Li Chen and F. W. M. Stentiford. Video sequence matching based on temporal ordinal measurement. *Pattern Recogn. Lett.*, 29(13):1824–1831, October 2008.

[DBL07] *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2007.

[DBL08] *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE Computer Society, 2008.

[DM79] E. Delp and O. Mitchel. Image compression using block truncation coding, 1979.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, pages 9–16, New York, NY, USA, 2001. ACM.

[FFGG$^+$10] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, and Marc Pollefeys. Building rome on a cloudless day. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 368–381, Berlin, Heidelberg, 2010. Springer-Verlag.

[FL12] Andreas Schilling Florian Liefers, Roman Parys. Analysis-by-synthesis texture reconstruction. In *Proceedings Second Joint 3DIM/3DPVT Conference: Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT2012)*, 2012.

[FP10] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, June 1994.

[FY94] James E. Fowler and Roni Yagel. Lossless compression of volume data, 1994.

[GFP10] D. Gallup, J.M. Frahm, and M. Pollefeys. A heightmap model for efficient 3d reconstruction from street-level video. In *3DPVT10*, pages xx–yy, 2010.

[GSC$^+$07] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. Multi-view stereo for community photo collections. In *Proceedings of the 11th International Conference on Computer Vision (ICCV 2007)*, pages 265–270, Rio de Janeiro, Brazil, 2007. IEEE.

[GWGS02] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straßer. Interactive rendering of large volume data sets. In *IEEE Visualization*, pages 53–60, 2002.

[HBH03] Markus Hadwiger, Christoph Berger, and Helwig Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 40–, Washington, DC, USA, 2003. IEEE Computer Society.

[HHB02] Arun Hampapur, Kiho Hyun, and Ruud M. Bolle. Comparison of sequence matching techniques for video copy detection. In Yeung et al. [YLL02], pages 194–201.

[HKLP09] Vu Hoang Hiep, Renaud Keriven, Patrick Labatut, and Jean-Philippe Pons. Towards high-resolution large-scale multi-view stereo. In *CVPR*, pages 1430–1437. IEEE, 2009.

[HZ04] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[HZL$^+$11] Christian Hane, Christopher Zach, Jongwoo Lim, Ananth Ranganathan, and Marc Pollefeys. Stereo depth map fusion for robot navigation. In *IROS*, pages 1618–1625. IEEE, 2011.

[IK00] Denis V. Ivanov and Yevgeniy Kuzmin. Color distribution - a new approach to texture compression. *Comput. Graph. Forum*, 19(3):283–290, 2000.

[IP99] Insung Ihm and Sanghun Park. Wavelet-based 3d compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18:3–15, 1999.

[JG82] Biing-Hwang Juang and Jr. Gray, A. Multiple stage vector quantization for speech coding. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '82.*, volume 7, pages 597 – 600, may 1982.

[Jol86] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[KC05] Young-tae Kim and Tat-Seng Chua. Retrieval of news video using video sequence matching. In *Proceedings of the 11th International Multimedia Modelling Conference*, MMM '05, pages 68–75, Washington, DC, USA, 2005. IEEE Computer Society.

[KI99] Z. Hong K. Iourcha, K. Nayak. System and method for fixed-rate block-based image compression with inferred pixels values. us patent 5,956,431, 1999.

[KKH01] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 255–262, Washington, DC, USA, 2001. IEEE Computer Society.

[Kni95] Guenter Knittel. High-speed volume rendering using redundant block compression. In *Proceedings of the 6th conference on Visualization '95*, VIS '95, pages 176–, Washington, DC, USA, 1995. IEEE Computer Society.

[KP02] Sang Hyun Kim and Rae-Hong Park. An efficient algorithm for video sequence matching using the modified hausdorff distance and the directed divergence. *IEEE Trans. Circuits Syst. Video Techn.*, 12(7):592–596, 2002.

[KP09] Guenter Knittel and Roman Parys. Pca-based seeding for improved vector quantization. In *International Conference on Imaging Theory and Applications, IMAGAPP09*, 2009.

[KR90] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.

[KS04] Yan Ke and Rahul Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR (2)*, pages 506–513, 2004.

[KSBB09] Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors. *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*. Curran Associates, Inc., 2009.

[KSKS96] G. Knittel, A. Schilling, A. Kugler, and W. Straßer. Hardware for superior texture performance. *COMPUTERS and GRAPHICS*, 20:475–481, 1996.

[LA09] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[LH07] Sylvain Lefebvre and Hugues Hoppe. Compressed random-access trees for spatially coherent data. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics, 2007.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[lMPH94] Kwan liu Ma, James S. Painter, and Charles D. Hansen. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14:59–68, 1994.

[Lou04] M.I.A. Lourakis. levmar: Levenberg-marquardt non-linear least squares algorithms in C/C++. [web page] `http://www.ics.forth.gr/~lourakis/levmar/`, Jul. 2004. [Accessed on 31 Jan. 2005.].

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[LPK07] P. Labatut, J.P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *ICCV07*, pages 1–8, 2007.

[MBS93] T.M. Martinetz, S.G. Berkovich, and K.J. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *Neural Networks, IEEE Transactions on*, 4(4):558 –569, jul 1993.

[MHS99] Michael Meissner, Ulrich Hoffmann, and Wolfgang Strasser. Enabling classification and shading for 3d texture mapping based volume rendering. In *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, VISUALIZATION '99, pages –, Washington, DC, USA, 1999. IEEE Computer Society.

[MLD$^+$06] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. 3d reconstruction of complex structures with bundle adjustment: an incremental approach. In *In ICRA 2006*, 2006.

[MT03] Kenneth Moreland and David Thompson. From cluster to wall with vtk. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, PVG '03, pages 5–, Washington, DC, USA, 2003. IEEE Computer Society.

[NH92] Paul Ning and Lambertus Hesselink. Vector quantization for volume rendering. In *Proceedings of the 1992 workshop on Volume visualization*, VVS '92, pages 69–74, New York, NY, USA, 1992. ACM.

[NS01] Ky Giang Nguyen and Dietmar Saupe. Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20:2001, 2001.

[NS06] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[NSD07] Kai Ni, Drew Steedly, and Frank Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *ICCV*, pages 1–8. IEEE, 2007.

[OT01] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. volume 42, pages 145–175, 2001.

[Per99] Anton Pereberin. Hierarchical approach for texture compression, 1999.

[PK08] Roman Parys and Guenter Knittel. On residual vector quantization for texture mapping. In *15th International Conference on Systems, Signals and Image Processing, IEEE IWSSIP*, 2008.

[PK09a] Roman Parys and Guenter Knittel. Giga-voxel rendering from compressed data on a display wall. In *Proceedings International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG '09)*, 2009.

[PK09b] Roman Parys and Guenter Knittel. Interactive large-scale volume rendering. In *5th High End Visualization Workshop*, 2009.

[PLS12] Roman Parys, Florian Liefers, and Andreas Schilling. Compact descriptor for video sequence matching in the context of large scale 3d reconstruction. In *The 8th International Conference on Multimedia and Network Information Systems (MISSI2012)*, 2012.

[PS12] Roman Parys and Andreas Schilling. Incremental large scale 3d reconstruction. In *Proceedings Second Joint 3DIM/3DPVT Conference: Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT2012)*, 2012.

[RAL03] Knut Magne Risvik, Yngve Aasheim, and Mathias Lidal. Multi-tier architecture for web search engines. *Web Congress, Latin American*, 0:132, 2003.

[RFP08] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *Proceedings of the 10th European Conference on Computer Vision: Part II*, ECCV '08, pages 500–513, Berlin, Heidelberg, 2008. Springer-Verlag.

[RGW$^+$03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the symposium on Data visualisation 2003*, VISSYM '03, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001.

[RL09] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In Bengio et al. [BSL$^+$09], pages 1509–1517.

[RMD04] Alex Reche, Ignacio Martin, and George Drettakis. Volumetric Reconstruction and Interactive Rendering of Trees from Photographs. *ACM Transactions on Graphics*, 23(3):720–727, 2004.

[Rod99] Flemming Friche Rodler. Wavelet based 3d compression with fast random access for very large volume data. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, PG '99, pages 108–, Washington, DC, USA, 1999. IEEE Computer Society.

[SAM04] Jacob Ström and Tomas Akenine-Möller. Packman: texture compression for mobile phones. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 66–, New York, NY, USA, 2004. ACM.

[SAM05] Jacob Ström and Tomas Akenine-Möller. ipackman: high-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '05, pages 63–70, New York, NY, USA, 2005. ACM.

[SBS07] Grant Schindler, Matthew Brown, and Richard Szeliski. City-scale location recognition. In *CVPR* [DBL07].

[SCD$^+$06] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard 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.

[SKZ99] Heung-Yeung Shum, Qifa Ke, and Z. Zhang. Efficient bundle adjustment with virtual key frames: A hierarchical approach to multi-frame structure from motion. In *Proc. of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'99)*, June 1999.

[SML$^+$03] Aleksander Stompel, Kwan-Liu Ma, Eric B. Lum, James Ahrens, and John Patchett. Slic: Scheduled linear image compositing for parallel volume rendering. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, PVG '03, pages 6–, Washington, DC, USA, 2003. IEEE Computer Society.

[SMW$^+$04] Magnus Strengert, Marcelo Magallón, Daniel Weiskopf, Stefan Guthe, and Thomas Ertl. Hierarchical visualization and compression of large volume datasets using gpu clusters. In *In Eurographics Symposium on Parallel Graphics and Visualization (EGPGV04) (2004*, pages 41–48, 2004.

[SSS] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Skeletal graphs for efficient structure from motion.

[SSS06] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, pages 835–846, New York, NY, USA, 2006. ACM Press.

[SSS08] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from Internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, November 2008.

[SW03] Jens Schneider and Ruediger Westermann. Compression domain volume rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[SZ03] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, October 2003.

[TFW08] Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *CVPR* [DBL08].

[TMHF00] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV ’99, pages 298–372, London, UK, UK, 2000. Springer-Verlag.

[Wik12a] Wikipedia. Bag of words model in computer vision — Wikipedia, the free encyclopedia, 2012. [Online; accessed 30-April-2012].

[Wik12b] Wikipedia. Dijkstra’s algorithm — Wikipedia, the free encyclopedia, 2012. [Online; accessed 30-April-2012].

[Wik12c] Wikipedia. Max-flow min-cut theorem — Wikipedia, the free encyclopedia, 2012. [Online; accessed 30-April-2012].

[Wik12d] Wikipedia. Surf — Wikipedia, the free encyclopedia, 2012. [Online; accessed 30-April-2012].

[WTF08] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In Koller et al. [KSBB09], pages 1753–1760.

[YC09] Mei-Chen Yeh and Kwang-Ting Cheng. Video copy detection by fast sequence matching. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR ’09, pages 45:1–45:7, New York, NY, USA, 2009. ACM.

[Yia93] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA ’93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[YLL02] Minerva M. Yeung, Chung-Sheng Li, and Rainer Lienhart, editors. *Storage and Retrieval for Media Databases 2002, 23 January 2002, San Jose, CA, USA*, volume 4676 of *SPIE Proceedings*. SPIE, 2002.

[ZM06] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), July 2006.

Curriculum Vitae

01. June 1979	born in Wrocław, Poland
1999 - 2001	Studies in Mathematics, Department of Mathematics and Computer Science, University of Wrocław, Poland
2000 - 2005	Studies in Computer Science, Department of Mathematics and Computer Science, University of Wrocław, Poland
2004	International Scholarschip at University of Saarland, Germany
2005 - 2006	Software Developer at BenQ/Siemens at Mobile R&D Center (Poland)
2006	Software Designer at Online Technology (Poland)
2006 - 2012	Research Associate at Department of Interactive Graphics Systems, Wilhelm Schickard Institute for Computer Science, Univeristy of Tuebingen, Germany
since 2012	Research Engineer at Leica Geosystems / Hexagon Technology Center, Heerbrugg, Switzerland