Methods for Real-time Visualization and Interaction with Landforms

Dissertation

zur Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Dipl.-Inf. Martin Schneider

aus Waldbröl

Bonn, April 2009

Universität Bonn,
Institut für Informatik II
Römerstraße 164, 53117 Bonn

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms Universität Bonn.

1. Referent: Prof. Dr. Reinhard Klein
2. Referent: Prof. Dr. Stefan Gumhold
3. Referent: Prof. Dr. Michael Clausen
4. Referent: Prof. Dr. Richard Dikau

Tag der Promotion: 3. September 2009
Erscheinungsjahr: 2010

Abstract

This thesis presents methods to enrich data modeling and analysis in the geoscience domain with a particular focus on geomorphological applications. First, a short overview of the relevant characteristics of the used remote sensing data and basics of its processing and visualization are provided. Then, two new methods for the visualization of vector-based maps on digital elevation models (DEMs) are presented. The first method uses a texture-based approach that generates a texture from the input maps at runtime taking into account the current viewpoint. In contrast to that, the second method utilizes the stencil buffer to create a mask in image space that is then used to render the map on top of the DEM. A particular challenge in this context is posed by the view-dependent level-of-detail representation of the terrain geometry.

After suitable visualization methods for vector-based maps have been investigated, two landform mapping tools for the interactive generation of such maps are presented. The user can carry out the mapping directly on the textured digital elevation model and thus benefit from the 3D visualization of the relief. Additionally, semi-automatic image segmentation techniques are applied in order to reduce the amount of user interaction required and thus make the mapping process more efficient and convenient. The challenge in the adaption of the methods lies in the transfer of the algorithms to the quadtree representation of the data and in the application of out-of-core and hierarchical methods to ensure interactive performance.

Although high-resolution remote sensing data are often available today, their effective resolution at steep slopes is rather low due to the oblique acquisition angle. For this reason, remote sensing data are suitable to only a limited extent for visualization as well as landform mapping purposes. To provide an easy way to supply additional imagery, an algorithm for registering uncalibrated photos to a textured digital elevation model is presented. A particular challenge in registering the images is posed by large variations in the photos concerning resolution, lighting conditions, seasonal changes, etc.

The registered photos can be used to increase the visual quality of the textured DEM, in particular at steep slopes. To this end, a method is presented that combines several georegistered photos to textures for the DEM. The difficulty in this compositing process is to create a consistent appearance and avoid visible seams between the photos. In addition to that, the photos also provide valuable means to improve landform mapping. To this end, an extension of the landform mapping methods is presented that allows the utilization of the registered photos during mapping. This way, a detailed and exact mapping becomes feasible even at steep slopes.

Zusammenfassung

In dieser Arbeit werden Methoden zur Verbesserung der Datenmodellierung und -analyse im Bereich der Geowissenschaften unter besonderer Berücksichtigung geomorphologischer Anwendungen vorgestellt. Zu Beginn wird ein kurzer Überblick über die relevanten Charakteristika der verwendeten Fernerkundungsdaten sowie Grundlagen zu deren Verarbeitung und Visualisierung gegeben.

Anschließend werden zwei neuartige Methoden zur Visualisierung vektorbasierter Karten auf digitalen Geländemodellen untersucht. Das erste Verfahren verfolgt einen texturbasierten Ansatz bei dem eine Textur anhand der darzustellenden Karte und unter Berücksichtigung der aktuellen Betrachterposition zur Laufzeit generiert wird. Im Gegensatz dazu benutzt das zweite Verfahren den Stencilbuffer um eine Maske im Bildraum zu erzeugen. Diese wird dann dazu benutzt, die Karte auf dem Geländemodell anzuzeigen. Eine besondere Herausforderung stellt dabei der blickpunktabhängige Detaillierungsgrad der Geländegeometrie dar.

Nach der Vorstellung geeigneter Visualisierungsmethoden für vektorbasierte Karten werden zwei interaktive Kartierungsverfahren zur Erstellung solcher Karten präsentiert. Die Kartierung erfolgt direkt auf dem texturierten Geländemodell, so dass der Benutzer von der 3D Visualisierung des Reliefs profitieren kann. Darüberhinaus werden durch die Anwendung halbautomatischer Bildsegmentierungsverfahren die notwendigen Benutzereingaben reduziert und der Kartierungsprozess damit effizienter und komfortabler. Die Herausforderung bei der Anpassung der Verfahren liegt in der Übertragung der Algorithmen auf die Quadtree-Repräsentation der Daten und der Verwendung von hierarchischen und Out-of-Core Methoden, um Interaktivität zu gewährleisten.

Obwohl hochaufgelöste Fernerkundungsdaten heutzutage immer häufiger verfügbar sind, ist ihre tatsächliche Auflösung in Steilhangbereichen aufgrund des schiefen Aufnahmewinkels deutlich niedriger. Aus diesem Grund sind solche Daten sowohl zum Zwecke der Visualisierung als auch zur Kartierung von Steilhangbereichen prinzipiell nur bedingt geeignet. Um eine einfache Möglichkeit zur Integration zusätzlicher Bildinformation zu schaffen, wird deshalb ein Algorithmus vorgestellt mit dem unkalibrierte Photos gegen das texturierte Geländemodell registriert werden können. Eine besondere Herausforderung bei der Registrierung stellen dabei die unterschiedlichen Auflösungen, Lichtverhältnisse, Jahreszeiten, etc. der Photos dar.

Die registrierten Bilder können dann zur Verbesserung der visuellen Qualität des texturierten Geländemodells, insbesondere in steilen Hangbereichen, verwendet werden. Dazu wird ein Verfahren vorgestellt, das es erlaubt, registrierte Photos zu Texturen für das Geländemodell zu kombinieren. Die Schwierigkeit in diesem Compositing-Prozess liegt in Erzeugung eines konsistenten Erscheinungsbildes und der Vermeidung sichtbarer Grenzen zwischen den Bildern. Neben der Verbesserung des visuellen Erscheinungsbildes können die Photos aber auch für die Kartierung verwendet werden. Dazu werden die beiden Kartierungsverfahren so erweitert, dass die Kartierung auf Grundlage der registrierten Photos ausgeführt werden kann. Auf diese Weise wird eine detailierte und exakte Kartierung sogar in steilen Hangbereichen möglich.

Contents

CHAPTER 1

Preface

1.1 Motivation

This thesis was written within the *Research Training Group (RTG) 437 - "Landform - a structured and variable boundary layer"*[1] a multidisciplinary graduation program funded by the German Research Foundation (DFG)[2]. In this program landform, as the boundary surface between different components of the earth system, is investigated within a range of disciplines ranging from geoscience (geomorphology, hydrology, climatology, geodynamics, meteorology, pedology) to biology, mathematics, computer science and remote sensing. In all these disciplines the use of remote sensing data for analysis and interpretation has become increasingly important during the last decades. For Turtmann valley, the alpine cluster of the RTG 437, a High Resolution Stereo Camera (HRSC) dataset was acquired that provides the basis for research. Its applications within the RTG are various including geomorphological mapping, geomorphometrical analysis, mapping of surficial grain-size distribution, rock glacier kinematics analysis, vegetation monitoring and 3D visualization.

One of the most basic tasks in geomorphology is the generation and interpretation of geomorphological maps. With the increasing availability and quality of remote sensing data, today the creation of geomorphological maps is typically carried out directly on such data on screen. While modern geoin-

[1]http://www.giub.uni-bonn.de/grk/
[2]http://www.dfg.de

formation systems (GIS) facilitate the compilation, production and distribution of maps, most of the cartographic features are limited to a 2D representation of the input data. Although different kinds of relief shading are often used to compensate for this, a fixed 2D aerial perspective heavily restricts the perception of landforms. Furthermore, considerable amount of manual interaction is usually involved in landform mapping. The user has to define the object boundary by extensively clicking and dragging with the mouse which is tedious and time-consuming. Even worse, not only the presentation and interaction with the data, but also the data itself are often an issue. Since remote sensing data is acquired from above, steep slopes are only sparsely sampled while overhangs are not captured at all. In particular in high alpine environments where the amount of steep slopes is very high, this results not only in low visual quality but also prohibits a detailed mapping in such areas.

Until now the main contribution of computer graphics to the geoscience domain has been the generation of digital elevation models (DEMs) along with their efficient visualization. Although an enhanced visualization of the data improves the perception of landforms, so far 3D terrain visualization software has typically been limited to simple data exploration. Considering this background and motivated by the numerous and manifold applications in the geoscience domain that become viable - in some cases even for the first time - using computer graphics methodologies, this thesis presents methods that address the aforementioned issues.

The methods presented in this thesis extend an existing terrain visualization system allowing an efficient visualization as well as an interactive generation of geomorphological maps directly on the textured DEM. To account for the resolution issues at steep slopes, photos can be registered to the textured DEM. The registered photos can be combined to textures for the DEM in order to increase visual quality, in particular at steep slopes. Apart from improving visual quality, the photos can also be utilized for mapping purposes allowing a detailed mapping even at steep slopes. Figure 1.1 provides an overview of the different components presented in this thesis.

1.2 Main Contributions

Several aspects of the work presented in this thesis have already been published at different conferences and journals [Schneider *et al.* 2005][Schneider & Klein 2006][Schneider & Otto 2006][Schneider & Klein 2007b][Schneider & Klein 2007a][Otto *et al.* 2007][Schneider & Klein 2008]. The content of this thesis is based on these publications, explaining the proposed methods in more detail and providing additional background knowledge. This is com-

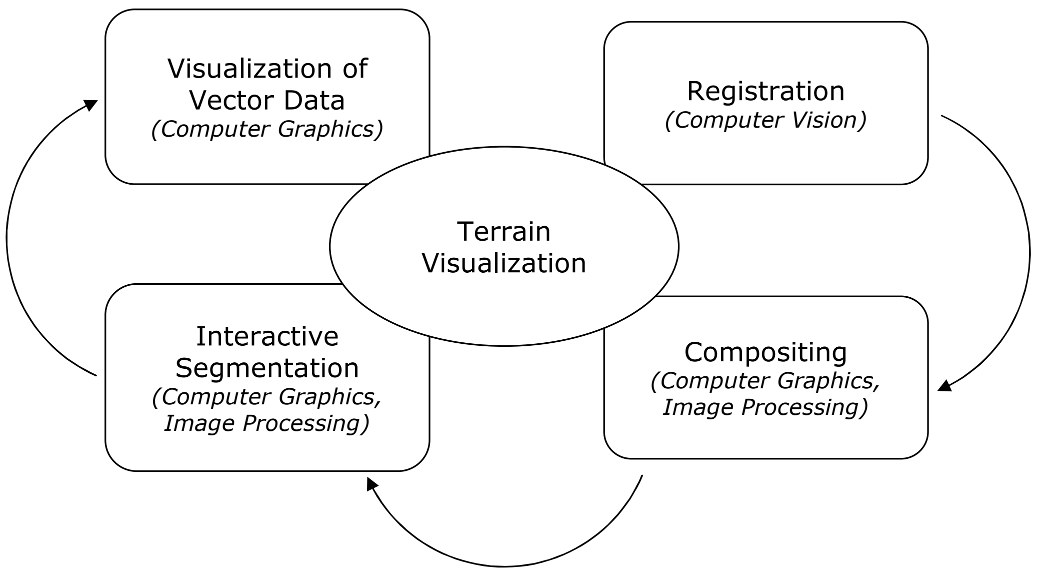


Figure 1.1: Overview. Main aspects covered in this thesis.

pleted with improvements and further results of the presented methods and algorithms. The main contributions of this thesis can be summarized as follows

- efficient and exact visualization of vector data on textured DEMs (chapter 3)
- robust registration of photos to textured DEMs (chapter 4)
- high-quality compositing of registered photos on DEMs (chapter 5)
- interactive landform mapping on large textured DEMs (chapter 6).

The methods and algorithms presented in this thesis are motivated and described with their applicability in the context of geomorphology offering a variety of applications and at the same time posing some specific demands. However, most of the contributions of this thesis are applicable and relevant to other disciplines as well.

1.3 Thesis Overview

The remainder of this thesis is organized as follows:

In chapter 2 background information on data formats used for geospatial data in general are supplied, followed by a detailed description of the specific characteristics of the HRSC dataset of Turtmann valley that is used throughout this thesis. Then, methods to efficiently handle such huge datasets are presented and the used terrain rendering engine is described.

Chapter 3 covers the *visualization of vector data* on digital elevation models. After motivating the topic in section 3.1, related work is reviewed in section 3.2. Then, two novel methods for the visualization of vector data are presented in sections 3.3 and 3.4. Finally, results are shown and discussed in section 3.5.

Chapter 4 deals with the *registration* of photos to textured digital elevation models. First, the topic is motivated in section 4.1 and related work is reviewed in section 4.2. Then, a robust algorithm for registering a set of uncalibrated photos to a given textured digital elevation model is described in section 4.3. Section 4.4 then concludes this chapter of the thesis by presenting and discussing the obtained results.

Chapter 5 covers the *compositing* of photos on textured digital elevation models. After motivating the topic in section 5.1, related work is reviewed in section 5.2. Then a compositing algorithm that combines registered photos on the terrain surface is presented in section 5.3. Finally, results are shown and discussed in section 5.4.

Chapter 6 deals with *interactive landform mapping* on digital elevation models. First, the topic is motivated in section 6.1 and related work is reviewed in section 6.2. Then, two novel interactive landform mapping methods are presented in section 6.3 and 6.4. After that, a hierarchical approach to accelerate both methods is given in section 6.5. In section 6.6 then an extension to the methods is presented that allows detailed landform mapping at steep slopes. Finally, section 6.7 concludes this chapter by presenting segmentation results and a comparison to standard mapping tools.

The thesis is concluded in chapter 7. Appendix A provides basics on registration complementing chapter 4, followed by basics on mesh parameterization complementing chapter 5. Finally, appendix C presents image segmentation basics complementing chapter 6.

CHAPTER 2

Background

Since ancient times people have been occupied with representing the earth's surface. While paintings are one of the oldest representations offering some general information, they lack accuracy and thus are inappropriate for scientific or engineering applications. A more effective representation are *maps* which are still widely used today. Modern maps employ a well-designed symbol system and a well-established mathematical basis. They are scientific generalizations and abstractions of the landscape, most importantly they are a 2D representation of 3D reality. In maps the third dimension is usually conveyed by means of special cartographic techniques, such as contour lines, relief shading or special signatures. However, these are rather abstract constructions that require training and imagination from the user. Especially in mountainous regions, characterized by a high proportion of steep slopes, the traditional map representation reaches its limits.

To overcome these shortcomings, *physical relief models* made of rubber, plastic, clay or sand can be used. In contrast to a map, they are real 3D representations of the earth's surface and allow the observer to choose his position freely. Therefore, they convey an immediate and natural impression of a landscape and are much more intuitive than 2D maps for most people. However, the building of physical relief models is costly and time-consuming.

With the invention of photography, photos and later *aerial images* have been used to represent the terrain. Since the 1970s *satellite images* have been used to complement aerial imagery, although the resolution of satellite images is still not comparable. Elevation data of the earth surface, on the other hand, can either be acquired by digitizing existing maps, derived from overlapping aerial imagery using photogrammetric methods, or directly measured using airborne laser scanning (LIDAR) or satellite based radar (InSAR). With the advance of computing technology and visualization algorithms, it is nowadays possible to create realistic looking, virtual 3D models of the earth's surface based on aerial imagery and corresponding elevation data.

In the remainder of this chapter, first basics on digital terrain data in general and characteristics of the Turtmann valley dataset in particular are presented. Then, methods to efficiently handle large mesh and texture data are reviewed with a particular focus on terrain visualization. Finally, the terrain rendering engine is introduced that is used as the basis of the algorithms presented in this thesis.

2.1 Digital Terrain Data

Along with the progression of computing technology, mathematics and computer graphics, various digital terrain representations have been developed. Even data that is not acquired in digital form initially is often converted into discrete data using digitizing techniques. The advantage of having the data available digitally is that it can be efficiently processed, stored and transferred using computers.

There are two main categories for the digital representation of geospatial data, namely vector data and raster data. The *vector data* model represents space as a series of discrete entity-defined point, line or polygon units, which are geographically referenced by Cartesian coordinates. Vector data is best suited to store discrete, well-defined data that can clearly be delimited. Location of samples (points), streets (lines) and lakes (areas) are examples of adequate candidates for a representation as vector data. *Raster data*, by contrast, consists of a matrix of cells (pixels) organized in an ordered grid of rows and columns. Each cell contains a value representing information, such as elevation. Cell values can either be positive or negative, integer or floating point but can also contain "no data" values to indicate the absence of information. Data stored in a raster format typically represents real-world phenomena, such as thematic and continuous data, or photos.

The most common terrain features acquired are spectral values and elevation. Spectral images of the earth's surface are naturally represented as raster data in the form of standard image file formats. On the other hand, *digital elevation models* (DEMs), also known widely as *digital terrain models* (DTMs), that model the variation of surface elevation over an area, can either be modeled by raster data or by triangle meshes. The preference for one or the other depends on the application. Raster data are the most common form of discretized elevation data because useful information about landform, such as slope and curvature, can easily be derived from it. However, the raster data representation includes large amounts of redundancy in uniform terrain areas due to its inability to adapt to areas of varying relief complexity. The use of triangle meshes for digital elevation modeling avoids the redundancies of raster data. Although triangle meshes provide efficient data storage of elevation data, they introduce a triangular discretization that may hinder some kinds of spatial data analysis.

In order to analyze several datasets jointly, they have to be in correct geographic relation to each other. Therefore, spatial data is usually *georeferenced*, i.e., geometrically registered to a generally accepted and properly defined coordinate system. Apart from local studies, the common frame of reference is provided by one of the limited number of geodetic coordinate systems. The most usual coordinate system is that of plane Cartesian coordinates, oriented north/south (latitude) and east/west (longitude). Since the earth is not a true sphere but flattened at the poles, geodesists have devised several ellipsoids for mapping the true curved surface of the earth to the plane. There are three main ways for projecting locations from an ellipsoid onto a planar surface, namely cylindrical, azimuthal and conical projections, where the best projection to use depends on the location. The most widely used, general projection, and a standard for topographic mapping and digital data exchange, is the Universal Transverse Mercator (UTM) [Yang *et al.* 2000]. Currently, the WGS84 ellipsoid [Lohmar 1988] is used as the underlying model of the earth in the UTM coordinate system.

2.1.1 HRSC Dataset of Turtmann Valley

In this section, Turtmann valley, the alpine cluster of RTG 437, and its corresponding HRSC dataset are briefly introduced. For further information on the HRSC dataset and its applications in physical geography the reader is referred to [Otto *et al.* 2007].

Turtmann Valley is an alpine catchment located in the southern mountain range of the Valais Alps between the Matter Valley and the Anniviers Valley in Switzerland. The Turtmann valley stream is a southern tributary of the Rhone River and drains a catchment of approximately 110 km^2 (139 km^2 real surface) at altitudes between 620 m and 4200 m. The valley is around 20 km long and up to 7 km wide and the main glacial trough is oriented from south to north. In addition to small glaciers in some of the hanging valleys, the Turtmann and Brunegg Glaciers at the valley head cover approximately 14 % of the valley surface. The hanging valleys are characterized by more than 80

	HRSC-A	HRSC-AX150	HRSC-AX047
Focal length [mm]	175	151	47
Number of CCD lines	9	9	5
Number of sensors per line	5184	12000	12000
Sensor size [μm]	7	6.5	6.5
Radiometric resolution [bit]	8	12	12
Multispectral viewing angle [°]	15.9 (R) 3.3 (G) -3.3 (B) -15.9 (nIR)		
Stereo viewing angle [°]	$\pm$18.9 $\pm$12.8 0	$\pm$20.5 $\pm$12	$\pm$14.4
Field of view [°]	$\pm$11.8	$\pm$29.1	$\pm$79.4
Flight altitude for 20 cm geometrical resolution	5000	4700	1500
Spectral resolution [nm]			
Blue	395-485	450-510	-
Green	485-575	530-570	475-575
Red	730-770	635-685	570-680
Near infrared	925-1015	770-810	-
Nadir (panchromatic)	585-765	520-760	515-750
Stereo (panchromatic)	585-765	520-760	515-750
Photometry (panchromatic)	585-765	520-760	-
Maximum line frequency per band [lines/s]	450	1640	1640
Platform	stabilized Zeiss T-AS-Platform		
Data recording	SONY high speed data recorder		
Weights: camera [kg]	~32	~70	~70
adapter [kg]	~40	~40	~40

Table 2.1: Technical Data on the HRSC Sensors.

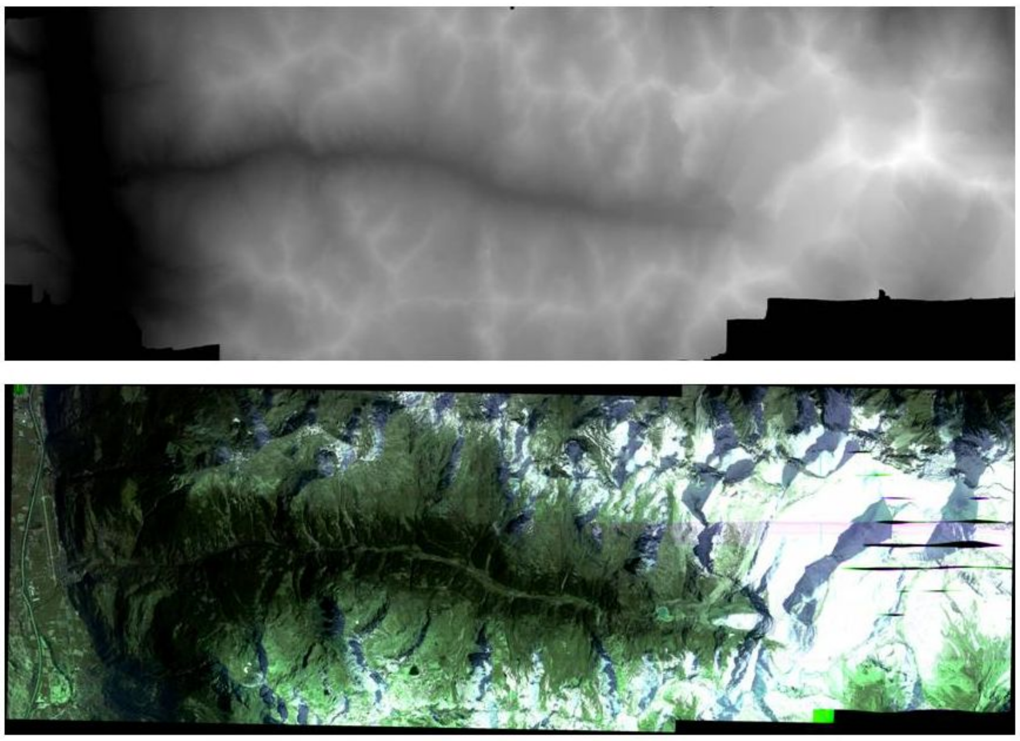


Figure 2.1: Turtmann Valley Dataset. Digital elevation model (top) and corresponding aerial imagery (bottom).

recent and relict rock glaciers. The inner Alpine location of the Turtmann valley is characterized by continental climatic conditions.

In September 2001 a flight campaign covering the entire Turtmann valley was carried out using the HRSC-A system. The *High Resolution Stereo Camera* (HRSC) was originally developed for the mission “Mars Express” by the German Aerospace Center (DLR)[1]. First airborne experiments on earth with HRSC-A (A - for airborne) showed good results in mapping and photogrammetry. For this reason two additional airborne cameras (HRSC-AX 150, HRSC-AXW 47) were developed in 2000 [Neukum 2001]. The HRSC-A sensor (see Table 2.1) is a multispectral stereo scanner containing nine bands: a blue, green and red band (tending to near infrared), a near infrared band, and five panchromatic bands covering the green and red spectrum. It is a pushbroom scanner consisting of CCD sensors in nine lines. The sensors scan line by line (nine lines at a time) along the flight path. Each CCD line scans another band, each with another viewing angle. The geometrical resolution

[1]http://www.dlr.de/

depends on the flying altitude and starts at 10 cm upwards.

For Turtmann valley 13 overlapping parallel tracks from a mean flight altitude of 6000 m (3000 to 4000 m over ground) were acquired. Limited by the lateral field of view of the sensors, several parallel tracks had to be recorded and mosaiced for each color and stereo band separately during on-ground processing. After data processing at DLR, including combined determination of position and altitude, geometric correction of image data, image matching, DTM generation, orthoimage generation and mosaicing [Hauber *et al.* 2000][Scholten *et al.* 2002], a several gigabyte large dataset consisting of multispectral stereo data as well as a digital terrain model (including vegetation) was created (see Figure 2.1). Image and DTM data are referenced to the standard Swiss map projection CH1903. The image data has a radiometric resolution of 8 bits, while the terrain model is coded in 16 bits resolving a height difference of 10 cm. The spatial resolution is 50 cm and 1 m for image data and DTM, respectively. Multispectral data tend to be slightly blurred in the flight direction due to their smaller filter bandwidth compared to the more panchromatic filters of the stereo bands resulting in longer exposure times. Nevertheless, sharpening can be achieved by using HSI transformation with the very sharp Nadir band as the intensity component.

2.2 Data Handling

Despite the capabilities of today's hardware, real-time visualization of large terrain datasets is not feasible by brute force. Sophisticated data structures and algorithms are indispensable to achieve real-time performance. In addition to that, most terrain datasets do not even fit into main memory and hence out-of-core strategies are required. Thus, to enable efficient rendering of such models, it is essential to reduce the amount of data that is used for rendering and kept in main memory as much as possible. *Compression* techniques can help to reduce the size of the data significantly in advance. During rendering *culling* techniques are typically applied in order to determine as early as possible in the rendering pipeline the subset of the data that actually contributes to the final image and to restrict further processing to it. *Level-of-detail* (LOD) techniques can then be used to adjust the complexity of the remaining relevant data, for example, based on its projected size in screen space. By using LOD techniques it is possible to reduce the complexity of distant objects without sacrificing quality. This leads to drastically increased rendering speed while at the same time aliasing artifacts are reduced.

In the remainder of this section the aforementioned techniques for handling large meshes and textures are briefly overviewed. First, compression algorithms for meshes and textures are reviewed, followed by culling techniques. Then, methods to generate multiresolution representations are presented and finally, data structures that enable an efficient management of such LOD representations are described.

2.2.1 Compression

Compression techniques directly tackle the memory and bandwidth problem and are therefore vital when dealing with large datasets. There are a variety of different compression techniques for meshes as well as textures.

Geometry Compression

In geometry compression vertex positions as well as connectivity can both be compressed. Typically, a vertex consists of three real-valued components usually represented in a 32 bit floating point format. Such a representation can distinguish between 2^{32} different values, which is often more than is needed for a given application and hence the same amount of information may be represented with fewer bits.

Quantization is the process of approximating a continuous range of values using a relatively small set of discrete values. The simplest form of quantization is uniform quantization, where the considered domain is discretized into a uniformly-spaced multidimensional grid structure. Another popular approach is vector quantization where the quantized values are chosen such that the overall quantization error is minimized. After the selection of a set of quantized values the original vertices are snapped to their closest quantized value and are then coded with accordingly fewer bits.

Another approach for the compression of vertex positions is *predictive coding*, which is based on the observation that there exists some correlation between the position of a vertex and that of its neighbors. Using specific prediction rules it suffices to code just the prediction error, namely the difference between the actual vertex position and its predicted value. A popular prediction rule is the parallelogram rule [Touma & Gotsman 1998] that predicts a vertex as the fourth vertex of a parallelogram based on three neighboring vertices. Significant savings result if the entropy of the prediction errors is much less than that of the original vertex positions.

In addition to vertex positions, the connectivity of a triangle mesh can be compressed as well. One popular class of connectivity coding methods grows a region over the mesh and incrementally encodes mesh elements and their incidence relation to the growing region. These methods can be categorized as face-based [Gumhold & Straßer 1998][Rossignac 1999], edge-based [Isenburg & Snoeyink 2000] and vertex-based [Touma & Gotsman 1998] coding schemes.

Texture Compression

High resolution textures are often even larger than the corresponding geometry, in particular in terrain visualization, and thus texture compression is at least as important as geometry compression. Since computer graphics systems are typically highly pipelined, it is desirable to keep the number of indirections during texture lookups low. What is more, texture caches are often used in graphics hardware to speed up texture accesses. Therefore, texture compression methods should work well with existing caches, which makes it important to preserve the locality of texture accesses. Overall, a decompression algorithm should be simple, fast, and easy to implement in hardware.

One solution that directly attacks the memory and bandwidth problem is *fixed-rate texture compression* [Beers *et al.* 1996]. By letting hardware decode compressed textures on-the-fly, textures do not only require less memory bandwidth when accessed but also consume less texture memory. S3 developed a scheme called S3TC [Inc. 1998], which was chosen as a standard in DirectX, called DXTC, and hence is available on all consumer graphics hardware. It has the advantage of creating a compressed image that is fixed in size, has independently encoded localized pieces and is simple and fast to decode. The DXT compression formats are made up of DXT1 through DXT5 and use a lossy compression that can reduce an image's size by a ratio of 4:1, 6:1 or 8:1 depending on the handling of the alpha channel.

Alternatively, one of the various image compression method used in image file formats, such as JPEG and PNG [Miano 1999], can be used. They offer the advantage of higher compression ratios but are not implemented on common graphics hardware. Hence, they reduce bandwidth requirements but have to be decoded on the CPU before being loaded into graphics memory and thus occupy their original uncompressed size there.

2.2.2 Culling

Culling techniques aim at reducing the amount of data sent through the rendering pipeline by detecting and removing parts of the scene that are not visible to the viewer. The three main culling techniques are backface culling, view frustum culling and occlusion culling (see Figure 2.2). *Backface culling* removes all opaque triangles in the scene whose normal points away from the observer. In this way, backface culling can decreases the load on the rasterizer significantly. It can be made even more efficient by testing not every triangle separately but whole sets of triangles. Such techniques are called clustered backface culling and often use normal cones [Shirman & Abi-Ezzi 1993].

In *view frustum culling* geometry located outside the view frustum is determined and not sent through the pipeline at all. To this end, the scene is typically organized in a spatial data structure, which is then used to perform the view frustum culling hierarchically. While view frustum culling only removes geometry outside the frustum, *occlusion culling* techniques try to identify geometry that is inside the frustum but occluded and therefore does not contribute to the final image. If scenes with high depth complexity are rendered, removing occluded geometry can significantly increase the rendering performance. On the other hand, if the depth complexity is moderate or low, the costs for occlusion culling often outweigh its benefits.

A characteristic of hilly terrain models and urban scenes is a high depth complexity in the horizontal direction. Therefore, whenever the viewer is close to the ground and looks horizontally, the depth complexity is considerably higher than for other views. As such views tend to be very common in the aforementioned applications, they can greatly benefit from efficient occlusion culling. Since exact solutions of the occlusion culling problem are computationally expensive, they are of limited use for real-time rendering. Therefore, the accuracy of the occlusion culling is typically traded with its computational complexity. The main operations performed in occlusion culling are visibility tests that are performed many times per frame and thus have to be efficient. For this reason, the visibility tests are typically performed with simple approximations of the objects in the scene, such as bounding boxes, that are conservative over-estimations of the extents of the considered objects.

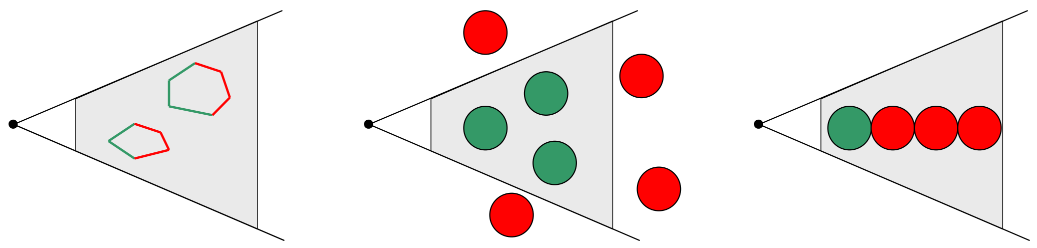


Figure 2.2: Culling. Backface culling (left), view frustum culling (middle) and occlusion culling (right). The green primitives and objects are further processed while the red ones are removed from further processing in the rendering pipeline.

2.2.3 LOD Generation

Even after the application of compression and culling techniques, the remaining amount of data is often still too large to be rendered in real-time. Using LOD techniques, the amount of data that need to be rendered can be further reduced significantly. While LOD representations for triangle meshes are typically obtained by *simplification*, simplified versions of textures are usually created by *downsampling*.

Geometry Simplification

Numerous approaches for the simplification of meshes have been proposed over the years, which can roughly be divided into iterative and remeshing approaches. *Iterative mesh simplification* schemes apply a series of local simplification operations to reduce the complexity of the given mesh, i.e., they reduce the number of vertices, edges and faces. Among the numerous simplification operations that have been developed the most popular are vertex clustering [Rossignac & Borrel 1993], vertex removal [Schroeder *et al.* 1992] and edge collapse [Hoppe *et al.* 1993] (see Figure 2.3). The advantage of these methods are that they are easy to implement and produce good results in short time. Therefore, iterative methods have become the most widely used technique for large models, although optimality of the simplified model can not be guaranteed.

In contrast to iterative simplification techniques, *remeshing* approaches

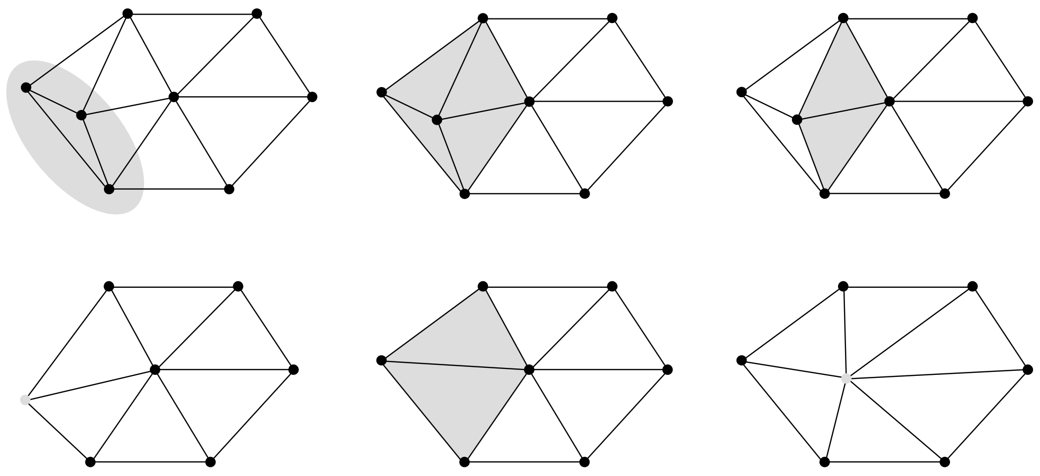


Figure 2.3: Simplification Operations. Vertex clustering (left), vertex removal (middle) and edge collapse (right).

handle the complete input mesh in a single step. They generate a new mesh that approximates the original by adaptively sampling the mesh surface [Turk 1992][Alliez *et al.* 2005] placing more points in regions of high curvature and less points in almost planar regions. The advantage of remeshing techniques is their global handling of input models, which enables them to create meshes that present an optimal balance between approximation error and mesh complexity. Unfortunately, the required processing times are typically much longer compared to iterative simplification approaches.

Texture Downsampling

In case of textures, different LODs are typically created by successively filtering an input image. Downsampling an image in the spatial domain consists of two steps: First, the image is filtered by an antialiasing low pass filter. Then, the filtered result is subsampled by a desired factor in each dimension. In practice a texture often has to be downsampled to a quarter of its original area. A naive but often applied approach to achieve this is to compute each new texel as the average of its four corresponding texels in the original texture. However, the filter used in such a case is a box filter, which is one of the worst filters possible and might result in poor quality. Instead a Gaussian, Lanczos, Kaiser or similar filter should be used.

Error Metrics

Error metrics control the application of simplification operations. In the case of iterative mesh simplification, they define the order of simplification operations. Klein et al. [Klein & Liebich 1996] recommended to compute the simplification error based on the *Hausdorff distance*. The Hausdorff distance of two sets of points $\mathcal{P}_1$ and $\mathcal{P}_2$ is defined as

$$d_H(\mathcal{P}_1, \mathcal{P}_2) = \max_{p_1 \in \mathcal{P}_1} \left\{ \min_{p_2 \in \mathcal{P}_2} \{d(p_1, p_2)\} \right\},$$

where d is typically assumed to be a Euclidean distance function for point-to-point distances. Since d_H is not symmetric, usually the symmetric version

$$d_{H_s} = \max \{d_H(\mathcal{P}_1, \mathcal{P}_2), d_H(\mathcal{P}_2, \mathcal{P}_1)\}$$

is used. Unfortunately, efficient computation of d_H and especially d_{H_s} is difficult. A method for the efficient computation of d_{H_s} that focuses on regions of potentially highest distance was described by Guthe et al. [Guthe *et al.* 2005a]. Other work proposed more computationally efficient metrics approximating d_{H_s}. Garland and Heckbert [Garland & Heckbert 1997] introduced *error quadrics* as a simplification metric. Compared to error metrics based on the Hausdorff distance, error quadrics can be computed much faster. However, a significant problem of this approach is the overestimation of the actual simplification error when summing error quadrics, which prohibits aggressive simplification and thus performance-optimal LOD models.

Since many attributes of an object contribute to its visual appearance, error metrics need to take more criteria into account than just geometric deviation. While early error metrics exclusively focused on geometric accuracy, later work [Garland & Heckbert 1998][Schilling & Klein 1998][Cohen *et al.* 1998] also considered additional surface attributes, such as color, texture coordinates or normals.

2.2.4 LOD Data Structures

LOD data structures enable efficient access to LOD representations and thus allow efficient changes between LODs. Such data structures are extremely important not only for runtime selection of LODs but additionally enable efficient offline generation of LODs. LOD data structures can be divided into *static*, *continuous* and *hierarchical* approaches.

Geometry LOD Data Structures

The concepts of multiresolution meshes have been extensively studied for general 3D triangle meshes and have been surveyed in [Cignoni *et al.* 1998][Garland 1999][Heckbert & Garland 1997][Luebke *et al.* 2002][Luebke 2001], and more recently in [Floriani *et al.* 2004].

The most simple LOD representation are sets of *static LODs* consisting of sequences of simplified versions of triangle meshes, where the different versions have been simplified up to some target simplification errors. Special care has to be taken to avoid sudden changes of the object's appearance when switching the displayed LOD known as popping artifacts. Static LODs are commonly derived from iterative mesh simplification algorithms, remeshing techniques, and approaches creating image-based representations. Mixing various LOD representations is easily possible since separate LODs are handled independently of each other. While static LODs are very simple to handle and quite efficient in terms of storage, a major disadvantage is that the resolution is fixed throughout the whole displayed model. As a result, if some parts of the model need to be displayed at a high resolution, the whole object has to be rendered at this high resolution. In situations where the extent of the displayed object is relatively large, as for example in terrain visualization, this increases the number of displayed triangles significantly.

To account for this problem, the idea of *continuous LODs* was introduced. Such LOD representations enable much finer control over the resolution of displayed objects, effectively allowing varying resolutions for separate parts of a simplified mesh. A widely known approach to continuous LODs are progressive meshes [Hoppe 1996], which are derived by iteratively simplifying an input mesh and storing the sequence of simplification operations and the simplified mesh in an appropriate data structure [Puppo 1998]. Progressive meshes have been adapted to terrain rendering in [Hoppe 1998]. Even though they allow fine-scale changes to the mesh to be made from one frame to the next, these changes, if geometrically large enough, may lead to popping artifacts. Geomorphing is a common approach to counter such visually disturbing phenomena by interpolating the geometric transitions between different levels of detail smoothly over time. However, downside of morphing is that vertices may have to be introduced earlier than otherwise necessary to allow a continuous transition while still satisfying an error tolerance.

While the use of static LODs is relatively inflexible compared to continuous LODs, they require much less runtime overhead. At each frame, only the most suitable LOD version has to be chosen. In contrast, continuous LOD approaches need to readjust the model complexity each frame, which implies validity checks for all active vertices. For large models this overhead easily outweighs the advantages of rendering less triangles. Therefore, Erikson et al. [Erikson *et al.* 2001] proposed *Hierarchical LODs* (HLODs). To create them, the scene is first subdivided into separate parts in a hierarchical manner. Then, the partitioned geometry parts are simplified separately, typically using iterative simplification methods. The HLOD concept has three major advantages: First, they allow for dynamic LOD by choosing separate resolutions for different parts of the geometry. Second, compared to progressive meshes, selection of LOD levels becomes much more efficient since resolution is selected for chunks of geometry instead of on a per-vertex basis. Third, hierarchical subdivision of the input model naturally lends itself to out-of-core handling. However, if parts of the geometry are simplified separately, which is particularly relevant for out-of-core models, cracks may be introduced into the simplified model since simplifications in neighboring parts are not taken into account. To avoid cracks, borders between neighboring parts can either be constrained during simplification, which might reduce simplification rates, or the individual pieces have to be merged afterwards using stitching operations [Borodin *et al.* 2003].

To partition a scene into separate parts, *spatial data structures* are typically used that arrange data in some n-dimensional space. Spatial data structures are often organized hierarchically, i.e., their structure is nested and of recursive nature. Examples of spatial data structures are bounding

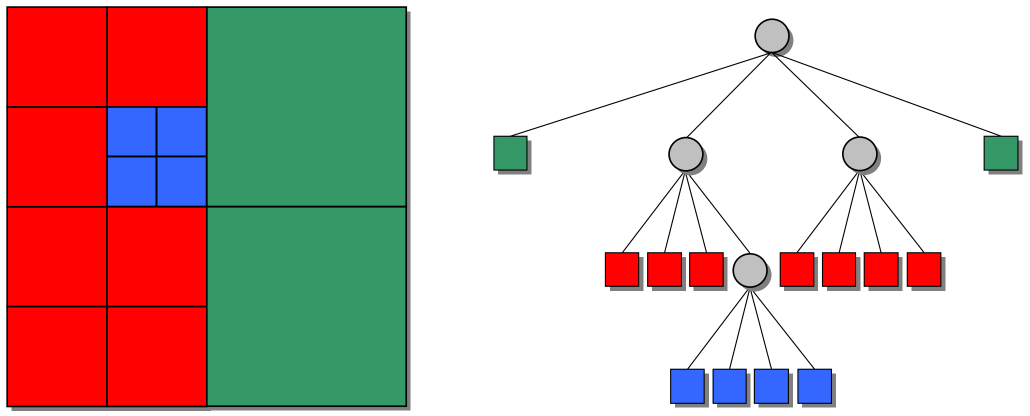


Figure 2.4: Quadtree. An example quadtree (left) and the corresponding graph representation (right).

volume hierarchies, BSP trees, quadtrees and octrees. In the field of terrain visualization *quadtrees* are often used. A quadtree is a tree data structure in which each internal node has up to four children and every node in the tree corresponds to a square in the associated spatial domain. It is constructed by enclosing the entire scene with a minimal square and recursively subdividing this area into four quadrants until some stopping criterion is fulfilled. Such a criterion can include that a maximum number of recursion levels has reached, or that there is fewer than a threshold number of primitives in a cell, otherwise subdivision is continued. An overview of spatial data structures for computer graphics and its application can be found in [Zachmann & Langetepe 2003].

Texture LOD Data Structures

As for meshes, several data structures have been developed to handle LOD representations of textures (see Figure 2.5). *Mipmaps* [Williams 1983] are a pyramidal texture representation comprising several images derived by successively downsampling the original texture. Each image in the sequence is exactly of half the resolution as the previous. A great advantage of mipmaps is that they are supported by consumer graphics hardware.

Tanner et al. [Tanner *et al.* 1998] presented *clipmaps*, directly addressing the texturing problem in terrain rendering. They observed that when the terrain texture is represented as a mipmap, only a small part of the finer levels of the mipmap is actually used for any particular view. Therefore, instead of holding the entire texture pyramid in texture memory, the finer levels are clipped to a fixed size around a center point. Thus, the finest resolution texture is used for the smaller part close to the viewer while coarser textures are used for correspondingly larger parts further away. However, a difficulty in the clipmap approach is the selection of a good center point so that the clipmap does not contain large areas outside the view frustum. Even if the clip center could be optimally chosen, the quadratic clip map would still not optimally fit the triangular view frustum. Unfortunately, clipmaps do not allow to take advantage of occlusion culling, as it is not possible to reduce the amount of loaded texture. In addition to that, clipmaps are not supported on consumer graphics hardware.

The idea of the *mp-grid* [Hüttner 1998] is to split the texture into a regular grid of standard mipmaps. In this approach only textures for visible grid cells at the necessary LODs, which are precomputed and stored, have to be loaded. However, a problem is that the regular grid cannot be adapted to the camera and thus grid cells can become arbitrarily small on the screen. Besides the problem that it is impractical to load a very large number of mipmaps in such a case, filtering of the textures when the footprint of a pixel covers several grid cells is not possible, as the lowest available level is reached when one texel corresponds to one grid cell.

The *texture tile* representation [Klein & Schilling 2001] organizes the texture into a pyramid. Each pyramid level is subdivided into equally sized blocks or tiles. The tile size is constant on all levels of the texture pyramid. The model corresponds to a quadtree subdivision where texture tiles from coarse levels correspond to large areas, those from finer levels to small areas. In contrast to the clipmap approach, only the needed textures are loaded. Compared to mp-grid, the size of a single cell on the screen is of constant order.

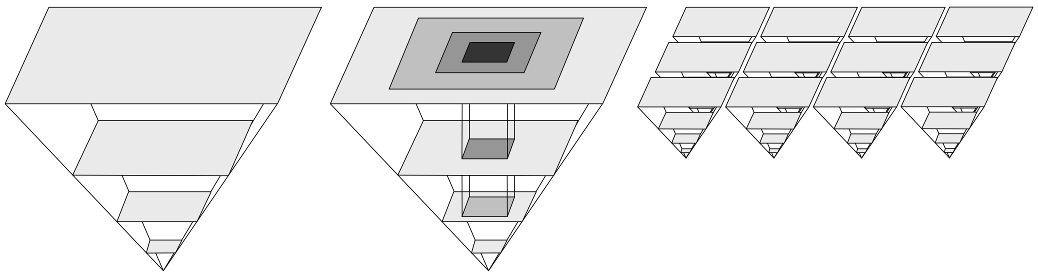


Figure 2.5: Texture LOD Data Structures. Mipmap (left), clipmap (middle) and mp-grid (right).

Error Metrics

In the same way as simplification is driven by error metrics, LOD selection is guided by error metrics in order to guarantee a certain *screen-space error*. In general, the screen space error depends on all viewing parameters: the eye position e, the object position p, the viewing direction n_i, the field-of-view ϕ and the screen resolution r. Since a precise calculation of the screen space error is quite expensive, an often applied approach is to establish only upper bounds on the object space error. The screen-space error ϵ can then be easily derived at runtime from the precomputed object space error δ. The intercept theorems state that $\epsilon = \delta \frac{d_i}{d} \cos(\alpha)$, where $d_i = \frac{r}{2} \cot(\alpha)$ and $d = (p - e) \cdot n_i$ as shown in Figure 2.6.

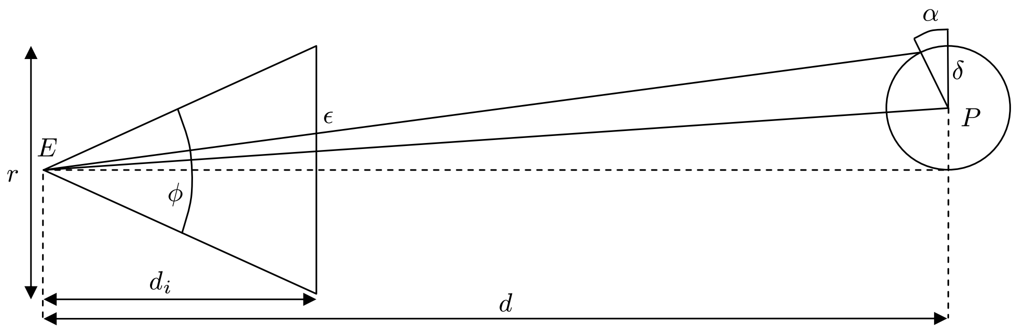


Figure 2.6: Screen-space Error. Approximation of screen-space-error by object-space error.

2.3 Terrain Visualization

Efficient interactive visualization of very large textured DEMs is important in a number of application domains, such as scientific visualization, GIS, mapping applications, virtual reality and interactive 3D games. Due to the ever increasing complexity of textured DEMs, real-time display imposes strict efficiency constraints on the visualization system, which is forced to dynamically trade rendering quality with usage of limited system resources. To best exploit the rendering resources, the scene complexity must be reduced as much as possible without leading to an inferior visual representation.

Although general data structures and algorithms as presented in the previous section are also applicable to digital terrain models, the most efficient systems to date rely on variations of these methods specifically tailored to terrain models, i.e., 2.5D surfaces (no overlap in elevation is possible for a particular geographical location). Quadtree-based multiresolution triangulations have shown to be exceptionally efficient for grid digital terrain data. An overview of quadtree-based terrain triangulation and visualization methods can be found in [Pajarola 2002], whereas a more general survey dealing with semi-regular multiresolution models can be found in [Pajarola & Gobbetti 2007].

2.3.1 The Used Terrain Engine

The work presented in this thesis builds upon the terrain rendering engine presented in [Wahl *et al.* 2004]. The main idea is to subdivide the geometry as well as the associated textures into equally sized blocks and organize them in a quadtree hierarchy (see Figure 2.7). In the remainder of this section this rendering engine is briefly introduced. First, the preprocessing procedure is described that transforms the input data, namely a digital elevation model given as a height map and corresponding aerial imagery, into a representation well suited for real-time rendering. After that, the rendering algorithm is presented that uses the preprocessed data for real-time visualization.

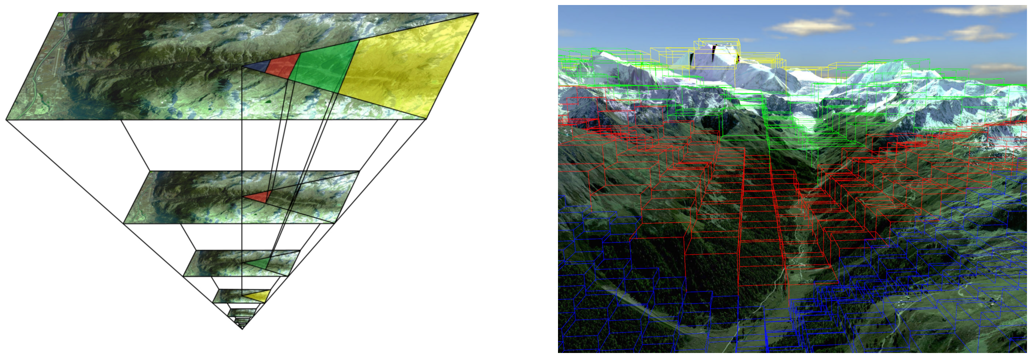


Figure 2.7: LOD Scheme. *Left:* Image pyramid of Turtmann valley together with a 2D camera frustum. In order to display the scene from the depicted camera position, textures of the visible areas are taken from different levels of the pyramid depending on their distance to the camera. *Right:* First person perspective of the same scene. The bounding boxes of the different quadtree tiles are color-coded depending on their level in the quadtree hierarchy.

Preprocessing

In the preprocessing step (see Figure 2.8) a hierarchical level-of-detail representation of the input data is created. To this end, geometry as well as

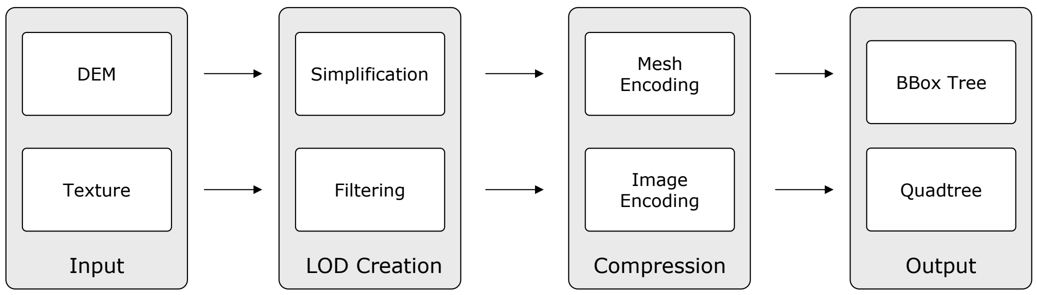


Figure 2.8: Preprocessing Stage.

textures are partitioned into equally sized tiles, which are associated with the finest level of the hierarchy. Tiles on coarser levels are constructed by geometry simplification and texture filtering, respectively, partitioning their parents' domain into equally sized quarters. Texture tiles are downsampled by a factor of two so that their resolution remains constant on all levels. Similar to the texture downsampling process, geometry tiles on higher levels are built by approximating the input mesh with half the accuracy of their children. The symmetric Hausdorff distance [Klein & Liebich 1996] between the two meshes is used to measure the approximation accuracy. Each geometry tile is represented by a triangulated irregular network (TIN). The vertices are placed on a local regular grid, which has constant resolution for all tiles of the hierarchy. In addition to that, a separate bounding box hierarchy is extracted during geometry encoding.

Finally, both texture and geometry tiles are discretized and compressed. Since the geometry tiles are represented as TINs, connectivity is encoded explicitly using a method that efficiently encodes a given traversal of the mesh. To compress the textures, standard compression algorithms, such as S3TC or JPEG, are used. S3TC compressed textures offer the great advantage that decoding is implemented on most standard graphics hardware, thus releasing the CPU from decompression. Moreover, S3TC reduces bandwidth and texture memory requirements, as the textures may reside in graphics memory in their compressed form. JPEG, however, offers better compression ratios and therefore can reduce the I/O load even further, but needs to be decoded on the CPU and resides uncompressed in texture memory. In practice a combination of JPEG (for the finest levels) and S3TC (for the remaining levels) has shown good results. In client/server settings, however, textures are exclusively transferred as JPEGs.

After preprocessing, the whole level-of-detail hierarchy of the Turtmann valley dataset, which allows viewing from large distance up to close-up views, requires only about 1/10 of the original input data size.

Rendering

Rendering is essentially parallelized among two threads (see Figure 2.9). The *rendering thread* traverses the hierarchy and selects tiles by considering their visibility and detail. Tiles that are not visible from the current viewpoint are not rendered at all, while the LOD of the remaining tiles is chosen based on the distance to the viewer, i.e., tiles close to the viewer are rendered with an accordingly higher LOD than tiles further away. During quadtree traversal a front-to-back ordering is ensured, such that a per-cell occlusion culling can be performed by conservatively testing tiles against potential occluders. The occlusion test is realized by rendering potential occluders into the depth-buffer during quadtree traversal. Visibility tests on potentially visible cells are then carried out by rendering an appropriate bounding volume of their geometry and testing if any pixels passed the depth test. As noticed by Lloyd [Lloyd & Egbert 2002], using bounding boxes as bounding volumes give satisfying results.

The *caching thread* performs an asynchronous retrieval of associated cell data (i.e., geometry and texture tiles). While traversal is still running, already requested tiles are loaded from hard disk in parallel. Once all pending requests are completed, the rendering thread hands over the cell data to the graphics hardware to be rendered. In order to avoid bursts of high workload, the caching thread can also perform a prefetching of tiles that are likely to be visible in subsequent frames based on the history of requests or a prediction of the camera path. This prefetching takes place while tiles are being rendered on the GPU.

Using the described rendering algorithm it is possible to visualize very large textured DEMs at real-time framerates. It is important to note that

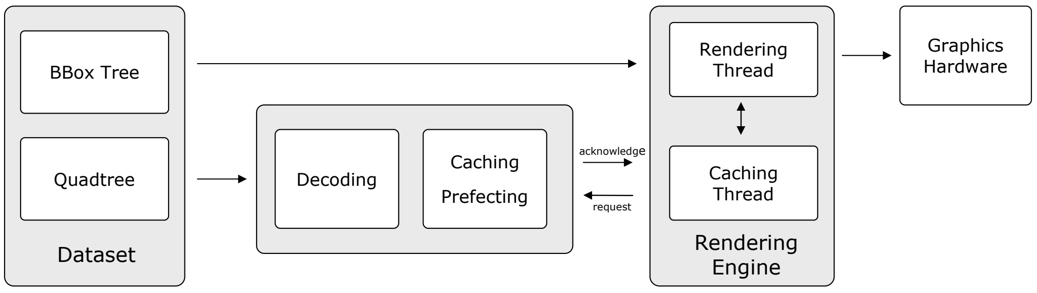


Figure 2.9: Rendering Stage.

the number of tiles to be rendered is generally constant and therefore performance is not limited by the amount of input data, but only depends on the complexity of the visible data and, of course, on the available hardware.

CHAPTER 3

Visualization of Vector Data

3.1 Motivation

Besides raster data, *vector data* are the fundamental information representation stored and managed in today's geographic information systems (GIS). While remote sensing data, such as aerial or satellite imagery, are typically represented as raster data in GIS, additional information are usually provided as layers of vector data. Such layers either aim at highlighting objects contained in the imagery, such as roads and buildings, or provide information complementing the imagery, such as state and country boundaries. Typically, vector data are either derived automatically from measurements (e.g., satellite imagery or GPS) or are manually digitized by the user. They represent geographic entities as a series of discrete point, line or polygon units. These units are different from raster data in the way that their geographic location may be independently and very precisely defined, as may be their topological relationship. Vector data are a static representation of phenomena that usually do not contain any temporal or spatial variability.

To store vector data, the Environmental Systems Research Institute (ESRI) developed the *shapefile format* [ESRI 1998]. It has become the de facto standard, especially in the GIS community, and many resources are available in this format. A shapefile stores a set of spatial features that consist of geometry and associated metadata. The geometry is described by a set of geometric shapes consisting of points, (poly)lines, or polygons. The geometric shapes are stored in a nontopological data structure (i.e., topological relationships,

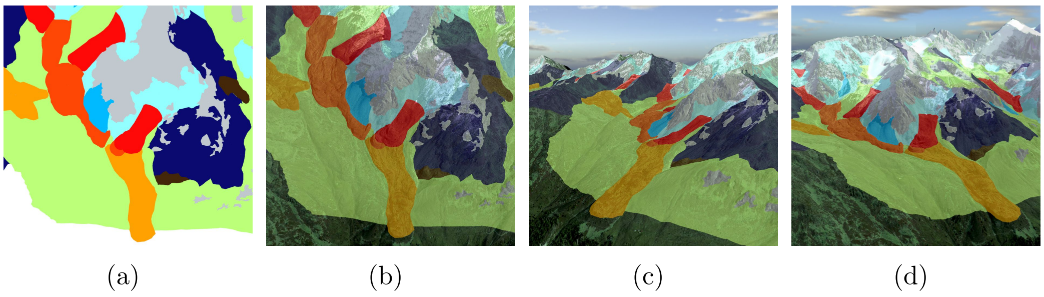


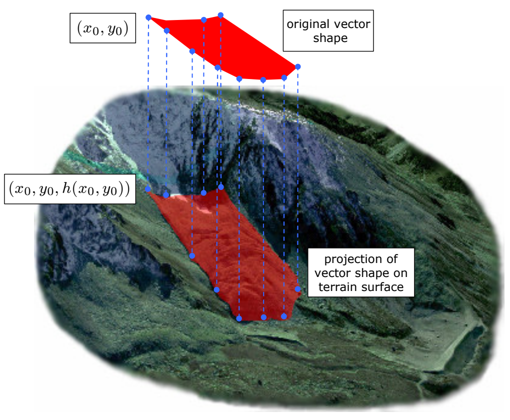
Figure 3.1: Different Map Representations. Figure (a) depicts a part of a geomorphological map of Turtmann valley given as vector data. In (b) the same map is overlayed on aerial imagery. Figures (c) and (d) show the vector data draped over a textured DEM from different views. In contrast to the fixed 2D view in (b), the 3D visualization gives the user an enhanced insight into the structure of the depicted geomorphological objects.

such as connectivity and adjacency, are not stored explicitly). The metadata are given by a user-defined set of attributes, such as textual descriptions, map colors, etc.

In practice, layers of vector data are typically overlayed on remote sensing data. To compensate for this 2D representation of the terrain surface, contour lines and different kinds of relief shading are often used as means to convey terrain topography. Nevertheless, such a representation is not easy to interpret, in particular without skills and practice in map reading. In contrast to that, a 3D visualization of the terrain provides a much more intuitive representation and is easier to understand for most people. Moreover, the visualization of vector data on a textured digital elevation model (DEM) provides additional insight into the structure of the considered objects (see Figure 3.1). In particular, the area of objects is not distorted as in the 2D case, and height differences are directly apparent. Consequently, there is a need for efficient methods that overlay vector data on textured DEMs.

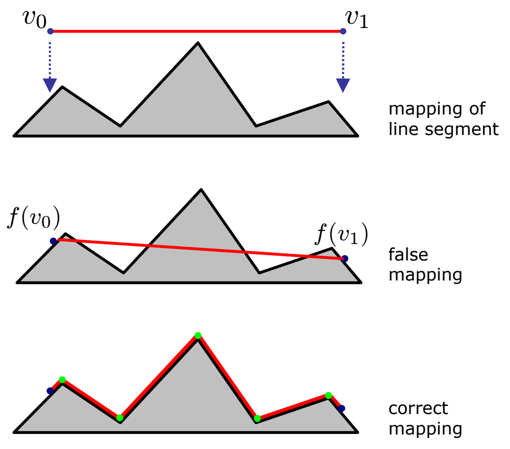
While the overlay of vector data is straightforward in the 2D case, it is much more challenging in 3D. Suppose a 2D vector shape $S \subset \mathbb{R}^2$ defined by a set 2D vertices $\mathcal{V} = \{v_1, \ldots, v_n\}$ with $v_i = (x_i, y_i)$, and a DEM

$$\begin{aligned} h : A \subset \mathbb{R}^2 &\rightarrow \mathbb{R} \\ h(x, y) &\mapsto z \end{aligned}$$



(x_0, y_0)
original vector shape
$(x_0, y_0, h(x_0, y_0))$
projection of vector shape on terrain surface

(a)



(b)

Figure 3.2: Problem Description. In (a) a 2D vector shape is projected on a textured DEM. Figure (b) shows a schematic example of a mapping of a line segment (top). If only the two vertices defining the line segment are mapped and interpolated afterwards, the result does not fit the surface (middle). The bottom image shows the correct mapping.

are given. The objective is then to map all points $p \in S$ to the height given by the DEM

$$\begin{aligned} f : S \subset A \subset \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ f(x, y) &\mapsto (x, y, h(x, y)). \end{aligned}$$

To ensure that the vector shape is consistent with the terrain surface everywhere, each point $p \in S$ has to be mapped with f. Note that it is not sufficient to map only the vertices v_i and interpolate the $f(v_i)$ afterwards (see Figure 3.2).

The remainder of this chapter is organized as follows: After reviewing related work, two novel methods for the visualization of vector data on textured DEMs are presented. Finally, results are shown and discussed.

3.2 Related Work

Methods for visualizing vector data on textured terrain models can basically be divided into overlay-geometry-based, geometry-based and texture-based approaches.

3.2.1 Overlay-geometry-based Methods

Overlay-geometry-based methods create 3D geometry from the vector shapes by adapting it to the terrain surface. The parts of the terrain mesh that are covered with vector shapes are determined, cut out and copied. To display the vector shapes, the created 3D geometry is then rendered on top of the original terrain geometry and colored or textured appropriately. The challenge for this kind of methods is to keep the created vector data geometry consistent with the original terrain geometry despite LOD changes. To this end, the vector geometry either has to be created and adapted to each level of detail in a preprocessing step or at runtime. Apart from this elaborate adaption process, the number of resulting primitives directly depends on the complexity of the terrain geometry. As a consequence, the visualization of very simple vector shapes on a high-resolution mesh requires significantly more primitives than the original 2D vector shapes consist of. Furthermore, the created geometry has to be rendered with an additional offset in order to avoid z-buffer stitching artifacts.

Wartell et al. [Wartell *et al.* 2003] presented an algorithm and an associated data structure that allows the rendering of polylines on multiresolution terrain geometry. Since their system is based upon a continuous level-of-detail terrain rendering engine, an adaption of the polyline to the current state of geometry is required at runtime resulting in additional computational costs. Agrawal et al. [Agrawal *et al.* 2006] used a similar technique for combining a textured terrain model with polyline data. They organized the terrain as a block-based LOD structure derived from a height raster. Based on this block-based simplification and visualization scheme, height values are picked up for each line segment from the underlying mesh with the highest resolution. For meshes with lower resolutions, these height values are corrected accordingly.

3.2.2 Geometry-based Methods

Geometry-based methods triangulate the vector shapes into the terrain mesh itself. An advantage of this approach compared to the class of overlay-geometry-based approaches is that the parts of the terrain mesh covered with vector data do not have to be rendered (and stored) twice. What is more, z-buffer stitching artifacts are not an issue, as the resulting terrain is still a single continuous surface but just with integrated areas that represent the vector shapes. A drawback of this kind of methods is, however, that the polygon count of the terrain mesh is increased, regardless whether or not vector data are actually visualized. Weber [Weber & Benner 2001] and Polis

[Polis *et al.* 1995] used a geometry-based approach to integrate roads into a terrain mesh. They performed this integration offline but did not deal with level of detail. Schilling et al. [Schilling *et al.* 2007] integrated polygonal vector data, such as building blocks, green areas, forests, and roads, into a custom constrained Delaunay triangulation. Then, triangles lying completely inside the vector shapes are identified and marked as owned by the respective vector shape.

3.2.3 Texture-based Methods

Texture-based methods first rasterize vector shapes into a texture and then project this texture onto the terrain geometry using texture mapping techniques. Due to the nature of texture mapping, geometry intersections or z-buffer artifacts as in geometry-based approaches are not an issue, and no additional geometry has to be extracted, stored or rendered. The simplest approach is to rasterize the vector data into a texture map in a preprocessing step and ortho-texture the terrain mesh with it. However, the main drawback of this approach is that in order to achieve reasonable quality, the resolution of the texture has to be at least as high as that of the aerial imagery. Consequently, memory requirements can become enormous in order to achieve satisfying quality. What is more, it is not possible to arbitrarily combine different layers of vector data or to visualize only selected objects of a layer. Therefore, some texture-based approaches create the texture on-the-fly and thus are able to arbitrarily combine different layers of vector shapes. However, texture generation at runtime may be costly, and the quality of the mapping is still determined by texture resolution.

Kersting et al. [Kersting & Döllner 2002] proposed a texture-based approach where the textures are generated on-the-fly by rendering the vector shapes into p-buffers. An on-demand texture pyramid that associates equally sized textures with each quadtree node is used to improve visual quality when zooming in. However, many expensive p-buffer switches have to be carried out. Even with more recent and efficient extensions, for example framebuffer objects, each switch still requires a complete pipeline flush. Bruneton et al. [Bruneton & Neyret 2008] presented a method for real-time rendering and editing of vector data using a GPU-friendly algorithm. They adaptively refine spline curves on the CPU and rasterize them into textures using triangle strips. In addition to that, they also deal with modifications of the terrain shape under constraints introduced by the vector shapes.

3.3 Texture-based Approach

In contrast to a fixed texture created in a preprocessing step, an on-the-fly generation offers several advantages: First of all, it provides improved visual quality when zooming in since resolution is not fixed but is optimized with respect to the current viewpoint. It also allows interactive editing of vector shapes as well as associated attributes (e.g., color, drawing style). What is more, different vector layers can be arbitrarily combined, and enabled or disabled as desired. Particularly for large vector maps, memory consumption is drastically reduced since only the original vector data representation and the offscreen buffer consume memory at runtime.

Previous texture-based approaches typically use some kind of pyramidal or tiled texture representation to which the vector data are rendered on-the-fly. If, however, the number of tiles becomes large, the necessary buffer switches between the different texture targets become expensive. Additionally, a clipping of vector data at tile borders or repeated rendering of vector data for all associated tiles is necessary. In contrast, the presented approach rasterizes the vector data into a single offscreen buffer only, making it very fast. To achieve good quality despite using a single texture only, it is vital to use the texture efficiently. To this end, a view-dependent reparameterization is applied. The main idea behind the reparameterization is to grant vector shapes

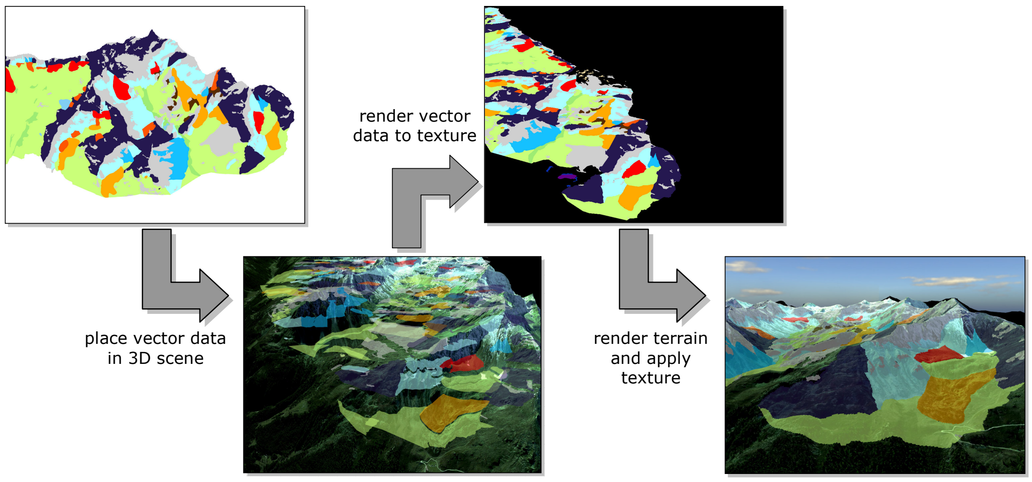


Figure 3.3: Overview of the Texture-based Method. The 2D vector shapes are placed in the 3D scene at a certain height close to the terrain. Then, based on the current view configuration a perspective transform is computed under which the vector shapes are rendered into a texture. Finally, the terrain is rendered and textured with the created texture map.

that are close to the viewer more space in the texture than objects that are further away. The reason behind this is that in the final rendered image distant objects appear smaller than nearby objects (of the same size) as a result of the perspective projection. An overview of the approach is given in Figure 3.3.

In the following, first, the algorithm used for texture generation is described, which is in particular able to efficiently handle non-simple polygons without needing to triangulate them. Then, the perspective reparameterization of this texture is presented that aims at reducing aliasing artifacts and thus improves visual quality compared to standard texture mapping with nearly no overhead.

3.3.1 Efficient On-the-fly Texture Creation

While the rendering of points and lines is straightforward, the rendering of polygons given as a set of vertices is not. To render polygonal vector shapes into the offscreen buffer efficiently, an algorithm similar to that presented by Guthe et al. [Guthe *et al.* 2005b] is used. The main advantage of this method is that it can handle non-convex polygons and polygons containing holes without needing to triangulate them in advance. The algorithm is inspired by the standard approach for area calculation of polygons. The main idea is that when spanning a triangle fan from the first vertex of each polygon ring, a point inside this ring will be covered an odd number of times as shown in Figure 3.4. Instead of counting the coverages, it is possible to simply consider the lowest bit and toggle between black and white. Since there is no need to take care of the orientation or nesting of the polygon rings, error prone

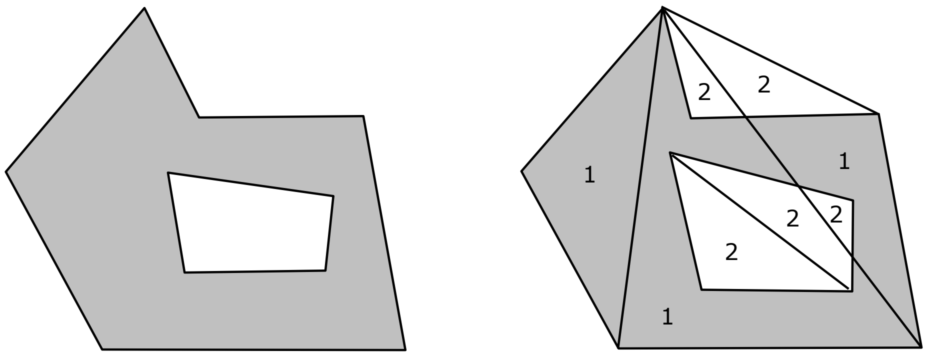


Figure 3.4: Polygon Rendering. *Left:* A concave polygon with a hole. *Right:* The polygon is drawn using a triangle fan and texel coverages are counted. Even numbers indicate outside, odd numbers inside of the polygon.

special case handling is avoided. Toggling of pixels is performed using alpha blending. Note that with this procedure the entire alpha texture for a vector layer can be generated in a single rendering pass. If only a single layer is rendered, the alpha texture can be directly used in a fragment shader. The color of the layer can be defined by setting an appropriate vertex attribute which is then multiplied in the shader before blending. If multiple layers are activated for rendering, they first have to be combined into a single texture. This is performed by accumulating the layers in a second offscreen buffer of the same size using standard alpha blending. No specialized shader is required for accumulation, since the primary color can be used to specify the color of the current layer. This way, an arbitrary number of possibly semi-transparent layers can be rendered on the terrain with only two additional textures.

3.3.2 Texture Reparameterization

In order to improve the visual quality of the mapping of vector shapes, a perspective reparameterization of the on-the-fly created texture depending on the current view is proposed. To motivate the application of the reparameterizaton, the aliasing problem arising in the texturing process is described first (see Figure 3.5). Each texel of size d_t of the texture is orthogonally projected into the scene. If a surface is hit under an angle β, the size of the projection is

$$\frac{d_t}{\cos\beta}.$$

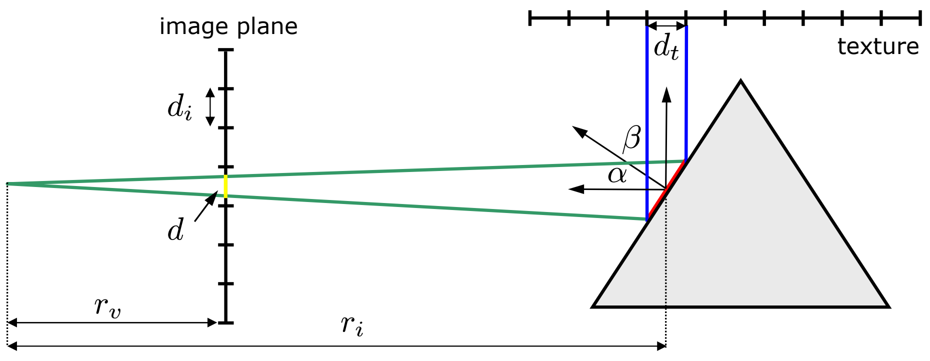


Figure 3.5: Aliasing in Texture Mapping.

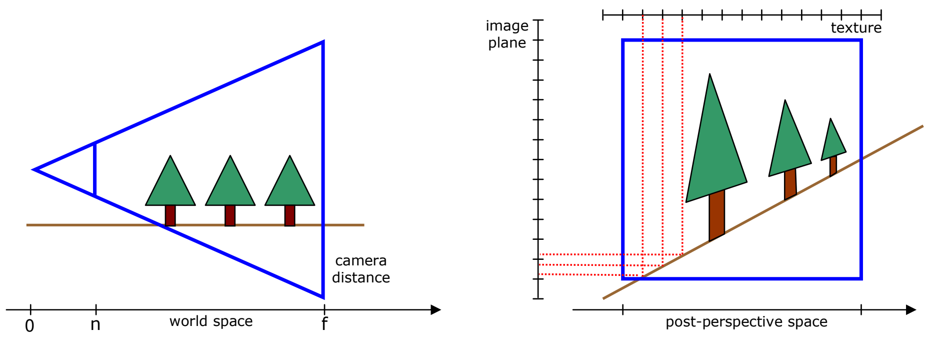


Figure 3.6: Texture Generation in Post-perspective Space. If the texture generation is performed in post-perspective space, texture map pixels projected on the ground are evenly distributed in the final image.

In the final image the projection area of the texel then has size

$$d = \frac{d_t r_v}{r_i} \frac{\cos \alpha}{\cos \beta},$$

where α is the angle between viewing direction and surface normal, r_i is the distance from the camera to the surface and r_v is the distance from the camera to the image plane. Undersampling occurs when d is larger than the image pixel size d_i, which can happen if either $\frac{d_t r_v}{r_i}$ (perspective aliasing) or $\frac{\cos \alpha}{\cos \beta}$ (projective aliasing) becomes large. Projective aliasing typically occurs for surfaces almost parallel to the projection direction. Since projection aliasing heavily depends on the scene's geometry, a local increase of sampling density is needed to reduce this kind of aliasing. This requires an expensive scene analysis at each frame and complex data structures, and can not be accelerated by current graphics hardware. Perspective aliasing, however, is caused by the perspective projection of the viewer. It can be reduced by applying a perspective transformation that change the distribution of texels in the scene.

Similar aliasing problems arise in shadow mapping [Williams 1978]. In this context, several papers have addressed the perspective aliasing problem. The most prominent of them is the perspective shadow map method by Stamminger and Drettakis [Stamminger & Drettakis 2002]. Perspective shadow maps attempt to reduce perspective aliasing by performing a perspective reparameterization. To this end, texture generation is performed in *post-perspective space* (see Figure 3.6), which significantly reduces perspective aliasing that is caused by the perspective projection of the viewer. Later, perspective shadow maps were improved by light space perspective shadow maps [Wimmer *et al.* 2004]. The first step in this method is to calculate a

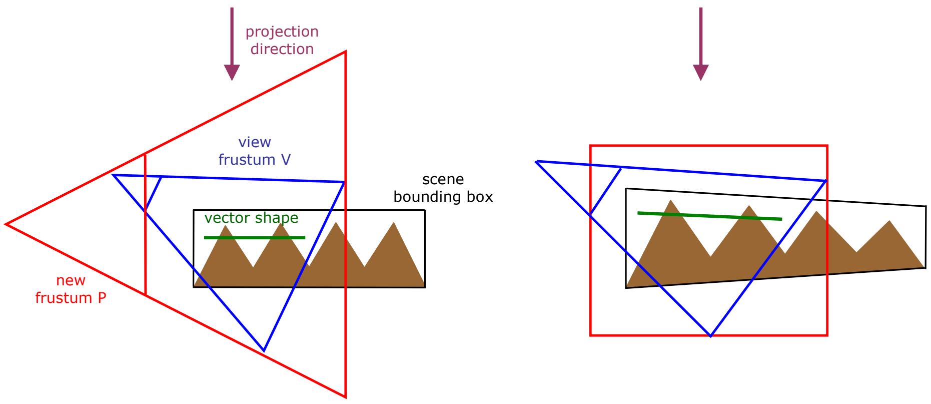


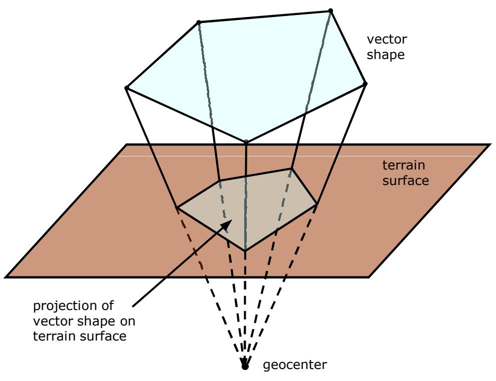
Figure 3.7: Texture Reparameterization. The left image shows an example configuration with the original view frustum V (blue), the scene bounding box (black) enclosing the terrain geometry (brown) and a vector shape (green). From this, the frustum P (red) is computed that defines the perspective transform. The image on the right shows the same configuration after the perspective transformation. Note that the projection direction remains unchanged.

convex body that encloses all relevant parts of the scene including the view frustum and all objects that can cast shadows on it. Next, this volume is enclosed with a perspective frustum that has a view vector orthogonal to the light direction. By varying the length of this frustum the strength of the warping effect can be controlled. The perspective transformation induced by the frustum is applied in two places, namely during the creation of the shadow map and in the computation of texture coordinates during rendering.

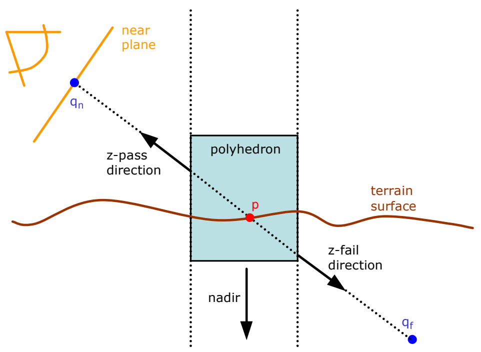
In order to improve the visual quality of the mapping of vector shapes, a perspective reparameterization of the on-the-fly created texture depending on the current view is applied. It adapts the technique used in light space perspective shadow mapping to the visualization of vector data. In contrast to perspective shadow mapping, there is no lighting but a projection direction in which the vector shapes are mapped onto the terrain (see Figure 3.7). Because of this a frustum P is chosen with a view vector orthogonal to the projection direction of the vector shapes. This frustum has to include all relevant parts of the terrain geometry as well as the 2D vector shapes, which can theoretically be positioned at an arbitrary height. Although the actual height of the vector data is not important considering the projective nature of the approach, they are rendered close to the surface to minimize the size of the frustum P. Then, the vector shapes are rendered into the offscreen buffer using the transformation associated with P. Once the texture has been created, the offscreen buffer is bound as a texture. Finally, when the terrain is rendered, appropriate texture coordinates are calculated according to the perspective projection of P and are used to access the texture. Blending with the terrain texture is carried out depending on the texture's alpha component.

3.4 Stencil-based Approach

The visualization of 2D vector shapes on top of a digital elevation model basically requires the determination of points in 3D space that lie on the terrain surface and whose 2D geographic location corresponds to that of the vector data. However, when transforming a 2D vector shape into 3D space its position is defined only up to its height. The possible positions of a vertex in 3D are therefore described by a ray originating at the geocenter with a direction defined by the 2D geographic coordinate of the vertex. Considering a polygonal vector shape, this results in a volume as depicted in Figure 3.8 (a). Since the vector shape is also supposed to lie on the terrain surface, the intersection between the volume and the terrain surface contains all points that meet both requirements. Consequently, by checking if a point on the terrain surface is located inside the volume of a given vector shape, it can



(a)



(b)

Figure 3.8: Problem Formulation. Figure (a) shows a polygonal vector shape and the corresponding volume indicating positions in 3D space with equivalent 2D geographic coordinates. The intersection of the volume with the terrain surface describes the area on the terrain covered with the respective vector shape. The determination of this intersection can be formulated as a point-in-polyhedra problem. Figure (b) shows a 2D illustration of the point-in-polyhedra problem.

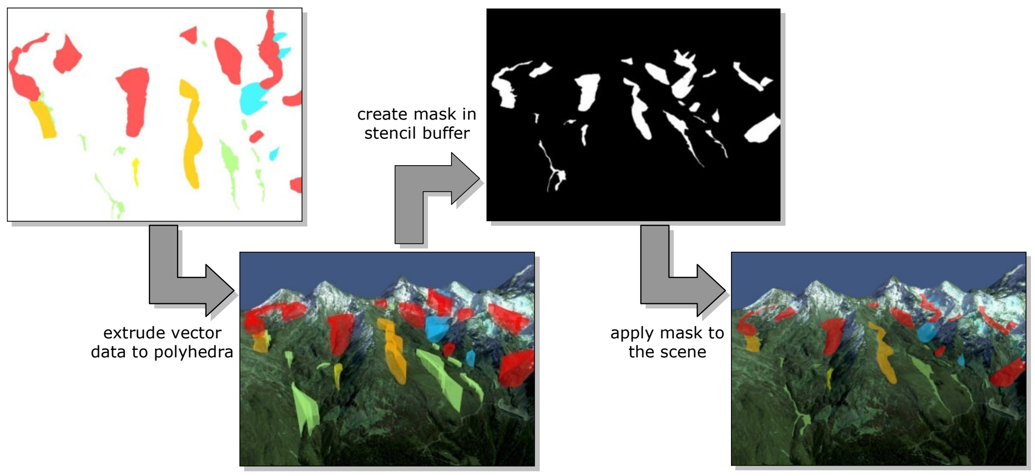

create mask in stencil buffer
extrude vector data to polyhedra
apply mask to the scene

Figure 3.9: Overview of the Stencil-based Method. First, the original vector shapes are extruded to polyhedra. Then, the polyhedra are used to create a mask in the stencil buffer that indicates the areas where the terrain surface and polyhedra intersect. Finally, the mask is used to overlay the vector shapes on the current view.

be determined if the point is part of the vector shape or not. This way, the problem of visualizing 2D vector shapes on textured elevation model can be formulated as a *point-in-polyhedra problem* (see Figure 3.8 (b)). Since this test can be performed efficiently in screen-space using graphics hardware, it can be used for an accurate visualization of vector data that completely avoids the aliasing problems of texture-based approaches.

In the remainder of this section, first the point-in-polyhedra problem is formalized and it is described how graphics hardware can be used to compute it efficiently. Then, the different steps of the algorithm as shown in Figure 3.9 are explained in detail. In the first step, vector shapes are extruded to polyhedra, indicating positions in 3D space with equivalent 2D geographic coordinates. Then, these polyhedra are used to create a mask in the stencil buffer indicating the position of the vector shape in image space with respect to the current view. Finally, this mask is used to render the vector data into the current view.

3.4.1 Point-in-polyhedra Algorithm

A general algorithm for performing a point-in-polyhedra test can be formulated as follows: Assume a point q that is outside all polyhedra is given. For a point p in question the objective is then to find all intersections of the line segment $\overline{pq}$ and the polyhedra. At each intersection a counter is incremented if the line enters and decremented if the line exits the polyhedron. After all intersection tests have been performed, the counter corresponds to the number of polyhedra containing p. If the counter is zero, then p is outside all polyhedra. Otherwise, if the counter is non-zero, p is inside at least one polyhedron.

The problem of shadow determination that can also be expressed as a point-in-polyhedra problem [Crow 1977]. Crow defined a shadow volume as a region of space that is in the shadow of a particular occluder given a particular ideal light source. The shadow test determines if a given point is inside the shadow volume of any occluder. Heidmann [Heidmann 1991] adapted Crow's algorithm to hardware acceleration by exploiting the stencil buffer to evaluate the per-pixel count for the point-in-polyhedra test.

By rendering the polyhedra's front- and back-faces to the stencil buffer, the point-in-polyhedra test can be performed simultaneously for all visible points of a scene. Each pixel is interpreted as a point p and the ray from the viewpoint through the pixel is considered. There are two possible choices for a point q along the ray outside any polyhedron (see Figure 3.8 (b)). The first possible choice is the intersection q_n of the ray with the near clipping plane. This point is known to be outside all polyhedra if the near clipping plane does not intersect any polyhedra. The alternative is the point q_f at infinity at the far end of the ray. This point is always outside all polyhedra because it is infinitely far away from the scene.

Entering intersections correspond to polyhedra front-faces while exiting intersections correspond to polyhedra back-faces. Thus, counting intersections can be performed by rasterizing the polyhedra faces in the stencil buffer with the stencil operation configured to increment the stencil value when a front-face is rasterized and to decrement the count when a back-face is rasterized. However, intersection counting has to be performed only along $\overline{pq_n}$ or $\overline{pq_f}$ respectively, but not along the entire ray. Since p is a visible point, these two kinds of intersections can be discriminated by a depth test. If q_n at the near clipping plane is used, only polyhedra faces passing the depth test are counted, whereas if q_f at infinity is chosen, only the polyhedra faces failing the depth test are considered. Counting towards the near clipping plane is thus called the *z-pass* method, whereas counting towards infinity the *z-fail* method [Everitt & Kilgard 2002][McGuire *et al.* 2003]. Once rendering has been finished, a stencil value of zero indicates that the same number of front- and back-faces was rendered and thus the corresponding pixel is outside all polyhedra. Otherwise the pixel is inside at least one polyhedron.

The z-pass method fails whenever a polyhedron intersects the near clipping plane, since in this case the assumption that q_n is outside all polyhedra does not hold true for all pixels. This near clipping problem has led to the development of the z-fail technique in the first place, which processes fragments that fail (instead of pass) the depth test. The the z-fail method moves the intersection problem from the near to the far clipping plane. This is advantageous because intersections with the far clipping plane can be avoided by moving the far clipping plane to infinity.

3.4.2 Polyhedra Construction

The first step of the algorithm is to extrude the vector shapes to polyhedra. The construction is started by duplicating each vertex of the vector shapes. While one vertex of each of the created pairs is translated towards the geocenter, the other is moved into the opposite direction. The set of upper and lower vertices constitute the polyhedron's top and bottom cap. The amount of translation is chosen such that the top and bottom cap are located completely above and below the terrain surface, respectively. Applying the described construction, the resulting polyhedron encloses the part of the terrain surface that is supposed to contain the vector shape.

In order to minimize the rasterization workload (typically the bottleneck when using shadow volumes) caused by rendering the polyhedra, their size is reduced as much as possible. To accomplish this, top and bottom caps of the polyhedra are created as close as possible to the terrain surface without intersecting it. In the current implementation the bounding boxes of the quadtree cells containing the terrain geometry are utilized for this. The bounding boxes encode an upper and lower bound of the enclosed terrain and therefore provide reasonable upper and lower bounds for the polyhedra as well.

Vector shapes consisting of polylines or point primitives are assigned a user-defined fixed width or radius respectively (i.e., they are converted to polygons before they are extruded to polyhedra). In the case of polylines height values of corresponding vertices of the top and bottom cap are determined as the minimum and maximum height values of the bounding boxes containing the projection of the line segment (see Figure 3.10). In the case of polygons the minimum and the maximum height value of the bounding boxes enclosing the projection of the whole polygon is used. After appropriate height values have been determined, the constructed polyhedra are tesselated ensuring a consistent winding order with all face normals pointing outwards. The resulting geometry of each object is stored in its own vertex buffer object remaining valid as long as the vector shape is not modified.

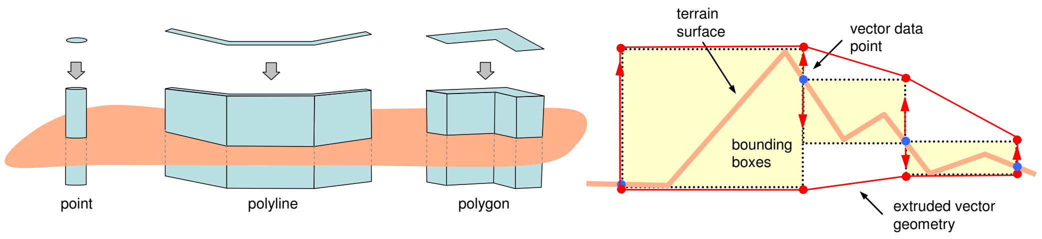


Figure 3.10: Vector Shape Extrusion. Left: Extrusion of points, polyline and polygon. Right: A 2D diagram of the extrusion of a polyline. The original vector shape points (blue) are duplicated and moved to the upper and lower bounds of the respective bounding box constituting the top and bottom caps (red).

3.4.3 Mask Generation

After the polyhedra have been created from the vector shapes, they are rendered into the stencil buffer. An often applied approach when using shadow volumes to decide on a per frame and volume basis if the z-pass or the z-fail technique is used. The z-pass method is preferred because it does not need capping (i.e., top and bottom caps need not to be rendered) and is therefore generally faster than z-fail. However, since the z-pass technique does not produce correct results when the near plane intersects a shadow volume, the robust z-fail technique is applied in such cases. Following this approach, it is decided conservatively which method is used by checking if the current viewport is inside the bounding box of the considered polyhedron. In contrast to the shadow volume algorithm, a top cap is needed for the polyhedra in the z-pass case. The reason for this is that there is no occluder that casts the shadow and acts as a top cap when visualizing vector data.

Before the polyhedra are rendered into the stencil buffer, color, depth and stencil buffer are cleared first and the terrain is rendered initializing the depth buffer with the required depth values. Next, depth buffer writing is disabled, while the depth test still remains active, and rendering is restricted to the stencil buffer only. A polyhedron's faces are rendered using different stencil operations depending on whether they face towards or away from the camera. To this end, face culling is enabled and the polyhedron is rendered twice, one time with back-face culling enabled, the other time with front-face culling enabled. If the z-pass method is used because the polyhedron does not intersect the near clipping plane, the values in the stencil buffer are modified when the depth test passes. The stencil value is incremented for fragments belonging to front-facing polygons and decremented for fragments belonging to back-facing polygons. If the z-fail technique is applied, values in the stencil buffer are modified when the depth test fails. The stencil value is incremented for fragments belonging to back-facing polygons and decremented for fragments belonging to front-facing polygons.

The OpenGL extensions `EXT_stencil_wrap` and `EXT_stencil_two_side` are used, if supported, which aim at simplifying the mask creation in the stencil buffer. The `EXT_stencil_wrap` extension specifies two additional stencil operations. These new operations are similiar to the existing increment and decrement operations, but wrap their result instead of saturating it, which leads to a reduction of the likelihood of incorrect intersection counting due to limited stencil buffer resolution. The `EXT_stencil_two_side` extension provides two-sided stencil testing where the stencil-related state can be configured differently for front- and back-facing polygons. With two-sided stencil testing front- and back-faces can be rendered in a single pass instead of two separate passes, which may improve performance.

A simple triangle fan can be used to draw the top and bottom caps, without needing to triangulate them in advance, even if they are non-convex or contain holes. The fan itself may be convex but the pattern of front- and back-faces will produce the correct non-convex shape in the stencil buffer. The process is similar to the rendering of polygons in the texture-based method. An example is shown in Figure 3.11. This technique has previously been used for rendering filled silhouettes in the stencil buffer from possible

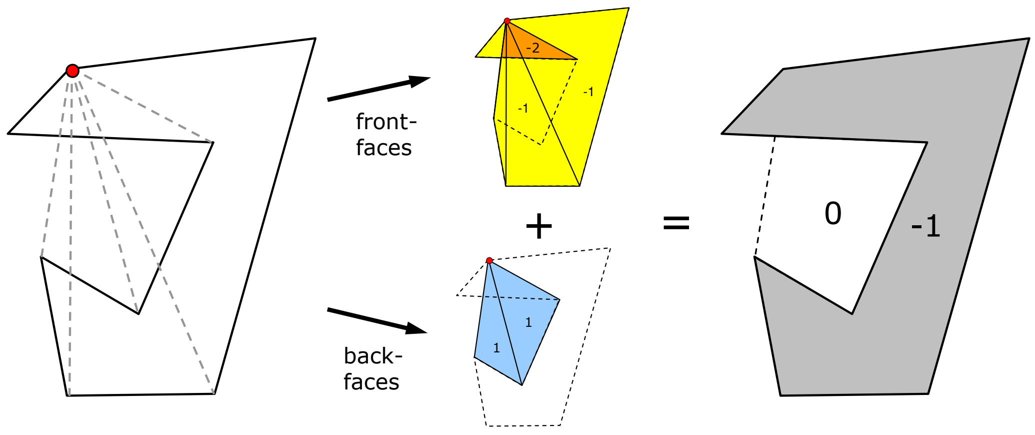


Figure 3.11: Triangle Fan Example. The concave polygon on the left is tessellated into a fan. In the middle decrements caused by front-facing polygons (top) and increments caused by back-facing polygons (bottom) are shown. Combining increments and decrements results in the original concave polygon (right).

silhouette edges for Silhouette Clipping [Sander *et al.* 2000].

To avoid far plane clipping in the z-fail case, the far plane is moved to infinity. This can be realized by using the following OpenGL projection matrix

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

to transform from eye-space to clip-space, where n and f are the respective distances from the viewer to the near and far clipping plane. $(l, b, -n)$ and $(r, t, -n)$ specify the (x, y, z) coordinates of the lower-left and upper-right corners of the near clipping plane. Positioning the far plane at infinity typically reduces the depth buffer precision only slightly. If the OpenGL `NV_depth_clamp` extension is supported and enabled during rendering of the polyhedra, the conventional projection matrix can be kept.

3.4.4 Mask Application

After the mask has been created in the stencil buffer, it is applied to the scene. To this end, writing to the color buffer is reactivated and additive blending is enabled. The stencil test is configured to pass only when the value in the stencil buffer does not equal zero. Instead of drawing a screen-sized quad to apply the mask to the scene, the bounding box of the respective polyhedron is rasterized in order to save rasterization bandwith. This is performed with depth test enabled and drawing only front-faces in the z-pass case and with depth test disabled and drawing only back-faces in the z-fail case. In order to avoid a complete stencil clear per object, the stencil function is configured such that the value in the stencil buffer is set to zero for each fragment that passes the stencil test. As a consequence, the entire stencil buffer is zero again when the rendering of a polyhedron is finished and thus does not need to be cleared.

The creation of the mask and its application to the scene has to be performed for each object separately. This is necessary because each object is allowed to have a different color and it is not possible to distinguish between individual objects once the mask has been created. If there are only objects with few different colors in the scene, sorting by color and then rendering each color group at once can help to reduce the required fill rate and state changes. An overview of the rendering process is given in Figure 3.12.

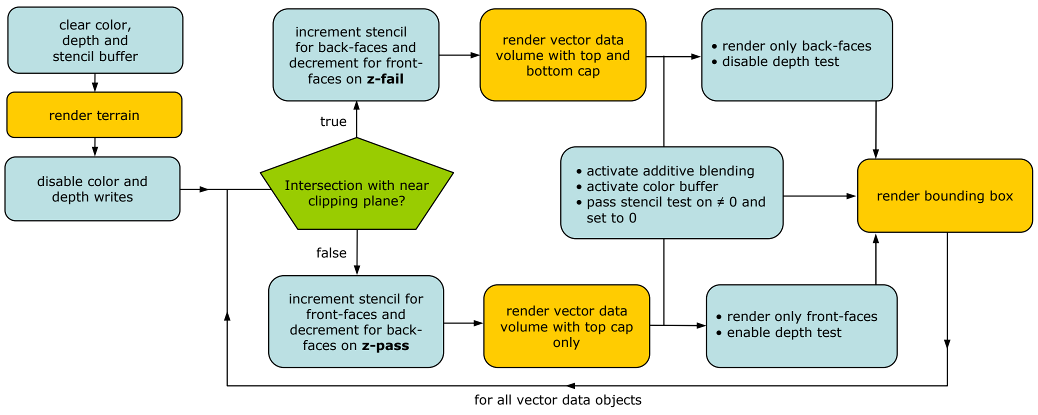

clear color, depth and stencil buffer
render terrain
disable color and depth writes
increment stencil for back-faces and decrement for front-faces on **z-fail**
true
Intersection with near clipping plane?
false
increment stencil for front-faces and decrement for back-faces on **z-pass**
render vector data volume with top and bottom cap
render vector data volume with top cap only
• render only back-faces
• disable depth test
• activate additive blending
• activate color buffer
• pass stencil test on ≠ 0 and set to 0
render bounding box
• render only front-faces
• enable depth test
for all vector data objects

Figure 3.12: Overview of the Rendering Process.

3.5 Results and Discussion

The presented approaches are well suited for visualizing large geospatial maps at real-time framerates. They do not require any elaborate and time-consuming preprocessing, instead vector data given as shapefiles can be loaded on-demand and immediately visualized at runtime. Different layers can be arbitrarily combined and individual objects can be enabled or disabled by the user. Furthermore, both methods allow interactive editing of vector shapes. A big advantage of the presented methods compared to geometry-based method is that they are independent of the terrain complexity, that is, computational costs for vector data visualization do not depend on the resolution of the underlying DEM but only depend on the number of primitives in the vector data. Considering the fast evolving acquisition devices resulting in ever higher sampled terrain datasets this fact will become even more important in the future.

The presented texture-based approach generates a texture from the vector data on-the-fly. This has the advantage that neither a texture nor a complete texture pyramid has to be precomputed and loaded into memory. Instead, only the much more compact polygonal representation of the vector data is required. Since the texture is generated each frame, the vector shapes can be interactively changed by the user at runtime. With the utilization of the perspective reparameterization, aliasing artifacts are significantly reduced and a quality superior to standard texture mapping is achieved. Figure 3.13 shows the quality improvement achieved by the applied reparameterization compared to a uniform parameterization.

Although the presented texture-based approach reduces perspective alias-

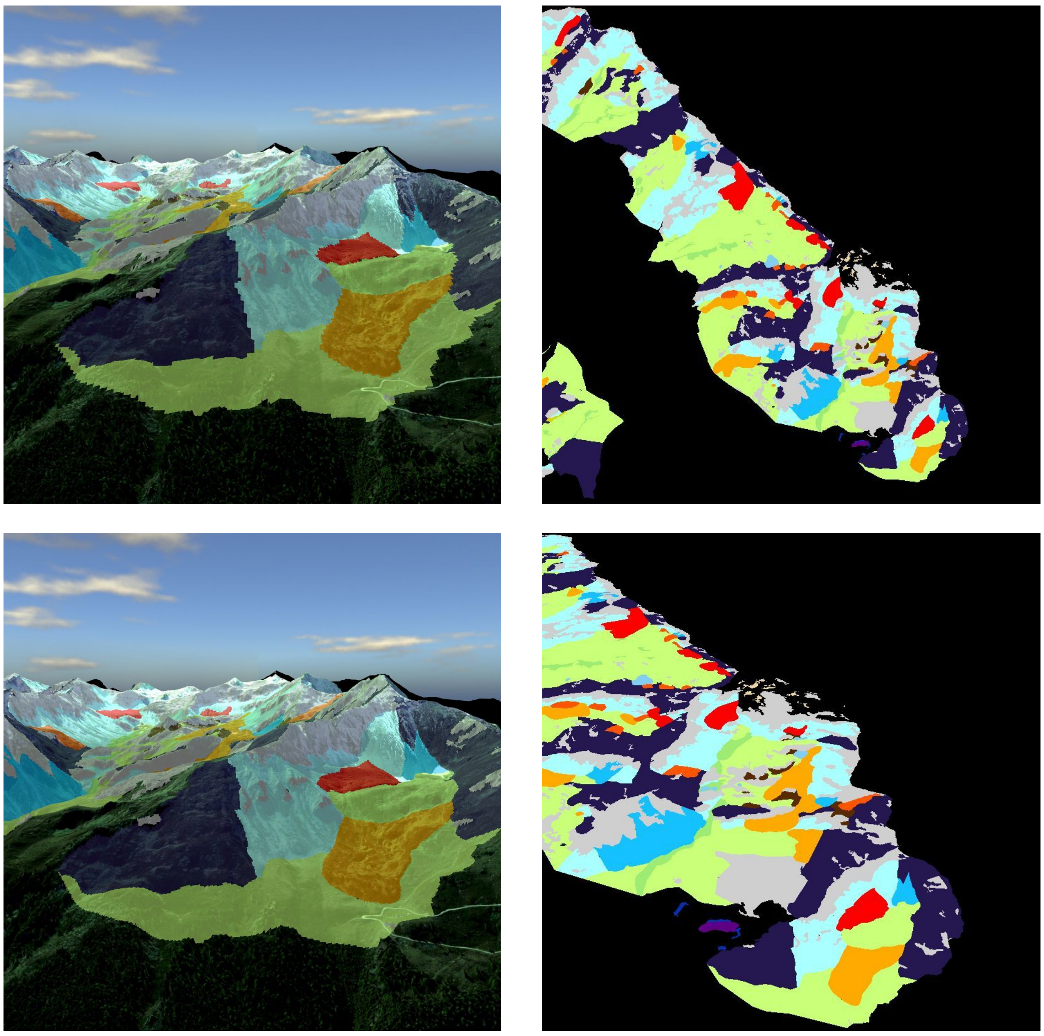


Figure 3.13: Effect of Texture Reparameterization. The top row shows a rendering of vector data using a uniform parameterization (left) and the corresponding texture (right). The bottom row shows the same but using the presented reparameterization. The rendering using the reparameterized version shows superior quality. In the uniform texture representation, vector shapes that are close to the viewer and those that are farther away have the same size. In contrast to that, in the reparameterized representation vector shapes that are further away from the viewer occupy less space than vector shapes close to the viewer.

ing, projective aliasing problems at steep slopes still remain (see Figure 3.14 (c)). Since the view frustum size is adjusted each frame to provide an optimal utilization of the available texture memory, the quality of the texture-based

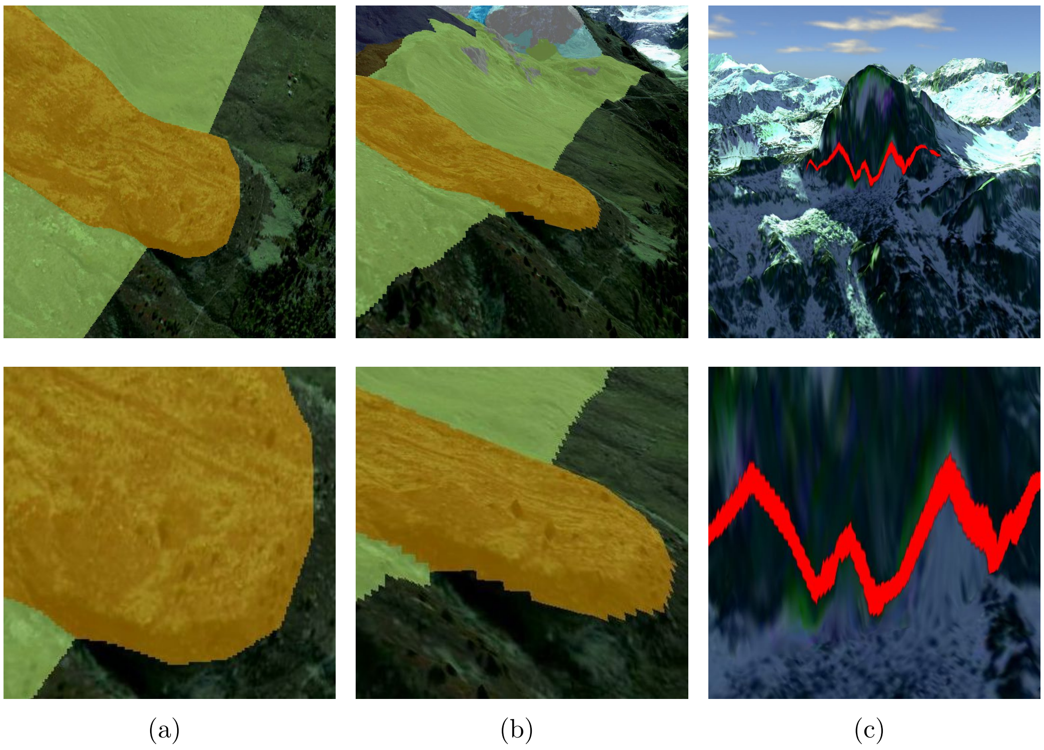


Figure 3.14: Limitations. Depending on the extent of the view frustum, the quality of the texture-based approach varies noticeably if the texture resolution is not chosen large enough. This effect is demonstrated in columns (a) and (b), which show the same area of the terrain observed from different viewpoints (top) and corresponding closeups (bottom). The quality in (b) is worse since a larger part of the terrain is visible and thus the texture has to be used for a larger area. The images in column (c) show the projective aliasing problem, which occurs at steep slopes and is especially noticeable for narrow features like lines.

method may vary depending on the current view configuration. In particular when the view frustum is very large, undersampling can occur if the current texture resolution is not sufficiently high. Even worse, in such cases disturbing "swimming artifacts", that is, vector shapes seem to frequently change their shape when the viewpoint changes (see Figure 3.14 (a) and (b)), may be visible. However, the aforementioned artifacts only occur if the resolution of the offscreen buffer with respect to screen resolution is chosen too low. Therefore, it is important to select the texture resolution in such a way that even in adverse view configurations sufficient quality can be guaranteed. In practice good quality is achieved by choosing texture resolution

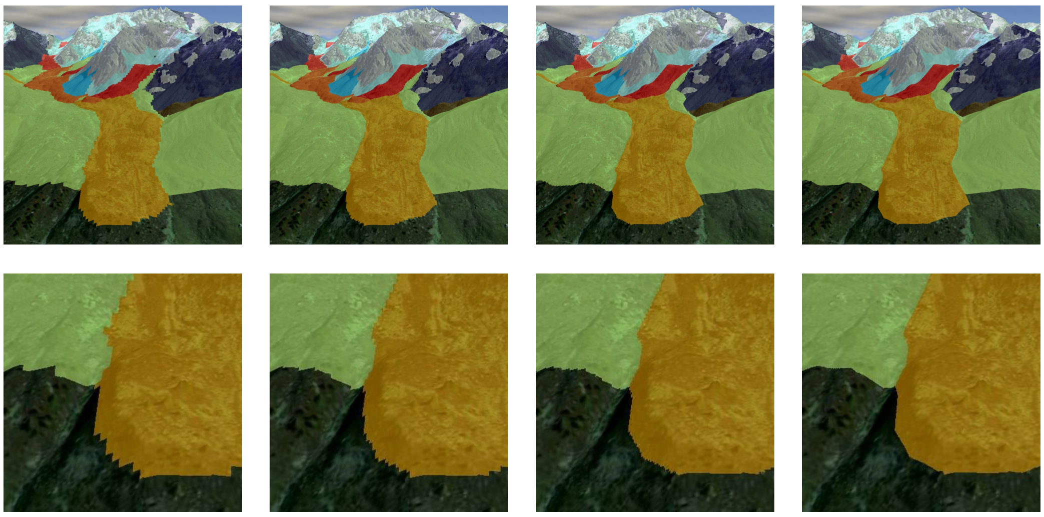
shapefile	#points	w/o	texture(1K/2K/4K)	stencil
buildings	12628	100	70/69/68	55
geom. map	99140	100	60/60/59	49
soil types	644958	100	45/44/44	38

Table 3.1: Framerates. The size of the used texture has nearly no influence on the performance of the texture-based approach. Since in the stencil-based approach more primitives have to be rendered, it is slower compared to the texture-based approach.

twice as large as screen resolution. Figure 3.15 shows a comparison of the quality achieved with different texture resolutions. An alternative to improve the quality of the presented texture-based approach without increasing texture memory consumption is to apply the idea of parallel-split shadow maps (PSSMs) [Zhang *et al.* 2006]. PSSMs split the view frustum into different parts by using planes parallel to the view plane and then generate multiple smaller texture maps for the split parts. While this technique increases quality without demanding more texture memory, the use of multiple texture maps reduces performance.

The stencil-based method allows high-quality vector data visualization as provided by geometry-based methods. Yet, it does not suffer from their shortcomings, namely the elaborate adaption process and the increased primitive count depending on the terrain complexity. Although the stencil-based method requires rendering a multiple of the amount of primitives contained in the original vector shapes, it is only a small, constant factor independent of the underlying terrain geometry.

In comparison to the presented texture-based technique that immediately renders the vector data into a texture, the stencil-based method requires more primitives to be rendered and is therefore slightly slower (see Table 3.1). On the other hand, it does not suffer from aliasing artifacts as the texture-based approach and hence provides superior quality (see Figure 3.15). However, the results are hardly distinguishable from the texture-based approach in most areas if a texture resolution of 2048×2048 or larger is used (assuming a moderate screen resolution). However, at steep slopes the stencil-based approach provides superior quality compared to the texture-based approach even if high resolution textures are used. Interactive editing and manipulation of vector data is also possible with the stencil-based approach as it only requires updating the polyhedra accordingly. In the current implementation vector shape extrusion is performed on the CPU. However, it could also be



(a) 256×256   (b) 512×512   (c) 1024×1024   (d) 4096×4096

Figure 3.15: Quality Comparison. The top row depicts screenshots of the geomorphological map of Turtmann valley visualized using the texture-based approach at different resolutions for the used texture. The bottom row shows closeup views of the respective images above.

carried out on the GPU, which might come in handy for future applications such as animated vector data.

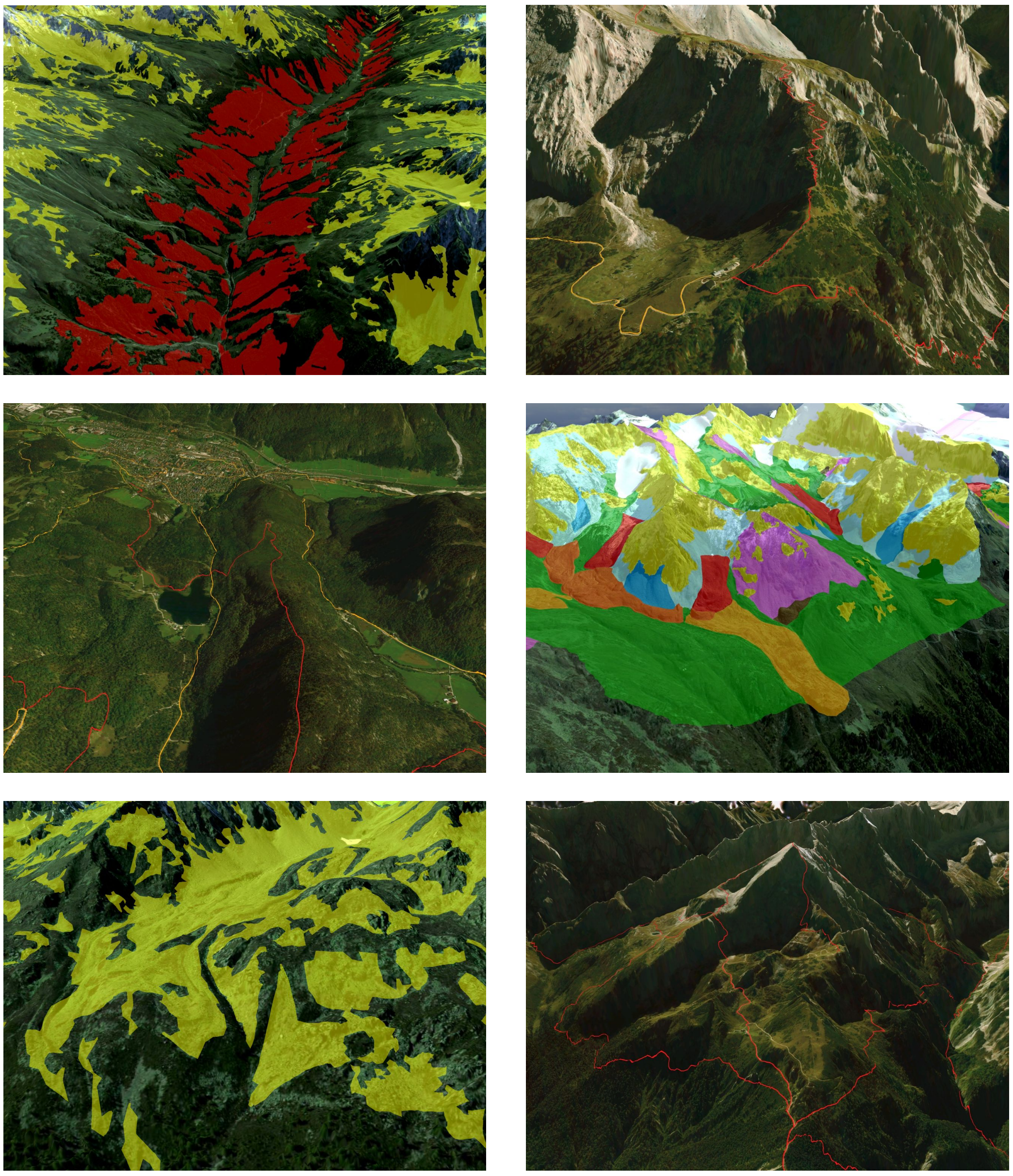


Figure 3.16: Results. Some results obtained with the presented method. The screenshots show complex vector data of polygonal and polyline vector data rendered onto a textured DEM.

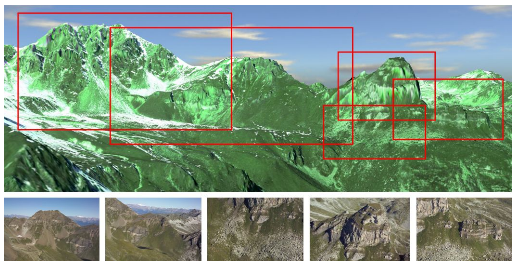
CHAPTER 4

Registration

4.1 Motivation

Since their origin in the late 1950s, digital elevation models (DEMs) along with corresponding aerial imagery have found wide application in various disciplines, such as mapping, remote sensing, land planning and communications. During this same time, acquisition and processing of terrain data as well as corresponding visualization techniques have continuously progressed so that nowadays a real-time visualization of large, high-resolution terrain datasets has become feasible. However, a major drawback of aerial imagery is the irregular sampling of the terrain surface leading to a coarse representation of steep slopes, while overhangs are not captured at all. In addition to that, clouds or shadows might obstruct the view to areas of interest. Apart from poor visual quality, this also severely limits the applicability of the data in disciplines, such as geomorphology.

Considering the quality and availability of today's digital cameras, the use of high resolution photos offers an easy and affordable way to capture additional information in areas of interest. However, in order to be able to complement the aerial imagery with photos, the photos have to be georegistered first (see Figure 4.1). Georegistration involves the computation of accurate information about the camera's absolute location and orientation in a georeferenced coordinate frame as well as its intrinsic parameters, such as focal length. Some of these information can be provided with GPS devices and electronic compasses. Also, many modern digital cameras embed


(a)

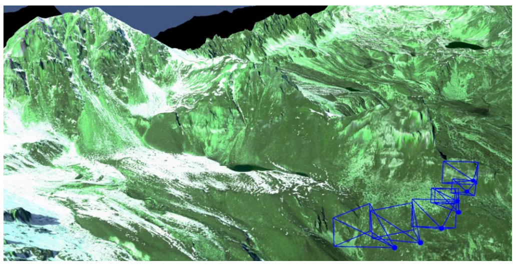
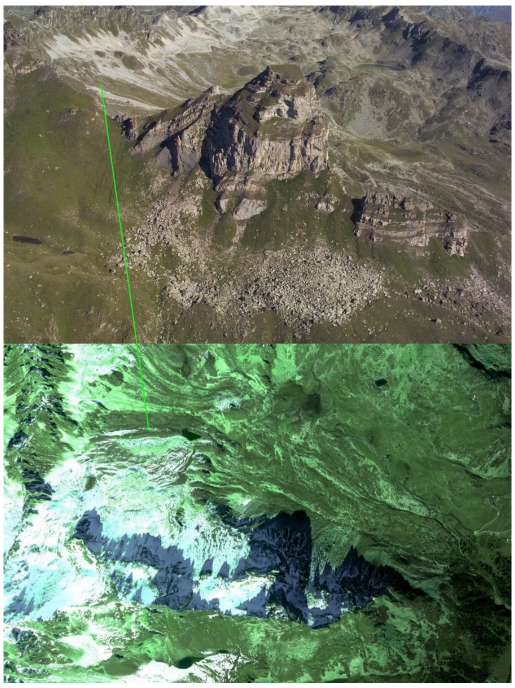

(b)

Figure 4.1: Motivation. (a) The top image shows a screenshot of the textured DEM. Texture quality in the steep areas is very low. The bottom row depicts some example photos of the same area taken on-site. The goal is to register the images to the textured DEM. (b) The image shows the frustra of the estimated cameras associated with the photos.

focal length and other information in EXIF (Exchangeable Image File Format) tags of produced image files. However, the vast majority of existing photographs lack such information. Furthermore, EXIF tags are sometimes inaccurate and camera parameters can not be estimated from them alone.

One approach to georegister images is to place marker points in the field and measure their location, for example via GPS. Then, when a photo of this area is taken, the marker points are detected in the photo and are used to estimate the camera parameters. Unfortunately, this approach is elaborate and time-consuming, in particular in high alpine environments, and thus impracticable when many images have to be registered. In such cases, structure from motion (SfM) techniques that do not rely on the camera or any other piece of equipment to provide camera parameters become especially attractive. The goal of SfM is to simultaneously recover the unknown 3D scene structure and camera parameters from a set of feature correspondences in the given images. However, for the application at hand the images have to be aligned to a textured DEM in addition to be registered to each other (i.e., not only *relative*, but *absolute* camera parameters have to be computed).

To georegister photos, two kinds of approaches have previously been applied. One approach is to first register the photos to each other using some structure from motion technique. Then, in a second step the resulting 3D scene points are fit to the given DEM in order to upgrade the camera parameters from relative to absolute ones. To this end, the user typically has to specify correspondences between the reconstructed point cloud and the DEM (which can be difficult). Moreover, this approach completely ignores the information contained in the given textured DEM during the SfM opti-


(a)

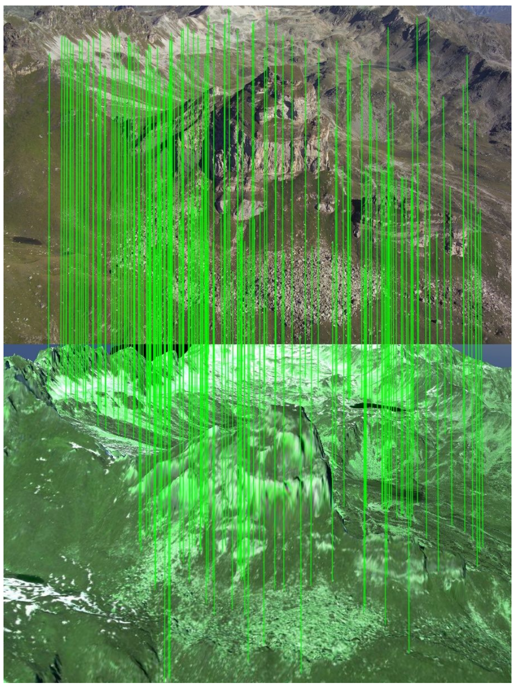

(b)

Figure 4.2: Problem. (a) A photo is matched to the aerial imagery using the method of Lowe [Lowe 2004]. Only one (false) match is detected, far too few needed for a successful registration. (b) By contrast, using a constraint matching (as presented later in section 4.3.3) between the photo and a screenshot of the textured DEM, the number of matches can be increased to more than 100.

mization. The other approach is to directly register the photos to the aerial imagery using image registration techniques. However, due to widely different illumination conditions, resolution and especially viewpoint, registration is very difficult and often fails (see Figure 4.2 (a)).

With this in mind, the presented registration algorithm first lets the user manually align an initial camera to the textured DEM. Then, an incremental SfM optimization is carried out similar to Snavely et al. [Snavely *et al.* 2006]. However, in contrast to them, the optimization is performed in an absolute coordinate system. This has the advantage that it is possible to relate to the textured DEM during the optimization. Instead of triangulating 3D points from image correspondences, they can now obtained from the DEM. As a result, estimates for correspondences with a small angle of separation are more robust. In addition to that, photos can be registered to screenshots of the textured DEM. By using the information about the (approximate) absolute camera parameters obtained in the optimization so far, the registration can be done much more efficiently than registering the photo to aerial imagery (see Figure 4.3.3 (b)).

The remainder of this chapter is organized as follows: First, related work is reviewed including structure from motion, image-based modeling and the registration of images to 3D models. Then, a robust algorithm for registering a set of uncalibrated images to a textured DEM is presented. Finally, results are shown and discussed.

4.2 Related Work

The work presented in this chapter aims at registering a set of images to a textured DEM, and is thus most closely related to methods that register images to 3D models. However, in addition to that, related work from structure from motion and image-based modeling, which are also relevant in the considered context, are reviewed as well.

4.2.1 Structure from Motion

The goal of structure from motion (SfM) techniques is to simultaneously recover the unknown 3D scene structure and camera parameters from a set of feature correspondences in images. The first effective structure from motion methods were developed in the 1980s. Longuet-Higgins [Longuet-Higgins 1981] introduced a still widely used two-frame relative orientation technique. Later, multi-frame structure from motion techniques, including factorization methods [Tomasi & Kanade 1992] as well as global optimization techniques [Spetsakis & Aloimonos 1991][Szeliski & Kang 1994][Oliensis 1999], were developed.

More recently, related techniques from photogrammetry, such as bundle adjustment [Triggs *et al.* 2000][Szeliski & Kang 1994], were introduced into the computer vision community. In situations where the camera calibration parameters are unknown, self-calibration techniques, which first estimate a projective reconstruction of the 3D world and then perform a metric upgrade, have proven to be successful [Pollefeys *et al.* 1999][Pollefeys & Gool 2002]. An iterative bundle adjustment was presented in [Brown & Lowe 2005], in which cameras are added to a bundle adjustment one by one. Snavely et al. [Snavely *et al.* 2006][Snavely *et al.* 2008] used a similar approach with several modifications to improve robustness. Their improvements include the initialization of new cameras using pose estimation in order to avoid local minima, a heuristic for the selection of the initial images for SfM, and a check if reconstructed points are well-conditioned before adding them into the optimization. They demonstrated the effectiveness of their approach by applying it to huge real-world photo sets found on Google and Flickr, including photos of different cameras, zoom levels, resolutions and illumination.

Schaffalitzky et al. [Schaffalitzky & Zisserman 2002] presented another related technique for the reconstruction of unordered image sets, focusing on efficiently matching feature points between images. Vergauwen and Van Gool proposed a similar approach [Vergauwen & Gool 2006] and are hosting a web-based reconstruction service addressing cultural heritage applications. In [Fitzgibbon & Zisserman 1998] and [Nistér 2000] a bottom-up approach is used instead, in which small subsets of images are matched to each other and then merged into a complete 3D reconstruction. Martinec et al. [Martinec & Pajdla 2007][Martinec & Pajdla 2006] reconstruct a scene by first registering all camera rotations and then translations using them, given pairwise Euclidean reconstructions for the cameras.

4.2.2 Image-based Modeling

During recent years, computer vision techniques, such as structure from motion, have been introduced into the computer graphics community under the name of image-based modeling. Image-based modeling is the process of creating three-dimensional models from a set of input images. A popular application of image-based modeling has been the creation of large scale architectural models. One of the first approaches was the semi-automatic Façade system [Debevec *et al.* 1996]. Given a sparse set of photographs, a basic geometric model of the architecture is recovered using an interactive photogrammetric modeling system. An automatic architecture reconstruction systems based on a Bayesian approach was presented in [Dick *et al.* 2004]. In the MIT City Scanning Project [Teller *et al.* 2003] thousands of calibrated images taken from an instrumented rig are used to construct a 3D model of the MIT campus. In the Stanford CityBlock Project [Roman *et al.* 2004] sideways-looking videos are taken from a vehicle driving down the street and are combined into a single multi-perspective image that summarizes one or more city blocks. The UrbanScape project [Akbarzadeh *et al.* 2006] presents an approach for fully automatic 3D reconstruction of urban scenes from video data. The videos are captured by an acquisition system consisting of eight cameras mounted on a vehicle. The 4D Cities project [Schindler *et al.* 2007] aims at building time-varying 3D models by temporally sorting a collection of input photos spanning many years.

4.2.3 Registration of Images to 3D Models

There has been a number of approaches to automated registering of imagery with a 3D model for texture mapping purposes. Stomas and Liu [Liu *et al.* 2006][Stamos & Allen 2002] developed an approach for texture mapping 2D images onto 3D range data. To identify camera parameters, they match features between the LIDAR data and the images using vanishing points and rectangular parallelepipeds on the building facades. The parameters are further refined using correspondences between the LIDAR data and the sparse point cloud generated from multiview geometry. Recently, Ding et al. [Ding *et al.* 2008] presented a fast, automated, camera pose recovery algorithm for texture mapping oblique aerial imagery onto 3D geometry models obtained via LIDAR. They take advantage of vanishing points and use a feature matching technique based on 2D corners associated with orthogonal 3D structural corners.

Hsu et al. [Hsu *et al.* 2000] presented an approach for video-based texture mapping. They use tracked features for inter-frame pose prediction, and refine the pose by aligning projected 3D model lines to those in images. Neumann et al. [Neumann *et al.* 2003] follow a similar idea by implementing an extended Kalman filter to perform interframe camera parameter tracking using point and line features. Both methods can lose track in situations with large pose prediction error due to occlusions. Zhao et al. [Zhao *et al.* 2005] instead use an iterative closest point algorithm to align a point cloud generated from video to that obtained from a range sensor.

Lee and Nevatia et al. [Lee *et al.* 2001] presented a method for integrating facade textures from ground view images into 3D building models. They use vanishing points and 3D-2D line matching to find single view camera poses. Recently, Hu et al. [Hu *et al.* 2006] presented a system for mapping ground and aerial-based imagery. Their system requires, however, human interactions in many places such as building contour extraction from aerial images and manual point correspondence to align aerial images to LIDAR data.

4.3 Registration of Images to a Textured DEM

In the presented algorithm, first an initial camera is manually aligned to the textured DEM prior to the SfM optimization. This way, it is possible to make use of the textured DEM during the SfM procedure. The textured DEM is then exploited in two ways during the optimization: First, instead of computing 3D estimates for corresponding features by triangulation, which can be unstable for small baselines, 3D estimates are obtained from the DEM.

Note that the main objective of the presented algorithm is to fit the photos as best as possible to the given approximate terrain geometry and not to estimate 3D scene geometry as best as possible. Second, the textured DEM is used for automatically establishing additional ground control points for the photos.

In the remainder of this chapter the different steps of the proposed algorithm are presented in detail. First, the feature detection and matching procedure is described. Then, the incremental bundle adjustment is presented, followed by a description of the algorithm to create additional ground control points.

4.3.1 Feature Extraction and Matching

The first step in the registration algorithm is to extract distinct features in each input image. The SIFT feature detector [Lowe 2004] is used for this since it has shown good invariance to image transformations. Alternatively, other feature detectors could be used (see [Mikolajczyk & Schmid 2005] for a comparison of various feature detectors). In addition to the location of a feature, SIFT also provides a local descriptor for each feature as a 128-dimensional descriptor vector. A typical image contains up to several thousand SIFT features (see Figure 4.3).

Next, corresponding features between each image pair are determined. To match features between two images I and J, first, a kd-tree is built from the feature descriptors in J. Then, for each feature in I the nearest neighbor in J is determined using the kd-tree. Since kd-tree searches can be slow for high dimensional spaces, the approximate nearest neighbors (ANN) kd-tree pack-

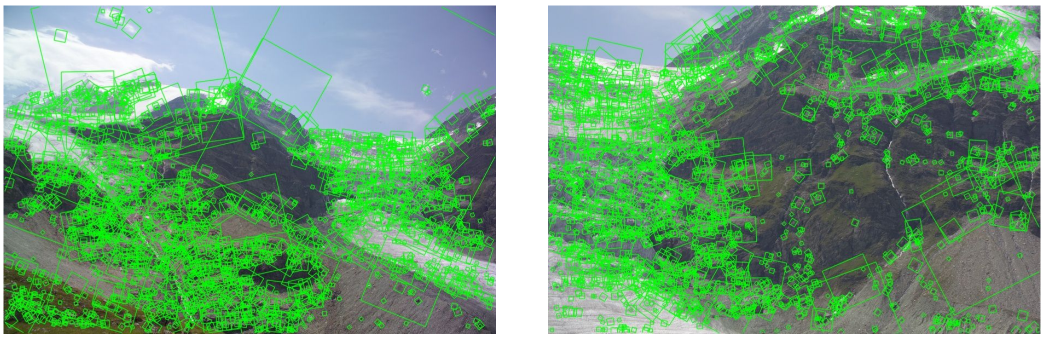


Figure 4.3: Feature Detection. SIFT features extracted in the two images are shown as square patches. The squares are scaled and rotated to reflect the scale and orientation of the detected features.

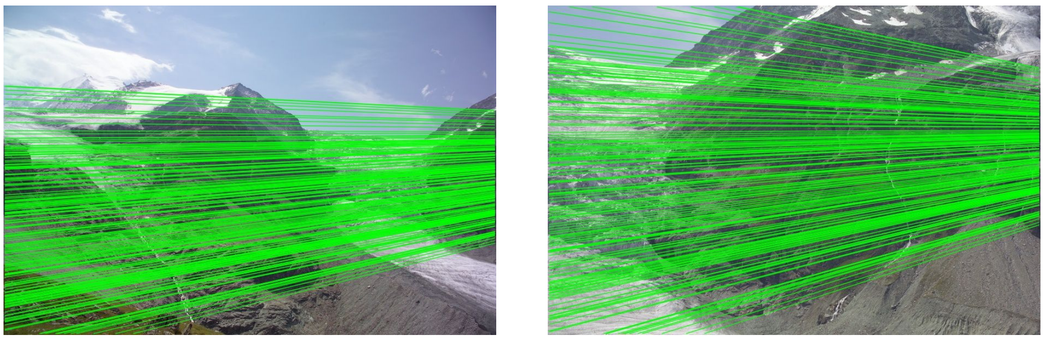


Figure 4.4: Epipolar Lines. The fundamental matrix is computed from corresponding features using the normalized 8-point algorithm inside a RANSAC procedure. The epipolar line of each detected feature is visualized in the respective other image.

age of Arya et al. [Arya *et al.* 1998] is used to further speed up the nearest neighbor search. Rather than classifying false matches by thresholding the distance to the nearest neighbor, the closest-to-next-closest matching scheme proposed by Lowe [Lowe 2004] is applied (see section A.6). In this matching strategy the ratio of descriptor distances to the nearest and second nearest neighbor is thresholded (a threshold of 0.6 is used[1]). If more than one feature in I matches the same feature in J, all involved matches are removed as some of them must be invalid.

After corresponding features for an image pair have been determined, the fundamental matrix for the pair is robustly estimated using RANSAC [Fischler & Bolles 1981]. In each RANSAC iteration, a candidate fundamental matrix using the normalized 8-point algorithm [Hartley 1997] is computed. The RANSAC outlier threshold is set to 6 % of $\max(w(I), h(I))$, where $w(\cdot)$ and $h(\cdot)$ are the width and height of the image, respectively. Matches that are not compatible with the computed fundamental matrix, i.e., whose distance to the corresponding epipolar line (see Figure 4.4) exceeds a given threshold, are removed from the optimization (see Figure 4.5). The obtained fundamental matrix is then refined by applying the Levenberg-Marquardt algorithm minimizing errors subject to the found inliers. If the number of remaining matches is less than 20, all matches are removed from consideration.

Once a set of geometrically consistent correspondences has been determined between each image pair, they are organized into tracks. A track is a connected set of corresponding features across multiple images (i.e., all features of a track are image projections of the same point in world space, and

[1]Most thresholds used in the following are set according to [Snavely *et al.* 2008]

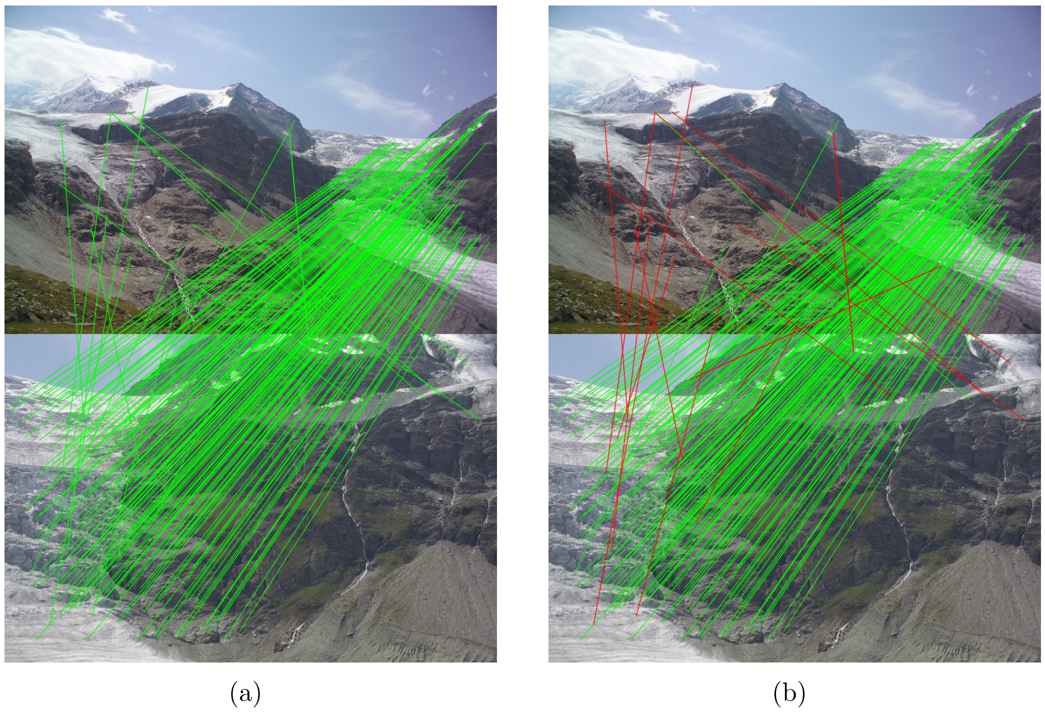


Figure 4.5: Outlier Removal. (a) Visualization of the detected correspondences during feature matching. (b) The computed epipolar geometry is used to remove outliers (red). Note that not all false matches can usually be detected and removed.

hence are in pairwise correspondence). Correspondences that were not found directly during feature matching are added accordingly as shown in Figure 4.6. If a track contains more than one feature from the same image, it is considered inconsistent and removed from the optimization. The determined feature tracks are then organized in an image connectivity graph, in which each image is a node and an edge exists between each pair of images with matching features. Typically, the graph consists of several connected components with one or more large components containing the vast majority of images and several smaller ones consisting of the remaining images that could not be matched. The following sections describe how the images of such a connected component of images can be robustly georegistered. If more than one connected component should be registered, the whole process has to be repeated for each of them accordingly.

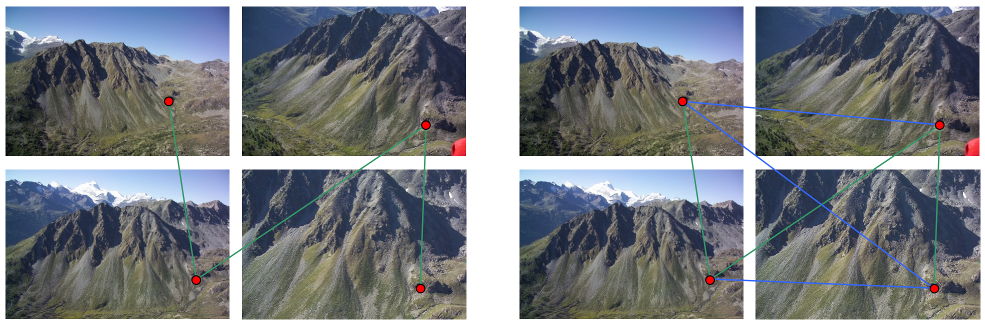


Figure 4.6: Feature Tracks. Matching features (green) are organized into tracks (left). All features in a track are image projections of the same world point and are thus in pairwise correspondence. Therefore, all feature correspondences that have not been found during the matching phase (blue) are added to the list of correspondences (right).

4.3.2 Incremental Bundle Adjustment

The cameras are parameterized using a seven-parameter model. Following the common assumptions of square pixels, zero skew, and that the center of projection is fixed and coincident with the image center, the remaining free parameters are the 3D orientation (three parameters), the camera center (three parameters), and the focal length (one parameter). For a detailed definition of camera models see section A.2.

In order to be able to exploit the textured DEM in the following incremental structure from motion optimization, first an initial camera is manually georeferenced. To this end, the user has to specify a few correspondences between the corresponding photo and the textured DEM (see Figure 4.7). To represent a good starting point for the optimization, the photo that is initially estimated should have a large number of correspondences. Therefore, the photo with the largest number of matches is proposed to the user to be initially matched (optionally, an arbitrary photo can be chosen by the user). From the user-specified ground control points the camera parameters are then estimated using the Gold Standard algorithm for camera estimation (see section A.3) inside a binned RANSAC procedure (see section A.8). For the RANSAC step, an outlier threshold of 0.4 % of $\max(w(I), h(I))$ and a sample of four features taken from four adaptive bins (one feature from each bin) is used.

Once the initial camera has been estimated, the remaining cameras are added one by one into the optimization. Among the yet uninitialized cameras always the one observing the largest number of tracks already in the opti-

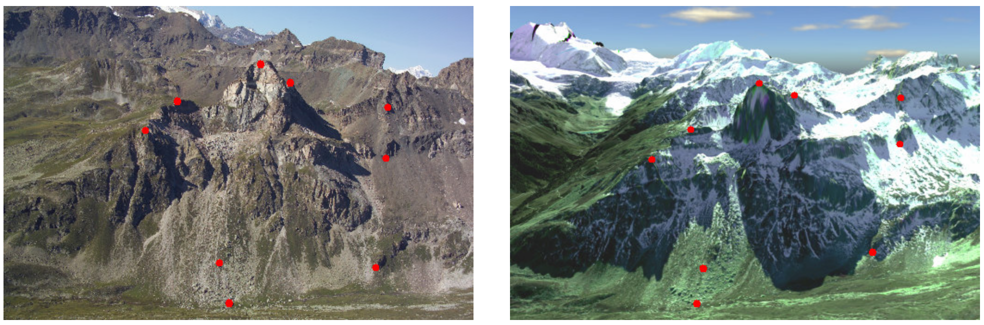


Figure 4.7: Manual Matching. In the manual matching the user has to specify corresponding 2D points in a photo (left) and 3D points on the textured DEM (right). From this correspondences an initial georegistered camera is estimated that is used as starting point for the optimization.

mization is added next. Then, the camera's 2D feature points are assigned corresponding 3D points transferred from matching features in other images that already have a 3D estimate (see Figure 4.8). This is because if two features correspond, then they originate from the same 3D world point. If sufficiently enough 2D-3D correspondences (>20) are available for the camera, the Gold Standard algorithm is used again inside a binned RANSAC procedure to estimate the camera parameters.

Once the camera matrix has been estimated, the yet uninitialized 2D feature points in the respective image are assigned corresponding 3D world points taken from the DEM. The 3D points are obtained by rendering the DEM from the estimated camera position and reading back the respective depth values from the z-buffer. Note that the 3D world points can only be obtained from the DEM because the optimization is performed in an absolute coordinate frame with respect to the DEM.

Next, a global bundle adjustment is performed using a sparse bundle adjustment library [Lourakis & Argyros 2004]. The bundle adjustment is configured such that only camera parameters (i.e., motion) are refined while the 3D points (i.e., structure) remain unchanged. The optimization is restricted to motion only because the main objective is to register the images as best as possible to the given geometry in the first place and not to improve geometric accuracy. In addition to that, triangulating a 3D point does not provide a well-conditioned estimates if there is only a small angle of separation between the rays. Although it is possible to detect and exclude such tracks from the optimization, it would prevent a successful registration of many images due to an insufficient number of remaining matches.

To obtain the 3D estimate for a track from the DEM, the DEM is ren-

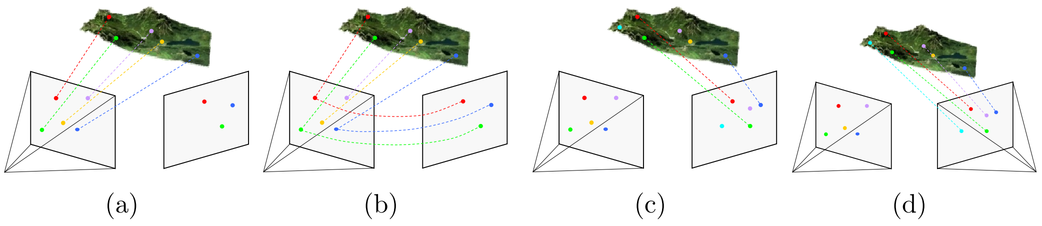


Figure 4.8: Incremental Bundle Adjustment. (a) Detected SIFT features in an already estimated camera are assigned corresponding 3D points obtained from the textured DEM. (b) A new camera is added into the optimization. Matching features in other images that already have a 3D estimate are determined and transferred. (c) Using the transferred 3D estimates the new camera is estimated via the Gold Standard algorithm. (d) The yet uninitialized features of the new camera are assigned 3D points from the DEM. Finally, a global bundle adjustment is performed.

dered from all involved camera positions. Then, a local search is performed around the position of each feature of the considered track in the corresponding depth map. The depth value that minimizes the mean reprojection error over all features of the track is taken as the new 3D estimate. Since the camera parameters change during each iteration, the 3D estimates of the tracks are updated after each run. Additionally, outlier tracks are removed after each iteration if their reprojection error is above a threshold of 0.8 % of $\max(w(I), h(I))$. This procedure is repeated, one camera at a time, until all cameras have been processed or no remaining camera observes enough 3D points (>20) to be robustly reconstructed.

4.3.3 Adding Ground Control Points

So far, the only direct correspondences established between the photos and the textured DEM are the ground control points (i.e., 2D-3D correspondences between the images and the textured DEM) specified by the user during the manual registration of the first camera. The remaining cameras are only indirectly constrained by these ground control points through 2D-2D feature correspondences in the images. Due to imperfect initializations, imprecise feature locations, and round-off errors during the incremental bundle adjustment, errors may accumulate during each iteration and introduce drift. Especially cameras for that the level of indirections is high with respect to the initial camera are susceptible to drift. To account for that, additional ground control points are determined in each iteration for the newly added photo (see Figure 4.9). To obtain ground control points for a photo, the

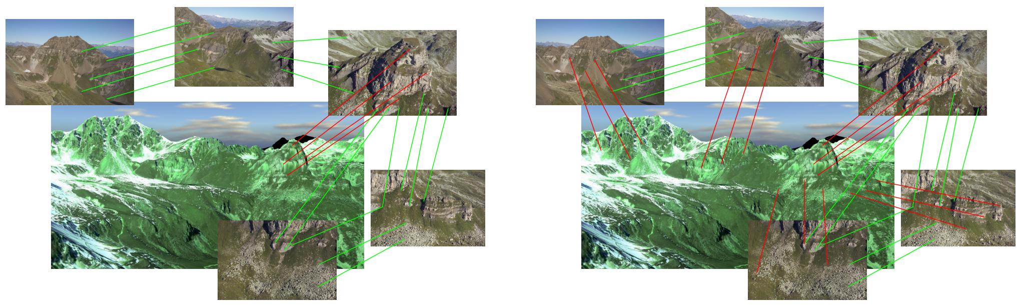


Figure 4.9: Adding Ground Control Points. *Left:* Only the manually registered image is directly linked to the textured DEM via ground control points (red). The remaining images are only indirectly connected through image-to-image correspondences (green) and thus errors might accumulate. *Right:* If a sufficiently large number of consistent 2D-3D correspondences between a photo and the textured DEM are found, they are added as ground control points (red) to avoid error accumulation and drift.

textured DEM is rendered from the estimated camera's point of view in the resolution of the photo. Then, SIFT features are extracted from the rendered screenshot and matched with the features in the corresponding photo that were detected in the preceding feature extraction stage. Each 2D feature in the photo, for that a matching feature in the screenshot exists, is then assigned the 3D point of the corresponding feature in the screenshot derived from the respective depth map.

However, the screenshot and aerial imagery typically differ drastically with respect to resolution, lighting conditions, seasonal changes, etc., typically even more than the photos among each other. Therefore, the matching strategy that was used to find corresponding features in the photos often results in too few matches to robustly estimate a camera from them (see Figure 4.10 (a)). Although lowering the applied matching threshold produces slightly more matches, the percentage of outliers is at the same time drastically increased. Even with robust estimation methods like RANSAC, the amount of outliers is then too large to robustly estimate camera parameters. However, in contrast to the feature matching performed between the input photos at the beginning, there now is an estimate of the camera parameters available. Consequently, the photo and the rendered image can be assumed to be already fairly good aligned. This observation can be used to reduce the set of potential matching candidates significantly and hence drastically increase matching performance (see Figure 4.10 (b)).

Given a feature in the photo, it is not compared to all features detected in the rendered screenshot but only to those features that have a similar

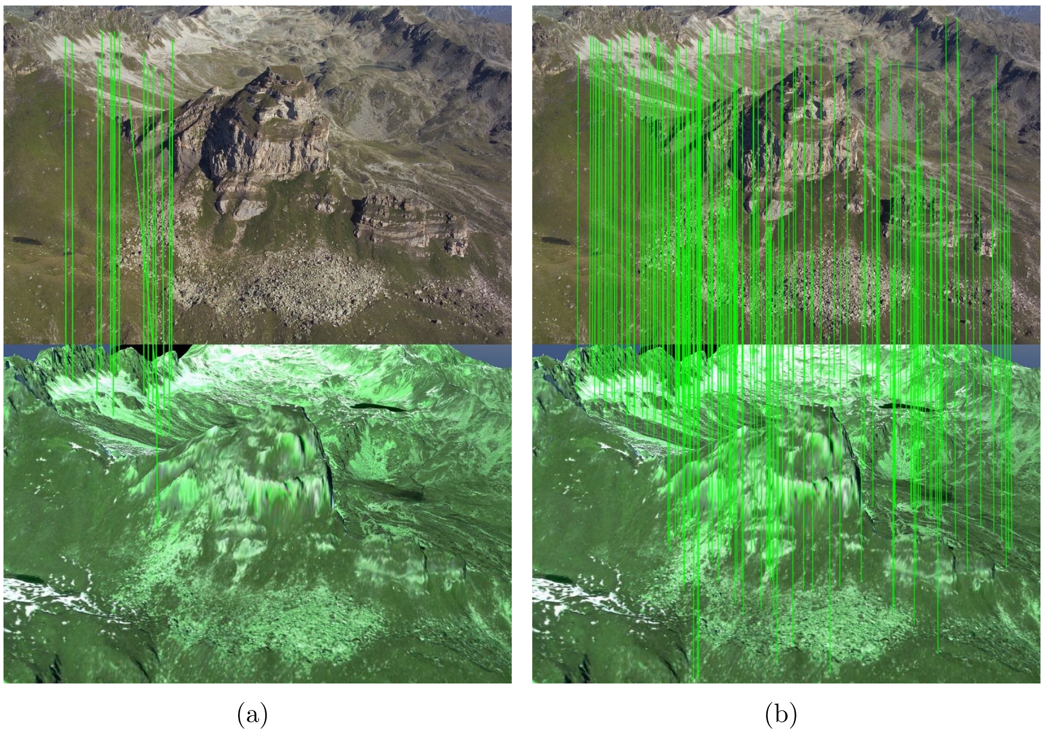


Figure 4.10: Comparison of matching strategies. (a) Standard matching results in 16 features only. (b) Applying the presented matching method results in more than 100 matches.

position p and whose descriptors coincide with respect to scale s and orientation o. Let $\mathcal{F}_p$ and $\mathcal{F}_s$ be the set of features found in the photo and the screenshot, respectively. Then, the set of possible matching candidates for a feature $f \in \mathcal{F}_p$ is defined as

$$\mathcal{C}_{\mathcal{F}_p}(f) = \{g \in \mathcal{F}_s \mid \|p(f) - p(g)\| < t_p \ \wedge \ o(f,g) < t_o \ \wedge \ s(f,g) < t_s\},$$

where

$$o(f,g) = \frac{\min(o(f), o(g))}{\max(o(f), o(g))} \quad \text{and} \quad s(f,g) = \frac{\min(s(f), s(g))}{\max(s(f), s(g))}$$

measure the difference in orientation and scale of a feature pair (f,g). In the current implementation the thresholds are $t_p = 0.05 \max(w(I), h(I))$ and $t_o = t_s = 0.8$. Note that the rotation and scale invariance of the SIFT descriptors is deliberately removed. Of course, other descriptors that are not invariant under rotation and scaling could be detected in the photo and the

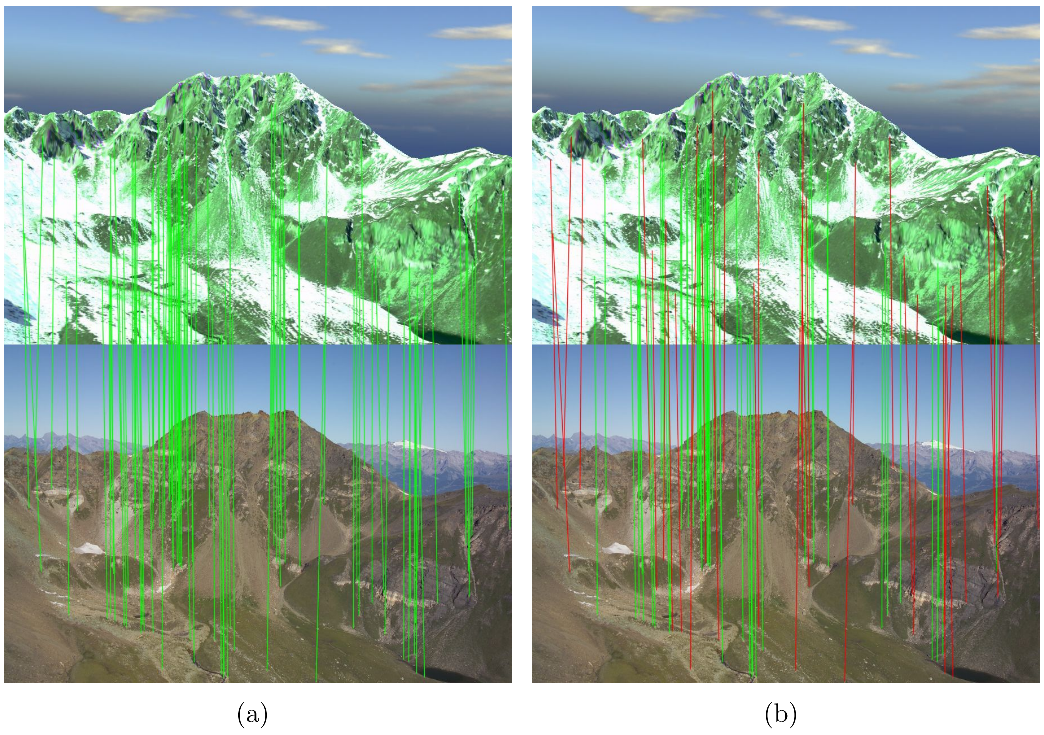


Figure 4.11: Outlier Removal. (a) Ground control points detected by matching features in the input photo and the rendered screenshot. (b) The remaining features (green) and the outliers (red) after application of the depth weighted disparity gradient technique and subsequent camera estimation via RANSAC.

screenshot and then matched. However, since SIFT features have already been determined in all input images, a repeated feature extraction on all input images is avoided this way.

From the set of possible matching candidates $\mathcal{C}_{\mathcal{F}_p}$, then the feature with the closest descriptor d is taken and thus the resulting set of matches for the photo is

$$\mathcal{M}_{\mathcal{F}_p} = \left\{ (f,g) \in \mathcal{F}_p \times \mathcal{C}_{\mathcal{F}_p} \;\middle|\; g = \underset{g_j \in \mathcal{C}_{\mathcal{F}_p}}{\arg\min} \|d(f) - d(g_j)\| \right\}.$$

The procedure is repeated for the features in the screenshot to obtain $\mathcal{M}_{\mathcal{F}_s}$. The matches obtained in this bidirectional matching are not symmetric. To enforce symmetry, the two sets are aggregated to

$$\mathcal{M} = \mathcal{M}_{\mathcal{F}_p} \cap \mathcal{M}_{\mathcal{F}_s},$$

keeping only those matches that are contained in both matching sets (assuming matches to be unordered, i.e., $(f, g) = (g, f)$).

Finally, outliers in the resulting set of matches $\mathcal{M}$ are removed by applying a novel depth weighted, iterative disparity gradient technique (see Figure 4.11). The idea behind the depth weighting is to use the depth information available for each feature as an additional indicator to what extent the disparities of two considered feature pairs have to coincide. While feature pairs with similar depth values should have similar disparities, those with highly varying depths are allowed to have larger differences in their disparity. The depth weighted disparity gradient for two pairs of corresponding features m and m' is defined as

$$dg_w(m, m') = w(m, m')dg(m, m'),$$

where $dg(\cdot)$ is the standard disparity gradient (see section A.7) and

$$w(m, m') = 1 - \frac{|z(m) - z(m')|}{|z_{max} - z_{min}|^4}.$$

$z(\cdot)$ is the depth of a match defined by the depth value of the respective feature in the screenshot, and z_{max} and z_{min} define the maximum and minimum depth over all matches. Using the depth weighted disparity gradient dg_w, the disparity gradient sum for a match m is defined as

$$dg_{sum}(m) = \frac{\sum_{m'} dg_w(m, m')}{\sum_{m'} w(m, m')}, \quad m' \in \mathcal{M}, \; m \neq m'.$$

The disparity gradient sum is calculated for each match in $\mathcal{M}$. Then, the n matches with the highest dg_{sum} are removed (n is set to 50 % of the currently valid matches). This process is iterated until the disparity gradient sum of all remaining matches is below a threshold (0.1 is used here). If a sufficent large number of matches is remaining (>32), they are used to estimate a camera matrix using the Gold Standard algorithm inside a binned RANSAC procedure with 4 adaptive bins. Then, if a sufficiently large number of inliers is found (>16), they are added as ground control points to the optimization and the estimated camera matrix is assigned to the current camera.

Although not every image can be registered directly to the terrain in this manner, the remaining images are at least indirectly constrained by the found ground control points. As a consequence, potential drift is reduced and robustness of the optimization is increased. The method for adding ground control points is outlined in Algorithm 1. The complete registration procedure is summarized in Algorithm 2.

Algorithm 1 AddGroundControlPoints($C, \mathcal{F}_I$)

```
Input:
  current estimate of camera $C$
  features $\mathcal{F}_I$ detected in image $I$

Output:
  set of ground control points $\mathcal{G}$
  refined estimate for $C$

$S \leftarrow$ render screenshot from camera $C$
$\mathcal{F}_S \leftarrow$ extract features from screenshot $S$
$\mathcal{M}_S \leftarrow$ match features of $\mathcal{F}_S$ to $\mathcal{F}_I$
$\mathcal{M}_I \leftarrow$ match features of $\mathcal{F}_I$ to $\mathcal{F}_S$
$\mathcal{M} \leftarrow$ aggregate matches $\mathcal{M}_S \cap \mathcal{M}_I$
$\mathcal{M}' \leftarrow$ use depth weighted disparity gradient to remove outliers from $\mathcal{M}$
if $|\mathcal{M}'| > 32$ then
  $C' \leftarrow$ estimate camera matrix from $\mathcal{M}'$
  $\mathcal{G}' \leftarrow$ remaining inliers after estimation of $C'$
  if $|G'| > 16$ then
    $C \leftarrow C'$
    $\mathcal{G} \leftarrow \mathcal{G}'$
  end if
end if
```

4.4 Results and Discussion

The presented registration approach is evaluated using the HRSC dataset of Turtmann valley. The respective photo set contains more than hundred images taken by several people with different cameras. The majority of the photos were acquired in summer 2006 within several weeks. Some of the photos were taken from the ground others during a helicopter overflight. Although EXIF tags for some of the images were available, they were not used in the SfM optimization.

The registration algorithm does not require any user-defined marker points on-site, which is essential in high alpine environments where many places are difficult to access. Instead, camera parameters are estimated from the photos and the textured DEM alone using computer vision methods. However, if additional marker points are available, they can naturally and easily be included in the registration process.

In general, the registration algorithm is not able to reconstruct all input photos. The main reason for this is that the input photo sets typically form

Algorithm 2 Registration

```
Input:
  set of images $\mathcal{I} = \{I_1, \dots, I_n\}$

Output:
  set of camera matrices $\mathcal{C} = \{\mathbf{C}_1, \dots, \mathbf{C}_n\}$
  set of features $\mathcal{F}_1, \dots, \mathcal{F}_n$ for each image
  set of feature tracks $\mathcal{T}$

for $i = 1, \dots, |\mathcal{I}|$ do
  $\mathcal{F}_i \leftarrow$ extract features from $I_i$
end for

for $i = 1, \dots, |\mathcal{I}|$ do
  $kd(\mathcal{F}_i) \leftarrow$ build kd-tree from $\mathcal{F}_i$
  for $j = i + 1, \dots, |\mathcal{I}|$ do
    $\mathcal{M}_{ij} \leftarrow$ match features $kd(\mathcal{F}_i)$ and $\mathcal{F}_j$
    $\mathcal{M}_{ij} \rightarrow$ remove outliers using epipolar constraint
  end for
end for
$\mathcal{T} \leftarrow$ find all tracks in $\mathcal{M}_{ij},\ \forall i, j$

$I_{init} \leftarrow$ select initial image
$C_{init} \leftarrow$ estimate camera from ground control points defined in $I_{init}$
initialize tracks in $C_{init}$

for $i = 1, \dots, |\mathcal{I}|$ do
  $I' \leftarrow$ select next image
  if $|\text{tracks}(I')| > 20$ then
    $C' \leftarrow$ estimate camera from tracks observed in $I'$
    $\mathcal{T} \leftarrow$ addGroundControlPoints($C', \mathcal{F}_i$)
    $\mathcal{C} \leftarrow$ perform bundle adjustment
    $\mathcal{T} \leftarrow$ update 3D estimates
    $\mathcal{T} \rightarrow$ remove outliers
  else
    break
  end if
end for
```

separate connected components after feature detection and matching, which are too weakly connected to be reliably reconstructed. To register all of them, the connected components have to be processed separately, which requires a manual initialization by the user to provide good initial estimates for each of them. For instance, the images taken of the Meidhorn (mountain in Turtmann valley) form two clusters. These clusters correspond to the north and the south side of the Meidhorn. However, only a few photos were taken

collection	#img.	#reg.	#reg. gcp	#points	#gcp	runtime	error	#bundler
debris field	17	17	14	7397	802	401	1.5	15
glacier	79	75	38	22208	1394	4896	1.64	38
valley slope	40	38	37	9684	1036	758	1.53	37

Table 4.1: Results. *Collection*: the name of the set, *#img*: the number of images in the set, *#reg.*: the number of images registered, *#reg. gcp*: the number of images registered via ground control points, *#points*: the number of points used in the optimization (including ground control points), *#gcp*: the number of ground control points used in the optimization, *runtime F+M*: the approximate time for feature detection matching and consistency checks, *runtime BA*: the approximate total time for the bundle adjustment, *error*: the mean reprojection error in pixels after optimization, *#bundler*: the number of images registered with bundler (only relative registration of photos)

from intermediate angles from the east from the opposite side of the valley. Because of this, too few SIFT features have been detected to connect the two clusters. More images would be needed to establish enough connections to bridge the gap between the clusters. In practice, however, there are typically only very few large connected components after feature detection and matching containing the majority of the photos.

Figure 4.12 depicts examples of the obtained registration results that demonstrate the accuracy of the presented algorithm. Each mosaic is created from a photo and the corresponding screenshot taken from the estimated camera position. Despite large differences between the photos and the textured DEM with respect to illumination, resolution, etc., an accurate registration for almost all images could be obtained. Figure 4.13 shows examples from the registered photo sets listed in Table 4.1. The first column shows the frusta of the estimated cameras obtained in the registration. The images in the second column show the terrain textured with the photo corresponding to the camera drawn in red using projective texture mapping. Compared to the original dataset the enhanced representation has drastically increased visual quality and information content, particularly at steep slopes.

More information on the registration results of the different connected components (including the number of input photos, the number of registered photos, and the average reprojection error) are listed in Table 4.1. The running times reported in this table refer to the incremental sparse bundle adjustment only (including the detection of ground control points), which is the dominating factor in the registration procedure. Typically, more than 50 % of the images can be directly matched to the textured DEM via ground control points. Overviews (i.e., photos that show large parts of the terrain)

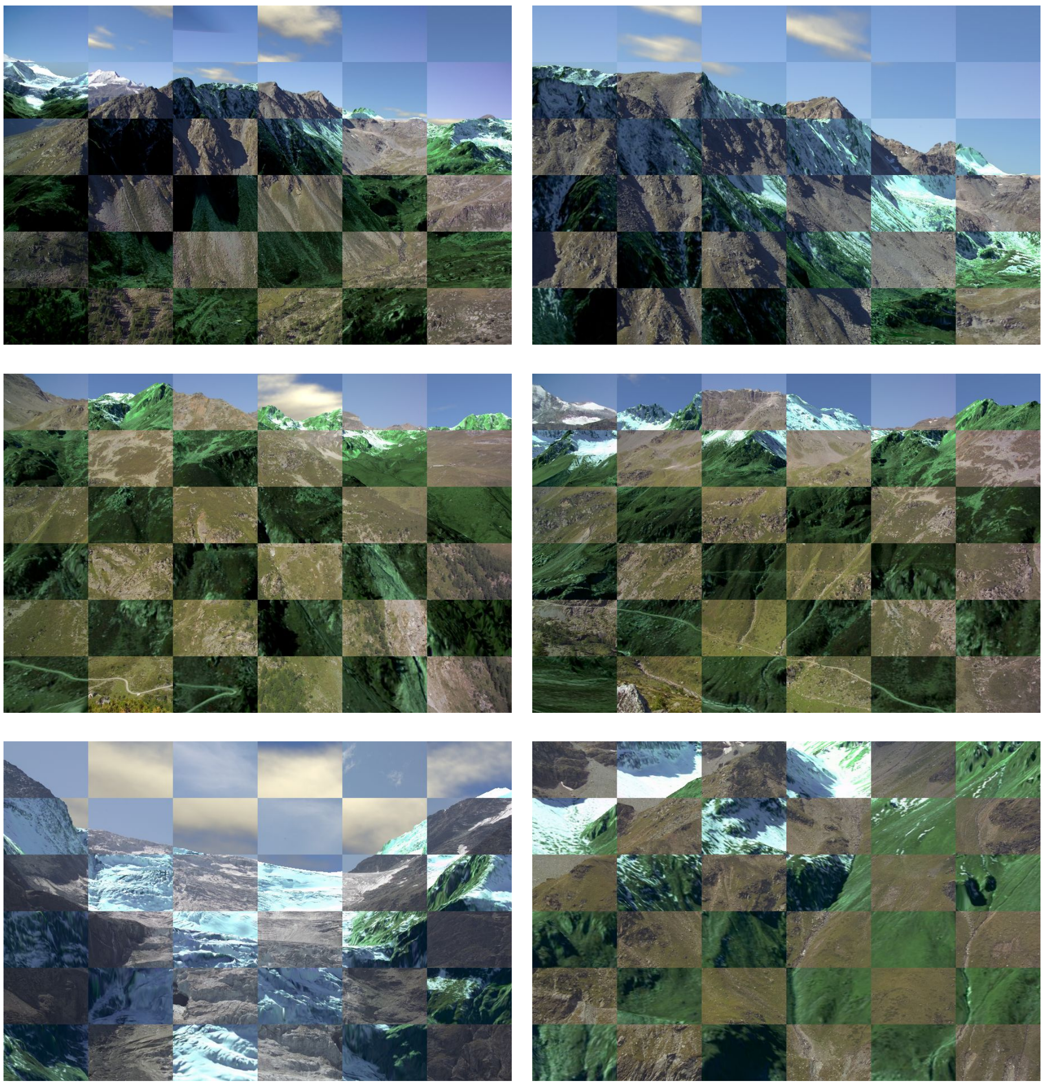


Figure 4.12: Registration Examples. The images show mosaics created from the photos and corresponding screenshots which were rendered from the position of the estimated camera.

can often be matched directly to the textured DEM. The reason for this is that they are likely to contain at least some regions that are rather flat and highly textured. Such regions are most promising for matching because flat regions are quite well captured in the aerial imagery, and highly textured regions produce many distinct features needed for matching. Close-up views, by contrast, especially of steep slopes, are often difficult to match directly

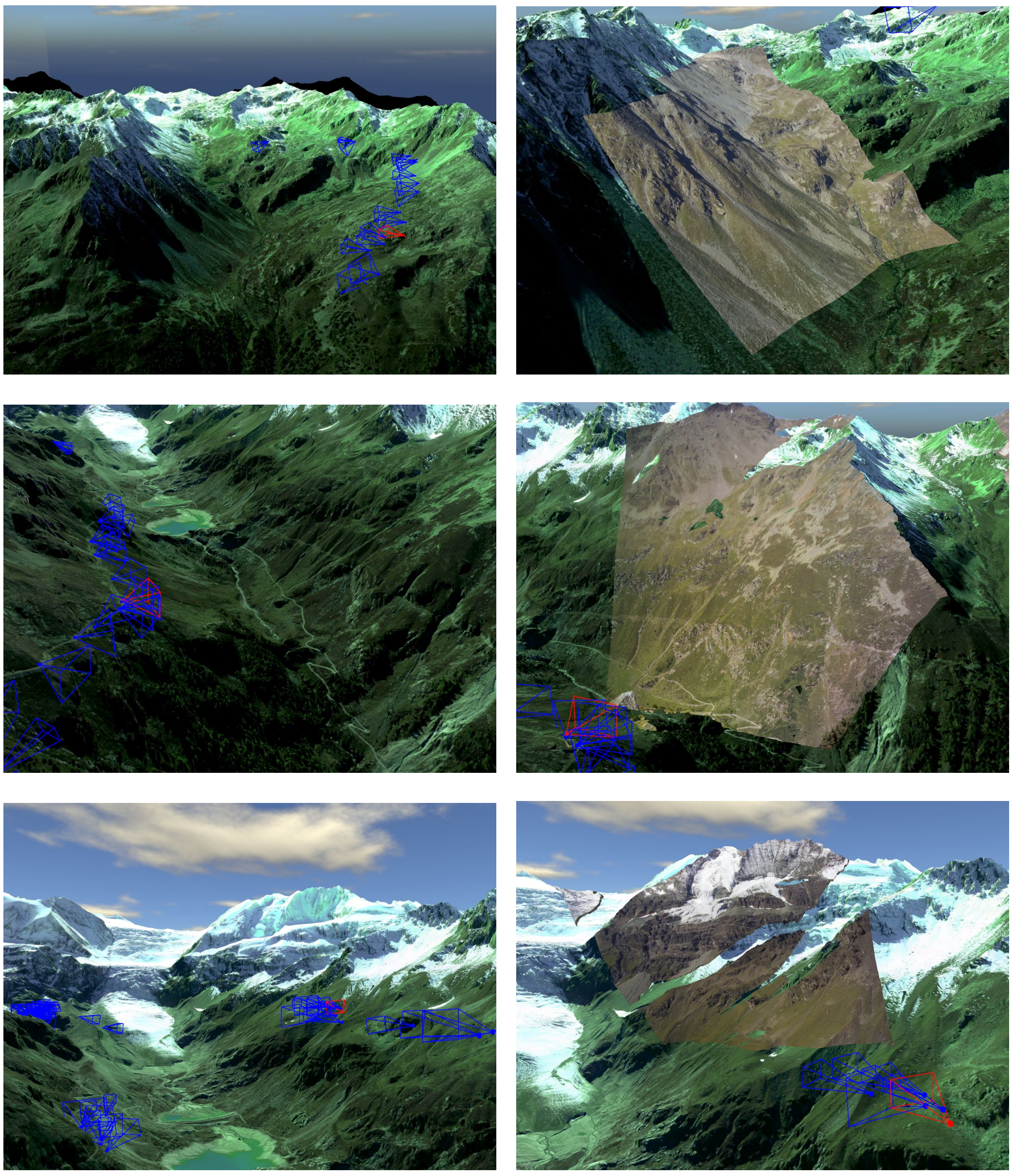


Figure 4.13: Registered Image Sets. The images in the left column show the frustra of the estimated cameras obtained in the registration procedure. In the images in the right column the terrain is projectively textured with the photo associated with the camera drawn in red.

to the textured DEM. This is due to large differences in texture resolution between the photo and the respective screenshot, which prohibits a detection of reliable correspondences. Moreover, photos containing large uniform areas, such as shadows or snow, are also difficult to match.

To assess the presented method, the photo sets were also registered using the Bundler[2] software package by Snavely. However, only relative camera parameters were estimated while no postprocessing step to register them to the textured DEM was carried out. For the used photo sets of Turtmann valley, the presented methods was able to register slightly more images from two of them while significantly more photos could be registered from the third. The reason for that is that the latter dataset contains many photos taken from positions very close to each other. In such situations, the presented method particularly benefits from its use of the textured DEM and the additional ground control points during the optimization.

The focus in the presented registration approach has been to align the photos as best as possible to a given terrain dataset. However, the ability to compute accurate camera parameters opens the door for techniques that compute dense surface shape models, such as multi-view stereo [Goesele *et al.* 2007]. For future work it might be interesting to investigate to what extend the geometric resolution of a DEM could be improved with such methods. Another interesting direction for future work would be the acquisition of time varying phenomena through repeat photography of the same site, with a time lag between the different images. For example, by taking photos of the Turtmann glacier at different times and registering them to the dataset, it would be possible to capture its changes over time without requiring an elaborate georeferencing of the photos during acquisition.

[2]http://phototour.cs.washington.edu/bundler/

CHAPTER 5

Compositing

5.1 Motivation

In the previous chapter an algorithm for registering sets of uncalibrated photos to a textured digital elevation model (DEM) was presented. One motivation for this was to use the photos to improve the visual quality of the textured DEM. However, the registration of the photos presents only the first step towards this goal. Given a set of registered images, the next step is to stitch them together on the terrain surface in order to create a visually pleasing and plausible composite. However, a simple combination of regions from the images produces visible artificial edges at the transitions between the individual images due to differences in camera gain, scene illumination or geometrical misalignments (see Figure 5.10). The main challenge is therefore to come up with a compositing algorithm that avoids such artifacts.

Typically, compositing involves two main steps: First, a suitable *compositing surface* along with a corresponding parameterization is selected that defines the domain where the blending takes place and how the blended result is represented. Second, an appropriate *blending* algorithm has to be applied that should avoid visible seams and minimize noticeable blur and ghosting artifacts.

The choice of a compositing domain is highly influenced by the available amount of geometry of the scene. In panorama stitching applications, for example, where no geometry is available at all, the registered images are typically projected onto some kind of proxy geometry. A simple and often

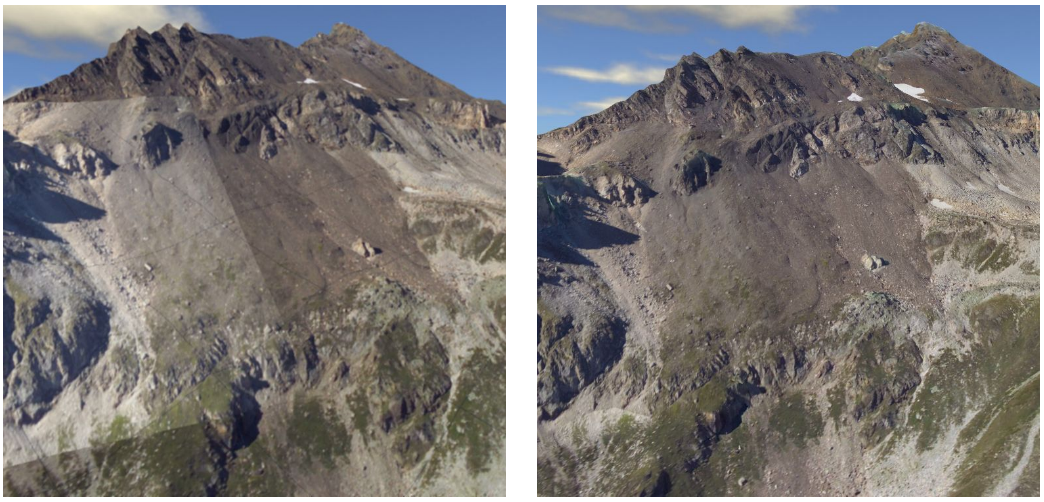


Figure 5.1: Motivation. The left image shows photos mapped onto the DEM without color correction and blending. As a consequence, the terrain surface shows a mosaic appearance and transitions between the individual images are visible. The right image shows the same scene but this time using the presented compositing approach. The terrain surface looks consistent without visible seams between the images.

applied approach is to select one of the input images as reference and then to warp the other images into the plane defined by the reference image. This approach produces reasonable results if only a few images are stitched together and the field of view is not too large. For wider fields of view, however, a flat representation introduces large distortions excessively stretching pixels near the border. Therefore, when compositing large panoramas typically cylindrical [Szeliski & Kang 1994][Chen 1995] or spherical [Szeliski & Shum 1997] projections are used. In fact, any representation used in environment mapping, as for example cube maps [Greene 1986][Szeliski & Shum 1997], can be used.

However, if an exact or approximate surface geometry is given on which the images are to be combined, such as a DEM, the use of a simple proxy geometry as a compositing surface has several shortcomings. First, for complex objects the use of a simple proxy geometry does not provide a good approximation of the object and may introduce significant distortion. Second, visibility of all triangles with respect to the proxy can not be guaranteed. And, third, continuity in the texture domain is not enforced everywhere, i.e., neighboring surface elements are not necessarily textured from neighbouring regions in the texture map (e.g., neighboring triangles may be textured using different sides of a cube map), which is important for a good blending.

With this in mind, the presented approach performs a reparameterization of the terrain surface instead to avoid the aforementioned problems. The reparameterization enfolds the terrain mesh into a planar domain (texture space) thereby minimizing the arising distortion with respect to area and angles of the triangles of the mesh. Furthermore, visibility of each surface element is guaranteed and continuity in the texture representation is ensured.

Once a suitable compositing domain has been selected, the images have to be blended in it. The simplest way to blend images is to perform an *averaging* of the values at each pixel. Unfortunately, simple averaging is usually not able to avoid blur and ghosting. However, it can be improved by weighting pixels near the center of the image more heavily than pixels close to the border. Similarly, if an image contains cutout regions, pixels close to the cutouts are down-weighted. For this purpose, weights based on a distance map are often used. Weighted averaging based on a distance map, often called *feathering* [Szeliski & Shum 1997][Chen & Klette 1999][Uyttendaele *et al.* 2001], produces reasonable results when combining images with different exposure but blurring and ghosting can still pose problems. In practice, however, it is difficult to achieve a pleasing balance between smoothing out low-frequency exposure variations on the one hand and preserving high-frequency details on the other hand.

To avoid the aforementioned issues, a two step procedure is presented: First, a color correction is applied that removes large scale color and lightness shifts in the images. Since neighboring parts in the images do not have to correspond to neighboring parts on the terrain surface, the application of color correction methods that work on whole images can not be applied here. Instead, corresponding regions (i.e, regions that show the same part of the terrain surface) in the images have to be detected and matched separately. However, to achieve a globally optimal solution, the different corresponding regions are not considered independently. Therefore, a simultaneous matching of their color distribution is carried out.

In the second step, a weighted multi-band blending approach is applied that produces smooth transitions between the images on the terrain surface while at the same time high-frequency details in the transition regions of the images are preserved. The blending is carried out in the texture domain induced by the reparameterization. Only because of the reparameterization an effective application of the multi-band blending becomes possible as the texture space represents a continuous planar representation of the terrain surface.

The remainder of this section is organized as follows: After a review of related work in the next chapter, the algorithm for blending a set of georegistered photos on a DEM is presented. Finally, results are shown and discussed.

5.2 Related Work

As the goal of the presented method is to use photos to texture a given digital elevation model, it falls into the field of *image-based rendering.* Image-based rendering techniques synthesize new views of a scene from a set of images, which are typically photographs of a static scene. Previous image-based rendering techniques can be classified by the amount of geometric information being used: no geometry, implicit geometry (i.e., correspondences), and explicit geometry (either with approximate or accurate geometry). The considered texturing problem clearly falls into the latter category.

In the following, work dealing with texturing a given 3D model with photos is focused. In addition to that, methods for blending and color correction that are relevant within the considered context are reviewed as well.

5.2.1 Texture Mapping 3D Models

Ignoring view-dependent effects, texture stitching methods can be used to create view-independent textures from a set of input images. This has the advantage that sophisticated image stitching algorithms can be applied in a preprocessing step. A common approach is to apply triangle-based mosaicing schemes and use feathering to mask seams afterwards. In general these techniques rely on a regular triangular mesh model and each triangle is assigned to the best camera by considering viewing angle and visibility. This kind of technique is quite effective but the transition width between regions textured from different images is fixed by the size and shape of the triangles. Consequently, if the triangles are too small, the seams between regions will only be slightly blurred and still visible. If, by contrast, the triangles are large, high-frequency details are blurred away. This can also cause ghosting due to misregistration of the cameras and inaccuracies in the surface model.

Rocchini et al. [Rocchini *et al.* 2004] stitch textures on a 3D object by building a patchwork of image subsections such that all of the object surface is covered and adjacent image subsections join smoothly on the object surface. They address ghosting by performing a local triangle-based registration at the region boundaries. However, their approach is limited by a simple linear model for local registration, which in practice only works for a small transition zone. Lensch et al. [Lensch *et al.* 2001] determine for each triangle the view that provides the best available texture. Then, they blend the textures across border triangles of regions assigned to different views. A disadvantage of their method and that of Rocchini et al. is that exposure and chromaticity differences in non-overlapping areas are not handled at all.

An alternative to mosaicing schemes is to use per-pixel weighted filtering over the whole mesh. Smooth weight functions are used to assure continuous transitions and to avoid visible seams. Bernardini et al. [Bernardini *et al.* 2001] partition the mesh into a set of patches. The use of calibrated lighting conditions allows the construction of albedo and normal maps. New textures are reconstructed by projecting the maps onto the patches and combining the best data available at each point using weights that reflect the level of confidence in the data. Baumberg et al. [Baumberg 2002] presented a multi-band blending approach that preserves high-frequency details in the transition regions of the surface textures and compensates for not perfectly aligned textures.

5.2.2 Blending

Burt and Adelson presented *multi-band blending* [Burt & Adelson 1983a]. Instead of a single, fixed transition width, multi-band blending uses a frequency-adaptive transition width based on image pyramids [Burt & Adelson 1983b]. The input images are decomposed into different frequency bands by building Laplacian pyramids from them. In addition to the input images, multi-band blending takes as input a set of corresponding binary images (masks) indicating the valid regions in the images that are used for blending. From each mask, a Gaussian pyramid is built in order to create appropriate weights for the different frequency bands. Using the weights from the Gaussian pyramids, a feathered blending is performed for each level of the Laplacian pyramids separately, resulting in a composite Laplacian pyramid. The final blended image is obtained by reconstructing the composite Laplacian pyramid.

An alternative approach to multi-band image blending is to perform the operations in the *gradient domain*. Pérez et al. [Pérez *et al.* 2003] showed how gradient domain reconstruction can be used to do seamless object insertion in image editing applications. Rather than copying pixels, the gradients of the new image fragment are copied instead. The actual pixel values for the copied area are then computed by solving a Poisson equation that locally matches the gradients while forcing an exact matching at the seam boundaries. Agarwala et al. [Agarwala *et al.* 2004] extended this idea to a multi-source formulation, where each source image contributes its own gradient field.

Copying gradients directly from the source images after seam placement is just one approach to gradient domain blending. Levin et al. [Levin *et al.* 2004] examine several different variants of this approach which they call gradient-domain image stitching (GIST). The methods they consider include feathering of the gradients from the source images, as well as using an L^1-norm in performing the reconstruction of the image from the gradient field. Their preferred technique is the L^1-optimization of a feathered cost function on the original image gradients. To speed up the rather slow L^1-optimization using linear programming, they develop a faster iterative median-based algorithm in a multigrid framework.

5.2.3 Color Correction

Pyramid and gradient domain blending are able to compensate for moderate amounts of exposure differences between images. However, if exposure differences become large, additional color correction approaches may be necessary. Reinhard et al. [Reinhard *et al.* 2001] presented a method to transfer the color characteristics from a source to a target image. The transformation is carried out in $l\alpha\beta$ color space [Ruderman *et al.* 1998]. Color distributions in the image are modeled by their mean and standard deviation. After the transformation, the color distribution in the target image matches that of the source image.

Agathos and Fisher [Agathos & Fisher 2003] introduced a global color correction method between two images by estimating an RGB color transformation between overlapping pixels. However, their research only considered pairwise corrections. Beauchesne and Roy [Beauchesne & Roy 2003] presented a method to relight overlapping textures of a 3D model. In their method they take two overlapping textures, relight them and merge them into one. This procedure is repeated with the other textures until there is only one left. However, their method can only handle very simple overlap configurations. It is not able to cope with cyclic image overlaps or when the intersection of more than two images is not empty.

Bannai et al. [Bannai *et al.* 2004] extended the method of Agathos and Fisher [Agathos & Fisher 2003] to multiple overlapping images in arbitrary configuration. They first apply a pairwise color correction between the images to obtain good initial estimates. After that, a global optimization based on minimizing per-pixel differences in the overlapping regions is performed. However, minimizing per-pixel differences requires a precise registration and involves high computational costs.

Uyttendaele et al. [Uyttendaele *et al.* 2001] iteratively estimate a local correction between each source image and a blended composite. First, a block-based quadratic transfer function is fit between each source image and an initial feathered composite. Next, transfer functions are averaged with their neighbors to get a smoother mapping, and per-pixel transfer functions are computed by interpolating between neighboring block values. Once each source image has been smoothly adjusted, a new feathered composite is computed. This process is repeated several times (typically three times).

5.3 Compositing of Images on a DEM

In this section a novel method for stitching images on a digital elevation model in order to improve its visual quality is presented. Given a textured DEM, the method takes as input a set of georegistered photos and combines them to textures for the terrain. In a preprocessing step the terrain mesh is reparameterized in order to create a suitable compositing domain. The reparameterization is independent of the given set of input photos and therefore has to be performed only once. The actual compositing then starts by determining visibility for each view in order to identify the regions on the terrain surface valid for texturing with the respective image. With this information at hand, the registered photos are then merged in a two-step procedure: In the first step, a color correction is carried out in order to remove large scale color and lightness shifts. In the second step, the images are blended using a weighted pyramid blending approach in texture space induced by the reparameterization of the terrain geometry. Finally, the obtained result is inserted into the quadtree data structure of the terrain engine to allow real-time rendering.

5.3.1 Texture Representation

Typically, ortho-projection is used to parameterize the terrain geometry for texture mapping. This is appropriate as long as only aerial imagery is used for texturing. However, if images taken from arbitrary viewpoints are to be used as textures, a representation as orthotexture introduces too large distortions at steep slopes. There are two main alternatives for texture representation: the use of a texture atlas and a reparameterization of terrain geometry. In a texture atlas approach, texture patches are stored per triangle and used to texture each triangle separately. This way, there is no distortion introduced at all. However, when using a texture atlas continuity along triangle borders is lost. Since the used blending approach requires a continuous texture patch of considerable size, the texture atlas representation is not suitable for blending.

In view of this, the terrain surface is reparameterized in a preprocessing

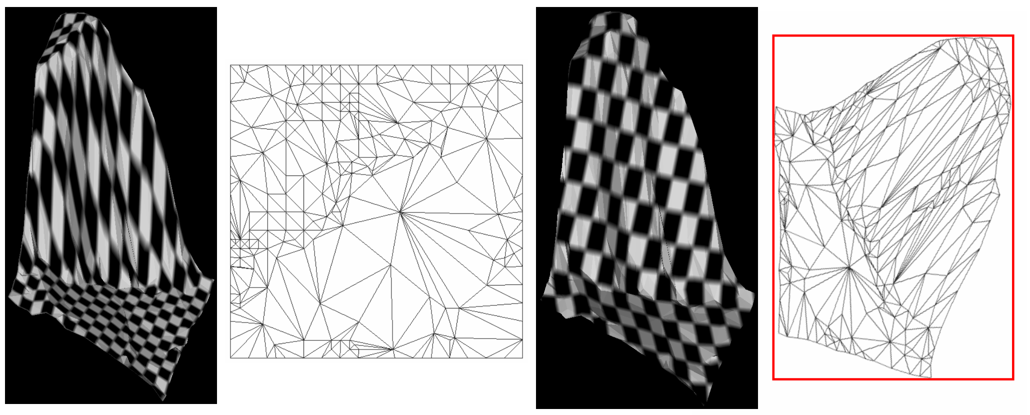


Figure 5.2: Reparametrization. The two images on the left show a geometry patch of a steep slope that is textured with a regular checkerboard pattern using ortho-projection and the corresponding texture space representation of the geometry. Distortions are clearly visible in the steep parts. The two images on the right show the same patch but now using texture coordinates obtained from the reparameterization. Compared to ortho-projection distortions are significantly reduced. The red square in the rightmost image indicates the minimum area rectangle of the mesh in texture space.

step (see Figure 5.2), which allows for a continuous texture representation. By performing the reparameterization for each quadtree tile separately, the introduced distortion is further reduced compared to a global reparameterization. In addition to that, a local reparameterization makes it possible to decide per-patch if a reparameterization is necessary at all. To this end, a simple thresholding scheme is applied. The area of the tile's geometry is computed and divided by the area of its ortho-projection. If this ratio exceeds a given threshold, too much distortion is introduced when using ortho-projection for the considered tile and thus a reparameterization is performed that minimizes the distortion. By using the proposed local and adaptive reparameterization, computational costs as well as memory requirements (needed to store texture coordinates that are no longer implicit as in ortho-projection) are significantly reduced compared to a global reparameterization.

Many parameterization algorithms demand the boundary vertices to be fixed in advance or map to convex polygons, which may be sufficient or even desirable for some applications. However, such restrictions usually introduce additional distortion. For this reason, the algorithm presented in [Degener *et al.* 2003] is used to reparameterize the geometry tiles, which does not constrain the boundary. It quantifies angle and global area deformations simultaneously and lets the user control the relative importance. The importance is chosen in order to obtain a parameterization that is optimized for a uniform sampling of the surface.

The texture space representation of a reparameterized geometry tile has arbitrarily shaped boundaries and arbitrary orientation (see Figure 5.2). To minimize memory requirements, its orientation is optimized to fit as best as possible into a rectangular texture. To this end, the minimum area bounding rectangle in texture space is computed using rotating calipers [Toussaint 1983]. Then, texture coordinates are rotated in such a way that the computed bounding rectangle becomes axis aligned.

5.3.2 Visibility Computation

To texture the terrain geometry with an image from a certain view, it is necessary to identify the visible parts of the surface with respect to this viewpoint and restrict texturing accordingly (see Figure 5.3). This visibility computation has to be performed for each camera and for each level of the quadtree hierarchy. The output of the visibility computation for a certain view consists of a list of indices of the completely visible triangles and a list of the visible subpolygons of the partially visible triangles.

Given a camera and a level of the quadtree hierarchy, the visibility computation starts by culling the tiles' bounding boxes at the camera's view frustum. Then, each of the remaining triangles is assigned a unique color and rendered from the camera's viewpoint into an offscreen buffer. The rendered image is then traversed to identify all rasterized triangles by their color ID. In addition to that, the number of rendered pixels for each triangle is

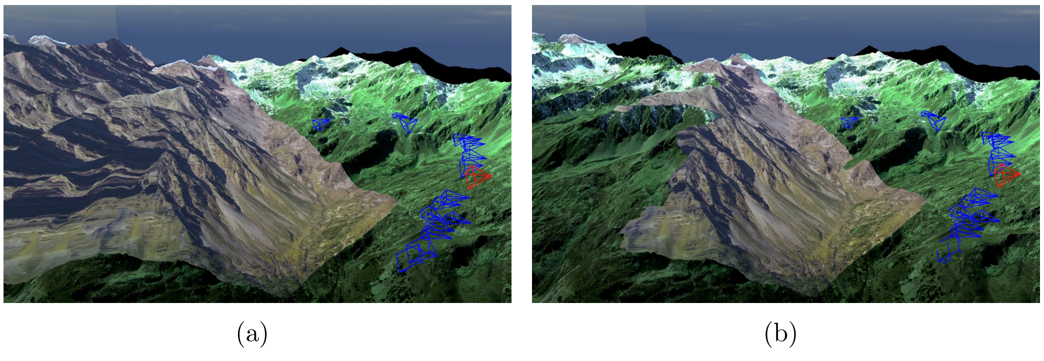


Figure 5.3: **Visibility.** (a) The terrain is projectively textured with a registered photo. Without visibility computations the photo is projected through the terrain geometry onto all geometry it hits. (b) If visibility is considered, the photo is only projected onto the parts of the terrain surface it hits first.

counted and its neighboring triangles in image space are determined. In the next step, it is tested if the gathered triangles are completely visible or only partially visible. To this end, the triangles in question are rendered again but this time with the depth test disabled so that they are all completely rasterized while occlusion queries are used to count the number of rendered pixels. If the number of rendered pixels of a triangle in the two passes coincides, the triangle is completely visible otherwise it is only partially visible.

Once the partially visible triangles in the scene have been detected, their visible parts are calculated exactly. Given a partially visible triangle, first all triangles that occlude it are determined. To find all occluding triangles, the previously detected image space neighbors are inserted into a queue. Then, each triangle in the queue is tested for occlusion. If a triangle is an occluder, it is marked as such, removed from the queue, and all its neighbors are in turn inserted into the queue Otherwise, if the triangle is not an occluder, it is simply removed from the queue. The process stops if the queue is empty. Once all occluding triangles have been detected, the visible area is calculated analytically in image space. To this end, the triangle is clipped against the occluding triangles and against the viewport. Finally, the resulting 2D polygons are unprojected to obtain the visible subpolygons of the triangle in object space.

Note that even at high framebuffer resolutions it cannot be ruled out that very small but visible triangles do not result in a rendered pixel due to discretization. However, such cases were not encountered in practice that would have made any special case handling necessary.

5.3.3 Color Correction

When texturing a 3D object with photos taken from different viewing angles and with different camera settings, measured color and intensity values of a surface element observed in the different photos do usually not agree. The reasons for this are various and include view-dependent lighting effects, such as highlights and specularities, as well as variations in the camera gain settings. Combining such images lead to a mosaic appearance on the surface. To reduce color differences between images, several methods that perform a pairwise color correction have been proposed. However, applying pairwise color corrections to multiple overlapping images is difficult considering the potentially complex topology of overlaps and usually does not result in a globally optimal solution. To overcome this problem, a simultaneous color correction of multiple overlapping images based on minimizing differences in the overlapping regions has to be performed. Using color distributions instead of per-pixel differences allows for a robust handling of mis-registrations and helps to reduce computational costs significantly.

The presented algorithm extends the pairwise color matching approach by Reinhard et al. [Reinhard *et al.* 2001] to a simultaneous matching of multiple overlapping images. In the original approach color characteristics are transferred from a source to a target image in $l\alpha\beta$ color space [Ruderman *et al.* 1998]. A new color $p'_t(x)$ in the target image is computed from the old color $p_t(x)$ as

$$p'_t(x) = \frac{p_t(x) - \mu_t}{\sigma_t}\sigma_s + \mu_s,$$

where μ_s, μ_t are the means and σ_s, σ_t the standard deviations of the underlying Gaussian distribution in the $l\alpha\beta$ color space of the respective source and target images. Despite its simplicity, this approach produces good results if the composition of the source and target images are similar. Since corresponding regions in the photos show the same parts of the terrain surface, this approach is well suited for the considered task.

However, instead of applying color transformations to the whole image, corresponding areas in the images are detected and the color differences between them are simultaneously minimized. The output of the optimization is a set of new means μ' and standard deviations σ' for each overlapping area that are used afterwards to apply the appropriate color corrections.

Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be the set of input images. In the first, step the overlapping areas R_{ij} and R_{ji} of each image pair (I_i, I_j) are determined, where R_{ij} denotes the areas in image I_i that are also visible in image I_j (see Figure

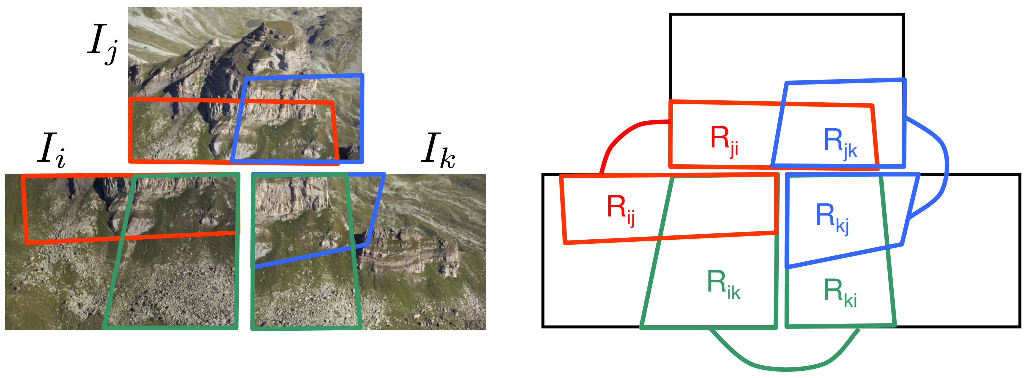


Figure 5.4: Color Correction. On the left, photos with several overlapping areas are shown. On the right, the respective configuration of overlaps is modeled as a graph. Chaining corresponding overlapping regions and defining local operators on them allows for a simultaneous correction of color differences in the images.

5.4). To this end, the information obtained in the previous visibility computation is used to identify the areas visible in both views. For each region R_{ij}, its mean μ_{ij} and standard deviation σ_{ij} are computed using the associated color values of the respective image regions. To measure the dissimilarity d_R between the color distributions associated with the regions, the Weighted-Mean-Variance (WMV) [Manjunath & Ma 1996] is used. With an additional weighting factor this yields

$$d_R(R_{ij}, R_{ji}) = a_{ij} \left(\left| \frac{\mu_{ij} - \mu_{ji}}{\alpha(\mu)} \right| + \left| \frac{\sigma_{ij} - \sigma_{ji}}{\alpha(\sigma)} \right| \right),$$

where $\alpha(\mu)$ and $\alpha(\sigma)$ are the standard deviations of the means and standard deviations of all regions. The area weight

$$a_{ij} = \frac{|R_{ij}| + |R_{ji}|}{|I_i| + |I_j|}$$

reflects the size of the respective overlapping regions.

The configuration of overlaps can be interpreted as a graph $G(\mathcal{V}, \mathcal{E})$, where the overlapping regions R_{ij} are the nodes and edges (R_{ij}, R_{ji}) with costs $d_R(R_{ij}, R_{ji})$ exists between pairs of corresponding overlapping areas (see Figure 5.4). The sum of all edge costs is then minimized

$$\sum_{(R_{ij}, R_{ji}) \in \mathcal{E}} d_R(R_{ij}, R_{ji}) \rightarrow \min_{\mu_{ij}, \sigma_{ij}} !$$

using the Levenberg-Marquardt algorithm. The resulting new means μ' and standard deviations σ' imply a color transformation for each region. The color transformation for a pixel $p_i(x)$ in image I_i with respect to a region R_{ij} is computed as

$$p_{ij}(x) = \frac{p_i(x) - \mu_{ij}}{\sigma_{ij}} \sigma'_{ij} + \mu'_{ij}.$$

The new color of a pixel $p'_i(x)$ in image I_i is then computed as a weighted average of the transformations induced by all regions p_i is located in

$$p'_i(x) = \sum_{\{R_{ij} \mid p_i \in R_{ij}\}} \left(w_{ij}(x) p_{ij}(x) + (1 - w_{ij}(x))\, p_i(x) \right).$$

To compute the weights $w_{ij}(x)$, a color influence map as proposed in [Maslennikova & Vezhnevets 2007] is created that contains for each pixel in the region a weighting factor $c_{ij}(x)$ that describes the influence of the region to this pixel. It is calculated based on the distance of the pixel's color to the color distribution associated with the region. Using the Mahalanobis distance this reduces to

$$d_p(p_i(x), R_{ij}) = \frac{\|p_i(x) - \mu_{ij}\|}{\sigma_{ij}}$$

since color channels are decorrelated in $l\alpha\beta$ color space. From this distance the entries in the color influence map are computed as

$$c_{ij}(x) = e^{-3d_p(p_i(x),R_{ij})^2}.$$

To ensure smooth transitions at region borders in the final image, a distance map is created for each region, which is one in the center of the region and smoothly decreases towards the borders. These per-pixel distance values $d_{ij}(x)$ are multiplied with the entries in the color influence map yielding the final per-pixel weights

$$w_{ij}(x) = c_{ij}(x)d_{ij}(x).$$

5.3.4 Blending

Since color matching only aims at removing large scale color and lightness discrepancies, the images still do not agree perfectly on a per-pixel level due to small mis-registrations and other unmodeled effects. Therefore a good blending strategy is important. A simple approach to blending images is to perform a weighted sum of overlapping color values. However, this approach can cause blurring of high frequency detail if there are small registration errors. To prevent this, multi-band blending was proposed in [Burt & Adelson 1983a]. The basic idea of multi-band blending is to decompose each image into frequency bands. Then, each frequency band is combined separately using a weighting function that fits the size of the features in the respective band. Thus, low frequencies are blended over a large spatial range and high frequencies over a short range. The resulting composite bands are finally recombined to obtain the blended image. This technique allows overlapping images to be blended without introducing visible seams between the images while at the same time high frequency details are preserved and noticeable ghosting artifacts are avoided.

To blend the photos, they first have to be projected into a common planar domain. The texture space representation of the terrain obtained by the reparameterization is well suited for this because it minimizes the distortion introduced by the mapping to 2D. The idea is therefore to carry out the blending in texture space and for each tile separately. Given a geometry tile, the blending can be summarized as follows: First, all cameras the tile is visible from are identified. Then, corresponding texture and weight maps are created in texture space for each camera. Finally, the textures are blended using a multi-band blending technique. In order to ensure smooth transitions across tile borders, blending is performed with slightly overlapping patches. Smooth transitions between the photos and the aerial imagery are obtained by considering the aerial imagery just as an image from another camera. In the following the blending procedure for a given tile is described in more detail.

Given a geometry tile, the texture T_i with respect to the i-th camera is created by rendering the tile's visible texture space representation in an off-screen buffer textured with the corresponding photo (see Figure 5.5). Due to occlusions the texture may contain holes that can cause artifacts during blending if not handled appropriately. Therefore, premultiplied alpha textures are used where the alpha channel contains visibility information with respect to the considered view. By using premultiplied alpha textures, the effects of the occluded areas introduced during blending can be cancelled out afterwards by dividing the final blended texture by its alpha component.

The corresponding weight map W_i for the i-th camera is created in a similar fashion as the texture. However, instead of texturing, each pixel is assigned a weight. The weight

$$w_i^r(x) = Area\,(P_i(t))$$

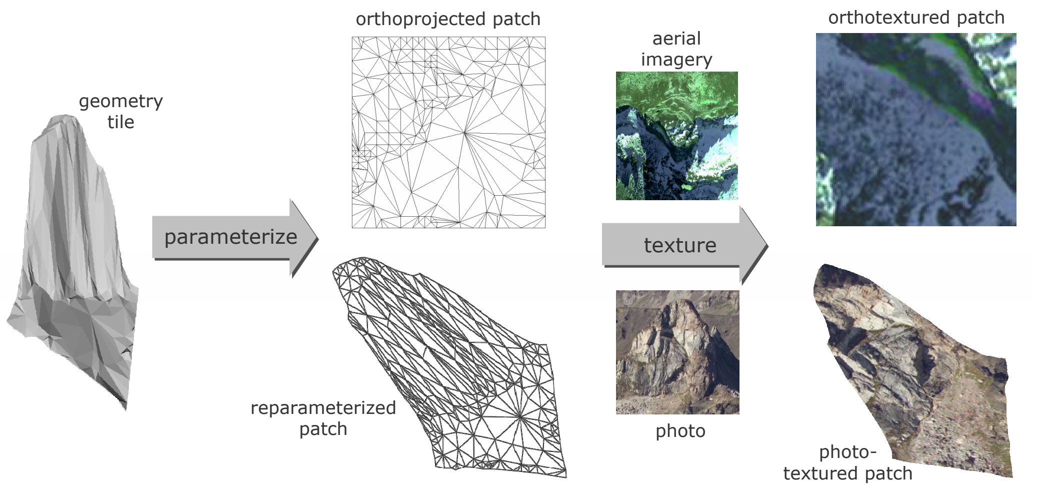


Figure 5.5: Texture Patch. Creation of a texture patch for a geometry tile. Standard ortho-texturing with aerial imagery is shown in the top row, while the bottom row shows the texturing process with a registered photo using the reparameterization.

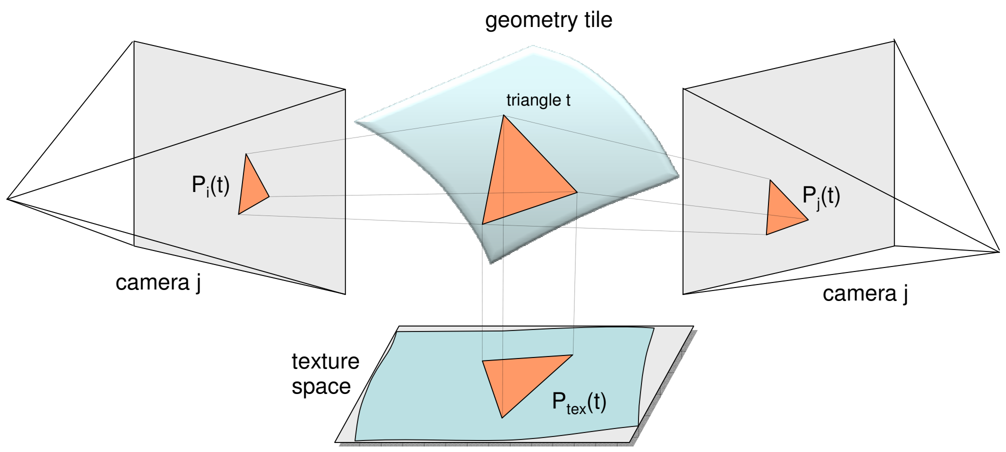


Figure 5.6: Domains. Triangle of a geometry tile and its projections in images and texture domain.

is based on the area of the triangle's projection $P_i(t)$ in the i-th view (see Figure 5.6) reflecting camera position as well as image resolution, where t is the triangle the pixels originate from. In addition to that, pixels close to an image's border as well as pixels near occluded areas are down-weighted using a distance map

$$d_i(x) = \left\| \arg\min_{x'} \{ \|x'\| \mid I_i(x + x') \text{ is not visible } \} \right\|,$$

where each pixel is assigned the distance to the closest pixel not visible in the respective view. The corresponding weights are computed from the distances as

$$w_i^d(x) = \left(\frac{d_i(x)}{\max_x d_i(x)} \right)^4 .$$

The weights are then multiplicated pixelwise

$$w_i(x) = w_i^r(x) w_i^d(x).$$

From these weights the final binary weight map W_i is derived by taking the pixelwise maximum

$$W_i(x) = \begin{cases} 1 & : \text{ if } i = \arg\max_j w_j(x) \\ 0 & : \text{ otherwise,} \end{cases}$$

over all weight maps. The use of these max-weight maps is motivated by the observation that blending high frequency content from multiple images

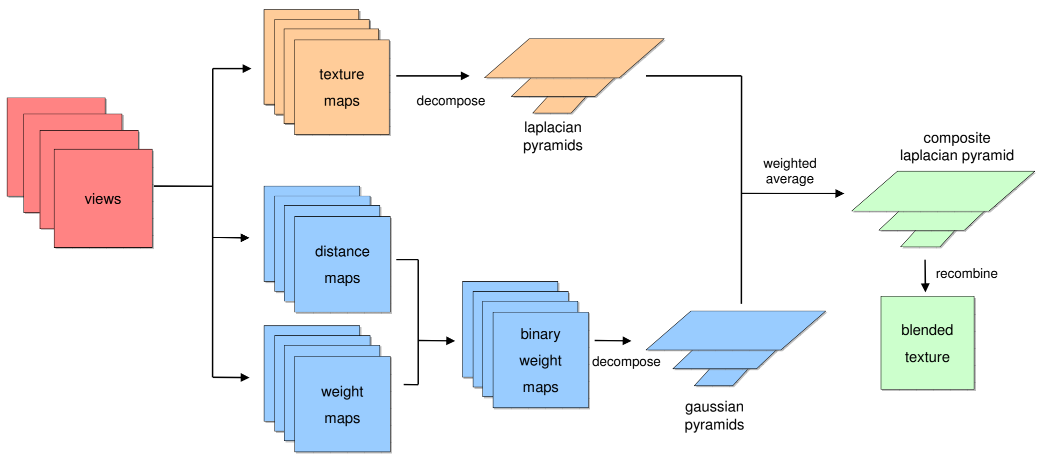


Figure 5.7: Blending Overview. An overview of the patchwise multi-band blending method in texture space.

produces blurred results. Therefore, the best (max-weight) high frequency content is taken only from a single image while for the remaining lower frequency bands the content of all images is blended together over an increasingly large domain.

Once texture and weight maps have been created for each camera, each texture T_i is decomposed into frequency bands by constructing a Laplacian pyramid L_i from it. In contrast, from each binary weight map W_i a Gaussian pyramid G_i is built containing the blending weights for the different bands. Then, a composite Laplacian L_c is created from them by a weighted filtering of the different frequency bands

$$L_c(x) = \frac{\sum_i G_i(x) L_i(x)}{\sum G_i(x)}.$$

The composite Laplacian pyramid L_c is then reconstructed to obtain the blended texture patch. In the resulting texture patch there may be overflow and underflow associated with the color and α values $c = (r, g, b, \alpha)$. To account for that, alpha values have to be clamped to $[0, 1]$ and color values to $[0, \alpha]$

$$\begin{pmatrix} r \\ g \\ b \\ \alpha \end{pmatrix} \longmapsto \begin{pmatrix} \max(0, r) \\ \max(0, g) \\ \max(0, b) \\ \max(0, \alpha) \end{pmatrix} \longmapsto \begin{pmatrix} \min(\alpha, r) \\ \min(\alpha, g) \\ \min(\alpha, b) \\ \alpha \end{pmatrix}.$$

Then, if $\alpha > 0$, the final color values are determined by dividing by α. Finally, the resulting texture patch is stored in the corresponding quadtree tile. An overview of the blending procedure is depicted in Figure 5.7. Optionally, a texture atlas can be created from the texture patches. However, this requires an additional extrusion of the triangles' borders to ensure correct texture filtering.

5.4 Results and Discussion

The presented compositing approach ensures smooth transitions between the images on the terrain surface despite illumination differences while at the same time high frequency details are preserved. In the first step, a color correction is applied that removes large scale color and lightness shifts by simultaneously compensating for differences in the overlapping regions of the

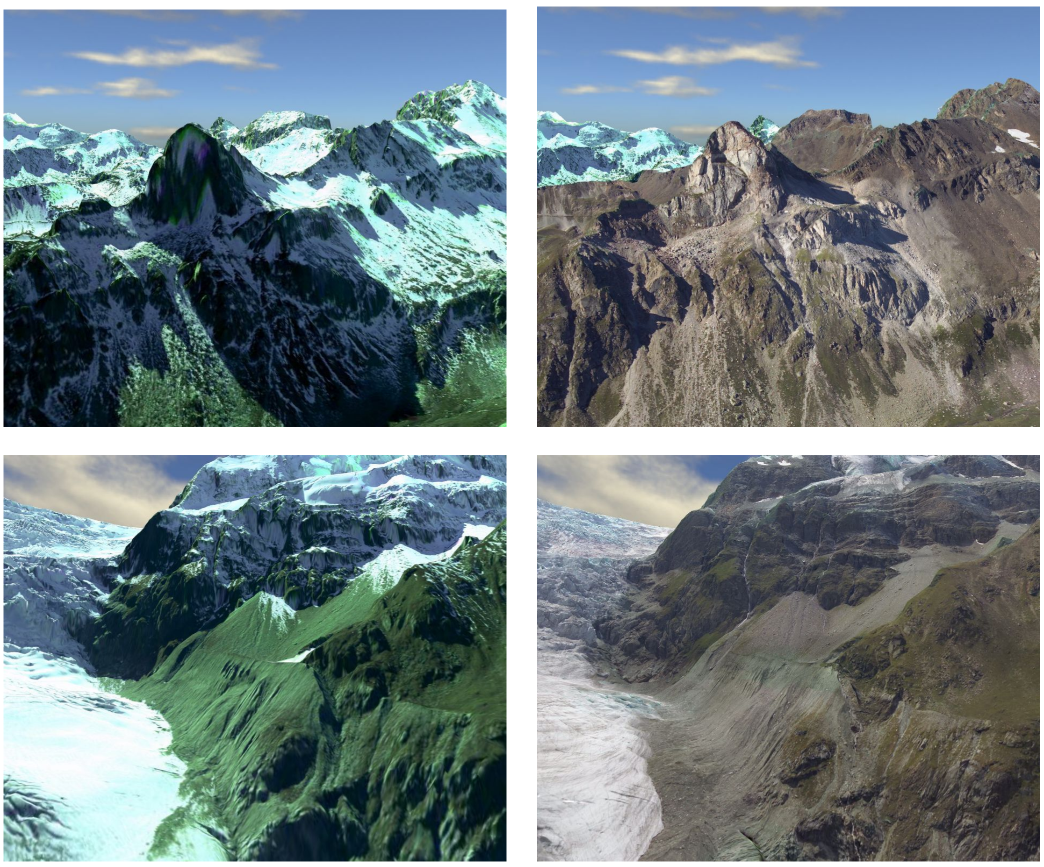


Figure 5.8: Results. The left column shows screenshots of the original ortho-textured DEM. The right column shows the same view but now with the composited photos applied.

images. Then, the resulting images are combined using a multi-band blending approach in texture space.

Figures 5.8 and 5.9 show some results obtained with the presented compositing algorithm. The used photo sets were registered using the registration algorithm presented in the previous chapter of this thesis. The left column shows screenshots of the original ortho-textured DEM using aerial imagery only. In contrast to that, the images in the right column show the same view but now using the composited photos to texture the DEM. The new textures drastically increase the visual quality and information content compared to the aerial imagery in particular in steep slopes.

In Figure 5.10 the presented compositing approach is compared to a use of only the best available view and a weighted averaging. In the results obtained with the approach that uses the best available photo only, seams between the individual photos are clearly visible. Using the weighted averaging, seams

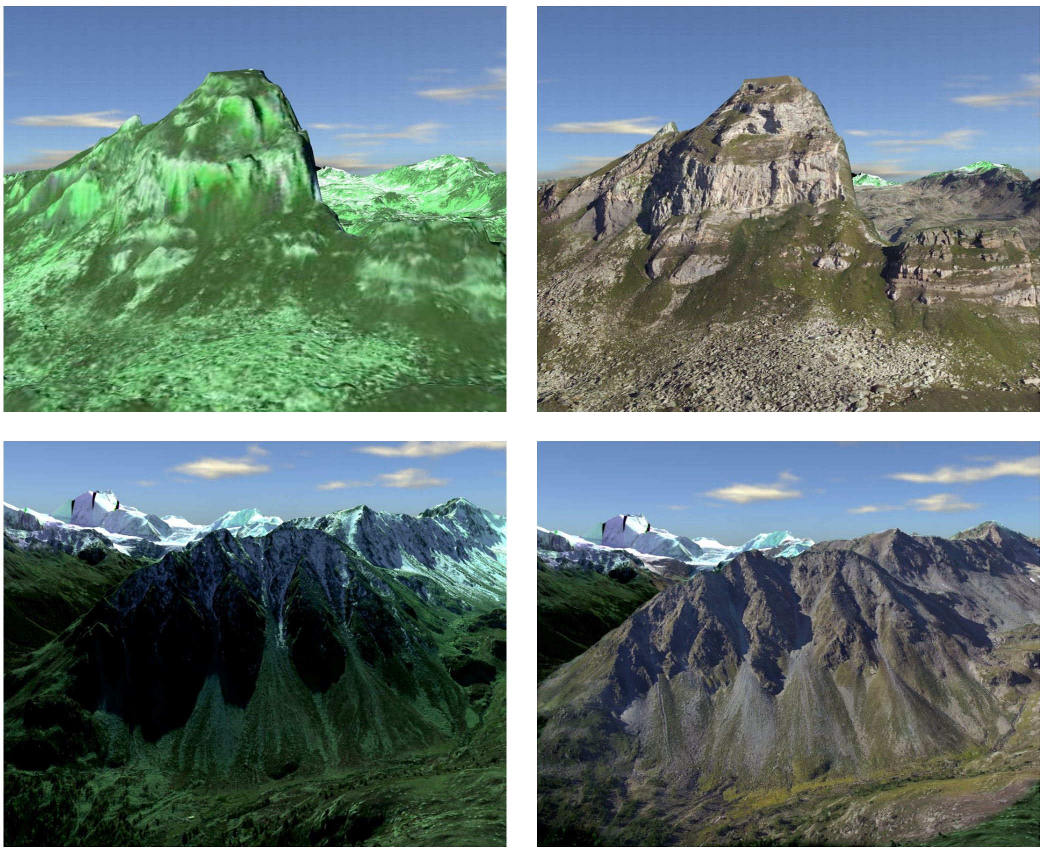


Figure 5.9: Results. Some more results.

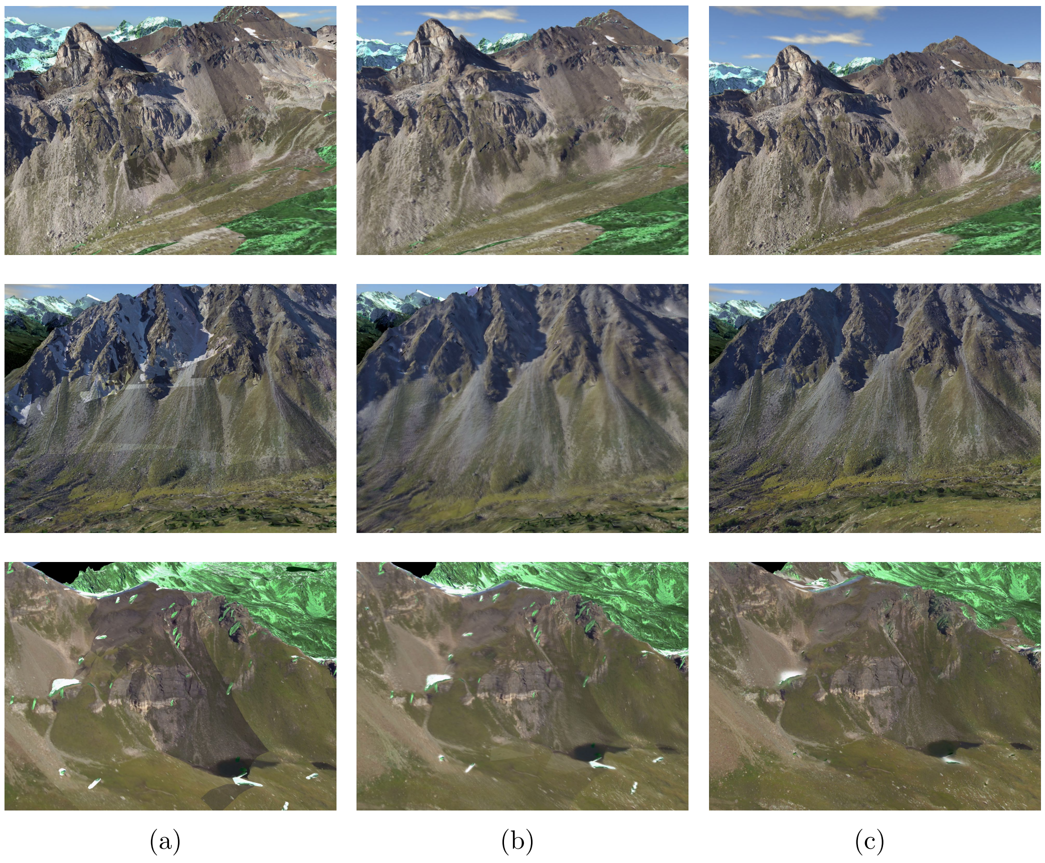


Figure 5.10: Comparison. Column (a) composites the photos using the best available view. Transitions between the individual photos are clearly visible. In (b) a per-pixel weighted blending is used. Seams are less pronounced but still visible. However, significant blurring is introduced. In (c) the presented compositing approach is used. The terrain surface looks consistent without visible seams between the images. In addition, high-frequency features are preserved.

are reduced but still visible. However, significant blurring is introduced. In contrast, using the presented compositing approach the terrain surface looks consistent without visible transitions between the images. In addition, high-frequency features are preserved.

An issue that was not addressed in this thesis are ghosting artifacts caused by moving objects. Such artifacts are common in crowded areas where people, cars or other objects move by, but are not a big issue in the considered application, where photos of the isolated high alpine region of Turtmann valley are to be combined. However, the photos contain temporal changes, as

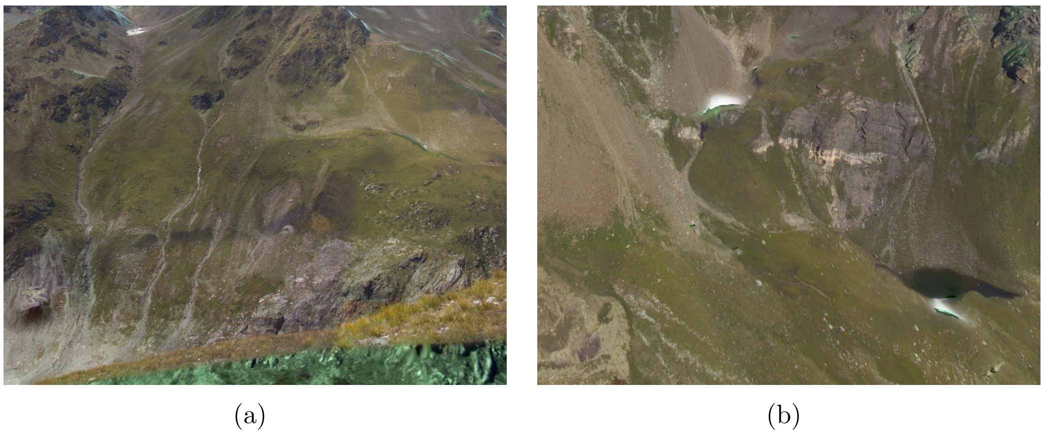


Figure 5.11: Limitations. Two examples of limitations of the presented approach. Image (a) shows grass that is incorrectly projected to the other side of the valley due to a slight misregistration and unmodeled small scale geometry. Image (b) depicts problems when blending between inconsistent images, such as images taken at different seasons.

for example snow coverage, size of the glacier, etc. On the one hand, such differences present a challenge for the compositing algorithms as the blending of inconsistent images might result in artifacts. On the other hand, they offer the opportunity to build time-varying 3D models that can serve to pull together large collections of images pertaining to the appearance, evolution, and events surrounding one place over time. To this end, the photos need to be temporally clustered. However, in the used photo set of Turtmann valley there are far too few photos for such an approach. Therefore, all photos are blended together to cover as much as possible of the terrain surface, albeit inconsistent transitions at places where temporally inconsistent images are blended together may result (see Figure 5.11 (b)). Another issue are artifacts due to registration inaccuracies or unmodeled terrain geometry (see Figure 5.11 (a)). An extension of the compositing approach by a detection and removal of regions of differences [Herley 2005][Uyttendaele *et al.* 2001] in the images prior to blending could potentially reduce these kind of artifacts.

CHAPTER 6

Digital Landform Mapping

6.1 Motivation

Geomorphological maps are the standard means in geomorphology to perceive and investigate an area at focus in a complex and holistic way. Such maps compile knowledge on landforms, surface processes and surface materials, and have widespread applications in land management practices, natural hazard assessments or landform evolution studies [Cooke & Doornkamp 1974][Otto & Dikau 2004][Seijmonsbergen & de Graaff 2006]. *Geomorphological mapping* or *landform mapping* is the process of decomposing the land surface into structural patterns, landforms and landform elements. Traditionally, landform mapping is based on field work supplemented by the interpretation of aerial photographs and literature research. However, due to the increasing availability and quality of digital elevation models, satellite and aerial images, landform mapping is nowadays mainly performed digitally on screen.

Legends and guidelines for geomorphological maps differ from country to country. A review of different geomorphological mapping systems can be found in [Rothenbühler 2003], whereas a presentation of recent mapping concepts is given in [Gustavsson *et al.* 2006][Seijmonsbergen & de Graaff 2006]. In Germany guidelines for geomorphological mapping at large scales (1:25,000 and 1:100,000) have been developed by [Kugler 1964] and within a national geomorphological mapping research program [Stäblein 1980]. At large scales information is often generalized for cartographic reasons, which restricts resolution and therefore the ability to discriminate between individual landforms. The classification of individual landforms implies different attributes that do not only describe the individual characteristics, but also reveal information about patterns of distribution and relationships between the landforms. Dikau [Dikau 1989] divided these attributes into *primary* and *secondary attributes*. Primary attributes include only geomorphometric parameters, such as slope, aspect and curvature, that represent derivates of the elevation data. Secondary attributes refer to the position of the landform relative to the surrounding environment, shape, material, and the geomorphodynamic and geomorphogenetic processes responsible for the landform evolution.

Some recent approaches in remote sensing and GIS aim at an *automatic* recognition of geomorphological objects based on digital terrain data. Automatic landform recognition can be performed using elevation data only [Schmidt & Hewitt 2004][van Asselen & Seijmonsbergen 2006], or by combining elevation data and imagery information [Schneevoigt & Schrott 2006]. However, so far automatic recognition suffers from land surface complexity and its continuous character, represented by diffuse landform boundaries, overlapping landforms and a great variety of structural properties. Consequently, fully automatic landform recognition is (at least at the moment) restricted to landform elements or units, while individual landforms cannot be identified in detail.

A detailed geomorphological map still requires *manual* landform mapping, whether transferred from previously acquired field data, or genuinely mapped from remote sensing data on screen. The accuracy of digital landform mapping depends on the resolution of the terrain data, the visual perception of the virtual land surface morphology, and the diligence and knowledge of the user. However, a fixed 2D bird's eye view representation of aeral imagery and elevation data, as it is common in standard mapping tools, significantly restricts the perception of landforms. Typically, derivatives of elevation data are often used to compensate for this by accentuating morphology changes and break lines in the land surface [Smith & Clark 2005]. For example, relief shading is commonly used for visualizing digital elevation models although it is prone to biasing due to the variable azimuth of the light source.

In contrast, a combination of aerial imagery and elevation data in a 3D visualization is a more natural and intuitive representation of the terrain. Consequently, such 3D visualizations more and more replace the traditional 2D visualization and interpretation methods. However, so far 3D visualization software has usually been restricted to simple data exploration.

In view of this, two *semi-automatic* landform mapping tools are presented that enable the mapping of geomorphological objects directly on the textured DEM (see Figure 6.1). The 3D visualization displays the landform

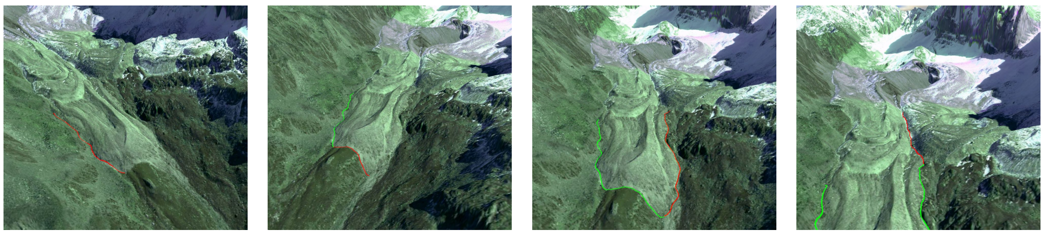


Figure 6.1: Motivation. Landform mapping carried out directly on the textured DEM. During mapping the user can change the viewpoint arbitrarily and thus get additional insight into the structure of the object at focus.

structure in its natural form similar to the perception in the field. Moreover, by navigating in the 3D environment landforms can be inspected from arbitrary views including perspectives hardly possible in nature. In addition to enhanced landform visualization, the proposed landform mapping tools assist the user in specifying geomorphological objects quickly and accurately by using semi-automatic image segmentation techniques. This relieves the user from defining landform borders exactly. Instead, only vague hints that roughly indicate the location of the boundary have to be provided. The two presented methods complement one another since depending on the kind of object that should be mapped, one method or the other may be more effective.

The remainder of this chapter is organized as follows: First, related work is reviewed. Then, the two landform mapping methods are presented, followed by a multilevel banded heuristic to accelerate the segmentation. After that, an extension of the segmentation algorithms is described that enables a detailed mapping at steep slopes. Finally, results are presented and discussed.

6.2 Related Work

In the following, after a brief overview of general image segmentation strategies, semi-automatic image segmentation methods that are relevant within the considered context are reviewed.

6.2.1 Image Segmentation

Image segmentation techniques can be classified with respect to the amount of user interaction necessary into manual, semi-automatic and fully automatic approaches. *Manual* segmentation, while good at object recognition, takes excessive amounts of time and effort for precise boundary capture. Fully *automatic* methods, while much more efficient, often result in an inaccurate segmentation and fail to recognize the object of interest. In contrast to that, *semi-automatic* techniques provide both high efficiency and accuracy by allowing the user to do the high-level task of object recognition and letting the computer capture the low-level details of the object's border. Thus, semi-automatic image segmentation techniques are of great practical use for various applications including medical image analysis, digital image composition, key extraction, etc. Semi-automatic segmentation algorithms can be further categorized into feature-based, region-based and edge-based approaches.

In *feature-based* segmentation algorithms pixels are classified independently of each other based on their position in feature space without explicitly considering connectivity among equally labeled pixels. Common features are intensity, color, gradient magnitude and texture. Grayscale thresholding is probably the simplest of all segmentation techniques since it relies solely on a pixel's intensity. A pixel is classified as belonging to an object if its intensity is greater than or equal to a given threshold intensity. The threshold can be set interactively or automatically, globally or adaptively, optimally or ad hoc. In distance-based classification feature space is divided into a limited number of classes represented by a feature vector or cluster center. Each pixel is assigned to the class that minimizes the distance to the pixel's feature vector. The minimum distance can be Euclidean, as in the nearest-neighbor algorithm, or it can be in terms of variance and covariance, as in Bayesian classification. The cluster centers can be specified manually or they can be determined automatically via a clustering algorithm. While feature-based segmentation is typically fast and simple, it is limited to the segmentation of objects that do not overlap in feature space with the background or other objects in the image. However, this rarely applies to real-world images.

In contrast to that, *region-based* methods specifically try to maintain connectivity while grouping pixels with similar features (i.e., region-based methods extend feature-based segmentation by including connectivity). Region growing [Zucker 1976] starts with some initial regions and grows them by adding neighboring pixels based on some homogeneity criterion. Instead of one or a few regions, region merging [Sonka *et al.* 2007] begins by considering each pixel (or many small regions) and then hierarchically merges neighboring regions with similar properties. In contrast to these bottom-up styles, region splitting [Sonka *et al.* 2007] uses a top-down approach to segmentation. It starts with the entire image as a single region and then recursively subdivides it until each region satisfies a homogeneity criterion. However, a typical problem of the aforementioned region-based techniques are leaking artifacts due to weak object boundaries. In graph cut based methods [Boykov & Jolly 2001] the user first marks certain pixels as object or background to provide hard constraints for segmentation, while additional soft constraints incorporate both boundary and region information. Then, a graph cut optimization is used to find the globally optimal segmentation that gives the best balance of boundary and region properties among all segmentations satisfying the constraints.

Where region-based methods try to identify connected groups of pixels that define an object, *edge-* or *boundary-based* segmentation approaches attempt to determine the contours enclosing the object. Although edge-based and region-based approaches have dual object representations, the segmentation results produced by the two approaches usually differ since edge-based methods typically utilize different image and object criteria than region-based techniques. One of the simplest boundary-based techniques is contour following [Ballard & Brown 1982] or border tracing [Sonka *et al.* 2007] that generates a closed, pixel-based boundary enclosing a region. Local edge following and edge relaxation are examples of local boundary-based algorithms. However, due to their local nature they are subject to local minima. To find globally optimal boundaries based on local cost or weighting criteria [Ballard & Sklansky 1973][Cappelletti & Rosenfeld 1989][Chien & Fu 1974][Sonka *et al.* 1995][Tan *et al.* 1992][van der Zwet & Reiber 1992], dynamic programming is often used. Active contours or snakes [Amini *et al.* 1990][Cohen 1991][Daneels 1993][Geiger *et al.* 1995][Kass *et al.* 1988][Williams & Shah 1992] are manually initialized with a rough approximation of the boundary of interest. The algorithm then iterates over the boundary to determine the boundary that minimizes an energy functional. The energy functional is a combination of external energy supplied by the image, such as gradient magnitude, and internal energy, such as boundary curvature. Intelligent Scissor [Mortensen & Barrett 1995] allows the user to choose a minimum cost contour by roughly tracing the object's boundary with the mouse. As the mouse moves, the minimum cost path from the cursor position back to the last seed point is shown. If the computed path deviates from the desired one, additional user-specified seed points are necessary.

6.2.2 Intelligent Scissors Based Methods

A well-known group of boundary-based techniques are those based on Intelligent Scissors [Mortensen & Barrett 1995][Mortensen & Barrett 1998]. Falcão et al. presented a slightly different version called Live Wire [Falcao *et al.* 1998]. While Intelligent Scissors provides highly interactive visual feedback on small images, the shortest path computation becomes time consuming when the image is large. Therefore, several approaches to accelerate Intelligent Scissors have been published in the following years.

Mortensen et al. [Mortensen & Barrett 1999] presented an enhancement of Intelligent Scissors that over-segments the image using tobogganing and then imposes a weighted planar graph on top of the resulting region boundaries. The derived region-based graph is many times smaller than the pixel-based graph used in the original Intelligent Scissors and thus provides faster graph searches. Furthermore, the region-based graph provides an efficient framework to compute a 4-parameter edge model that allows subpixel localization as well as measuring edge blur. Wong et al. [Wong *et al.* 2000] noticed that pixels within non-edge regions of the image are seldom involved in the determination of boundaries. They exploit their observation by generating a slimmed graph in order to reduce the number of pixels involved in path computation.

Falcão et al. [Falcao *et al.* 2000] presented an approach to improve the time efficiency of their Live Wire method by exploiting the basic properties of Dijkstra's shortest path algorithm. They incrementally expanded the shortest path map only up to the cumulative cost of the current cursor position. This way they avoid an unnecessary computation of paths with bigger cumulative costs, which results in much faster segmentation for large images. However, the response time tends to get longer as the cursor moves further away from the seed point since the overall path map gets bigger.

Live Lane [Falcao *et al.* 1998] restricts the search domain and constructs the path map only within a local window centered at the current seed point. As the cursor moves in this window, the corresponding boundary segment is interactively displayed according to the path map. Whenever the cursor crosses the window, the boundary segment from the seed point to the crossing point is automatically fixed. The crossing point then automatically becomes the new seed and a new path map is constructed within a new window centered at the new seed. Hence, Live Lane requires more seed points than Live Wire, especially when the window size is small. Moreover, the seed points may not lie exactly on the target boundary desired by the user, which degrades the accuracy and repeatability of Live Lane.

Enhanced Lane [Kang & Shin 2002] adopts the idea of the local search from Live Lane in the respect that it also restricts the boundary segment to a small window. However, in Enhanced Lane the window is moved together with the cursor, incrementally extending and updating the path from the current seed point to every pixel in each successive window. In contrast to Live Lane, new seed points are only inserted by the user whereas no additional seed points are inserted automatically. Assuming that the window sequence completely contains the target boundary in the right order, it can be proven that Enhanced Lane always produces the same result as Live Wire.

Kang et al. [Kang 2005] proposed a new Live Wire algorithm called G-wire that is based on a generalized multi-dimensional graph formulation. In addition to external energy, such as gradient magnitude, G-wire is capable of handling internal energy, such as curvature of the boundary curve. As a result, the method produces smoother boundaries and shows improved robustness to noise. However, the consideration of internal energy drastically increases memory requirements and computational costs.

6.2.3 Graph Cut Based Methods

In their seminal work Boykov and Jolly [Boykov & Jolly 2001] presented a technique to segment greyscale images into two disjoint regions. For this purpose, the image is represented as a graph where pixels in the image correspond to nodes in the graph and weighted edges exist between neighboring pixels. The edge weights consist of two components, namely local boundary costs computed from pixel gradients and global region costs derived from intensity histograms. Once the user has marked certain pixels as foreground or background, a min-cut/max-flow algorithm is used to segment the image by minimizing the cost function. The user input is used as hard constraints in the optimization as well as to initialize the intensity histograms.

Li et al. presented an image cutout tool [Li *et al.* 2004] that uses a similar graph cut formulation. However, they cluster the colors in the foreground and background regions using k-means [Duda *et al.* 2000] into 64 clusters. The region costs for a pixel are then computed by its distance to the closest cluster. In addition to that, they perform an over-segmentation using a watershed segmentation prior to the graph cut optimization in order to accelerate the segmentation. After the optimization the user can optionally edit the boundary by moving around individual boundary vertices until the result is satisfactory.

Since the original model depends on parameters which must be set by hand, Blake et al. [Blake *et al.* 2004] formulated a generative, probabilistic model in terms of a Gaussian mixture Markov random field. They used a pseudolikelihood algorithm to learn color mixture and coherence parameters for foreground and background regions. A database of images with correctly segmented results was used to assess their approach.

Rother et al. [Rother *et al.* 2004] extended the original one-shot algorithm to an iterative energy minimization. After each minimization step, pixels are re-labeled and used to update a Gaussian mixture model that is used to model foreground and background regions. Using the iterative approach the amount of user editing is reduced. An incomplete labeling (the user only has to specify hard constraints for either the background or foreground) is sufficient to initialize the optimization.

Lombaert et al. [Lombaert *et al.* 2005] presented a multilevel banded heuristic for the computation of graph cuts. It is motivated by the well-known narrow band algorithm in level set computation. While their approach drastically increases time and memory efficiency, the global optimality of the result can no longer be guaranteed and is limited to the segmentation of large, roundish objects. Sinop et al. [Sinop & Grady 2006] addressed this shortcoming of the banded graph cuts by using information from a Laplacian pyramid to force thin structures to be included into the band. This way, the computational efficiency of the banded graph cut approach can be retained while at the same time the segmentation of thin features is improved.

Juan et al. [Juan & Boykov 2007] presented a hierarchical approach to graph representation called capacity scaling. It can improve the theoretical complexity and practical efficiency of the min-cut/max-flow algorithm. However, unlike the method by Lombaert et al., capacity scaling preserves the global optimality of the solution.

6.3 Intelligent Scissors on Textured DEMs

6.3.1 Basic Idea of Intelligent Scissors

The basic idea of Intelligent Scissors [Mortensen & Barrett 1995] is to formulate the boundary detection problem in an image as an optimal path search in a graph. Nodes in the graph represent pixels in the image and weighted and directed edges are created between nodes that correspond to adjacent pixels in the image. The corresponding edge costs are defined in such a way that an optimal path is likely to correspond to an object boundary.

When the user plants a seed point, a path map is constructed that contains the minimum-cost paths from the seed to every pixel in the image. By interactively moving the cursor near the boundary of an object, the current path is extended according to the path map forming a boundary segment. Whenever the path deviates from the true object boundary, the user can insert an additional seed point. This fixes the current boundary segment and starts a new one originating from the new seed. If a new seed point is created, the path map has to be recomputed with regard to the new seed replacing the previous path map.

Let $G(\mathcal{V}, \mathcal{E})$ be an image graph and $s, d \in \mathcal{V}$ two nodes. Then, a *path* from s to d can be defined as an ordered set of nodes

$$P(s,d) = \{s = p_0, p_1, \ldots, p_n = d\}$$

with $(p_i, p_{i+1}) \in \mathcal{E}, \forall i = 0, \ldots, n-1$. Given non-negative, local costs

$$l : \mathcal{E} \to \mathbb{R}^+$$

defined between adjacent nodes, the costs c of a path P can be computed as

$$c\,(P) = \sum_{i=0}^{n-1} l(p_i, p_{i+1}).$$

An *optimal path* P_{opt} between two nodes is then defined as the path with the lowest costs among the set of all paths $\mathcal{P}$ connecting the two nodes, thus

$$P_{opt} = \arg\min_{P \in \mathcal{P}} c(P).$$

In the remainder of this section, first the local edge costs l are defined. Then, the algorithm is described that is used to compute the optimal path between two given nodes.

Local Edge Costs

Local costs between each pair of adjacent pixels are created as a weighted sum of Laplacian zero-crossing f_z, gradient magnitude f_g and gradient direction f_d. Given a directed edge from node p to a neighboring node q, *local costs* are defined as

$$l(p, q) = w_z f_z(q) + w_g f_g(q) + w_d f_d(p, q),$$

where w_z, w_g and w_d are empirically chosen weights of the corresponding edge features ($w_z = 0.43$, $w_g = 0.43$, and $w_d = 0.13$ are used in the original implementation).

Laplacian zero-crossing and gradient magnitude are common edge operators that convolve an image with multi-scale kernels. The different kernels, each corresponding to a different standard deviation of the underlying 2D Gaussian distribution, are normalized so that comparisons can be made between the results of the convolutions at the different scales. Multiple kernel sizes are used because smaller kernels are more sensitive to fine details, while larger kernels are better at suppressing noise. Using multiple kernels at different scales allows the gradient magnitude and Laplacian zero-crossing to adapt to variety of image types and object edges.

Given an image I, *gradient magnitude* is computed by approximating the partial derivatives in x- and y-direction using derivatives of a Gaussian kernel. Let

$$N_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

be a radially symmetric 2D normal distribution with standard deviation σ. Then, the gradient magnitude for a given σ is computed as

$$G_\sigma(x,y) = |\nabla (N_\sigma * I)| = |(\nabla N_\sigma) * I|\,, \ \forall \sigma \in \mathcal{S},$$

where $\mathcal{S} = \left\{\frac{1}{3}, \frac{2}{3}, 1, 1\frac{1}{3}, 1\frac{2}{3}, 2\right\}$. From these gradient magnitudes computed at the different scales, the scale that best approximates the natural spatial scale of the edge is used to compute the gradient magnitude weight. It is computed on a per-pixel basis as the maximum gradient magnitude over all scales

$$G(x,y) = \max_{\sigma \in \mathcal{S}} G_\sigma(x,y).$$

Since the cost derived from the gradient magnitude needs to be low for strong edges and high for weak edges, the final cost is computed by subtracting the gradient magnitude image from its own maximum and then dividing the result by the maximum gradient. In addition to that, costs are scaled by Euclidean distance $d(p,q)$ of p and q. Thus,

$$f_g(q) = d(p,q)\left(1 - \frac{G'(q)}{\max_{q \in I} G'(q)}\right),$$

with $G'(q) = G(q) - \min(G(q))$ and

$$d(p,q) = \begin{cases} 1 & : \ p \text{ and } q \text{ are diagonal neighbors} \\ \frac{1}{\sqrt{2}} & : \ p \text{ and } q \text{ are vertical or horizontal neighbors.} \end{cases}$$

Laplacian zero-crossing is determined by first computing the Laplacian L of the image as

$$L_\sigma(x,y) = \nabla^2 (N_\sigma * I) = \left(\nabla^2 N_\sigma\right) * I, \ \forall \sigma \in \mathcal{S}.$$

However, a discrete Laplacian image typically produces very few zero-valued pixels. Therefore, zero-crossing is instead represented by two neighboring pixels with opposite sign, where the pixel that is closest to zero is associated with the zero-crossing. Thus, Laplacian zero-crossing is 0 for Laplacian image pixels that are either zero or closer to zero than any neighbor with an opposite sign, otherwise it is 1. Hence,

$$L'_\sigma(q) = \begin{cases} 0 & : \ L_\sigma(q) = 0 \ \vee \ (|L_\sigma(q)| < |L_\sigma(p)| \wedge L_\sigma(p)L_\sigma(q) < 0) \\ 1 & : \ \text{otherwise.} \end{cases}$$

Finally, the zero-crossing cost is computed as the weighted sum of the binary zero-crossings at the different scales

$$f_z(q) = \frac{1}{|\mathcal{S}|}\sum_{\sigma \in \mathcal{S}} L'_\sigma(q).$$

In contrast to gradient magnitude and Laplacian zero-crossing, which aim at edge localization, *gradient direction* adds a smoothness constraint to the boundary by causing high costs for large changes in the boundary direction. The boundary direction is defined as the unit gradient vector

$$D(x,y) = \frac{\nabla \left(N_{\sigma'} * I\right)}{\left|\nabla \left(N_{\sigma'} * I\right)\right|},$$

where σ' denotes the scale with maximum gradient magnitude. Let $D'(p)$ denote the unit vector perpendicular to $D(p)$. Then, the gradient direction cost is defined as

$$f_d(p,q) = \frac{2}{3\pi}\left(\arccos\left(d_p(p,q)\right) + \arccos\left(d_q(p,q)\right)\right),$$

where

$$\begin{aligned} d_p(p,q) &= \langle D'(p), L(p,q)\rangle \\ d_q(p,q) &= \langle D'(q), L(p,q)\rangle \end{aligned}$$

are vector dot products and

$$L(p,q) = \frac{1}{\|p-q\|}\begin{cases} q-p & : \quad \langle D'(p), q-p\rangle \geq 0 \\ p-q & : \quad \text{otherwise} \end{cases}$$

is the normalized link between pixels p and q. The main idea of including the neighborhood link direction is to associate a high cost with an edge between two neighboring pixels that have similar gradient directions but are perpendicular to the link between them. By contrast, gradient direction cost is low when the gradient direction of two neighboring pixels are similar to each other and the link between them.

When computing local edge costs for color images, each color band is processed separately. Then, the results are combined by maximizing over the respective outputs to produce a single-valued cost.

Shortest Path Computation

The minimum-cost path is computed by utilizing an optimal graph search. It is similar to that presented by Dijkstra [Dijkstra 1959], which was later extended by Nilsson [Nilsson 1980] by an additional heuristic to prune the graph search. The formulation of boundary finding as a 2D graph search allows the extraction of boundaries of arbitrary complexity.

Whenever the user places a seed point s, a *path map* is computed that contains the optimal paths

$$\mathcal{P}_{opt}(s) = \{P_{opt}(s, d),\ \forall d \in \mathcal{V}\}$$

from the seed to all nodes d in the graph. Note that, if $P = \{p_0, p_1 \ldots, p_n\}$ is an optimal path, then all sub-paths $P' = \{p_i, \ldots, p_j\}$ with $i < j$ are also optimal paths. Therefore, instead of storing the complete path for each node, it is sufficient to store for each node a pointer $ptr(p_i) = p_{i-1}$ to the next node on the shortest path (see Figure 6.2). Once such a path map has been computed with respect to a given seed point, a desired boundary segment can be chosen dynamically by moving the free point associated with the current cursor position. Interactive movement of the free point causes an update of the current boundary segment by following the optimal path pointers in the path map from the new free point back to the seed point. Thus, by selecting seed points and free points to lie near an object's edge, the user is able to interactively wrap the boundary around an object of interest. If the movement of the free point causes the proposed boundary to digress from the desired object's edge, a new seed point prior to the point of digression can be inserted causing a fixation of the current boundary segment and a recomputation of the path map with respect to the new seed.

The graph search algorithm is initialized by placing the current seed s with a cost $c(s) = 0$ in a sorted list L. All other nodes are initialized with infinite costs. During the algorithm the costs c of a node corresponds to the costs of the path from the seed to the respective node with lowest costs found so far. At the end of the algorithm the costs correspond to the costs of the

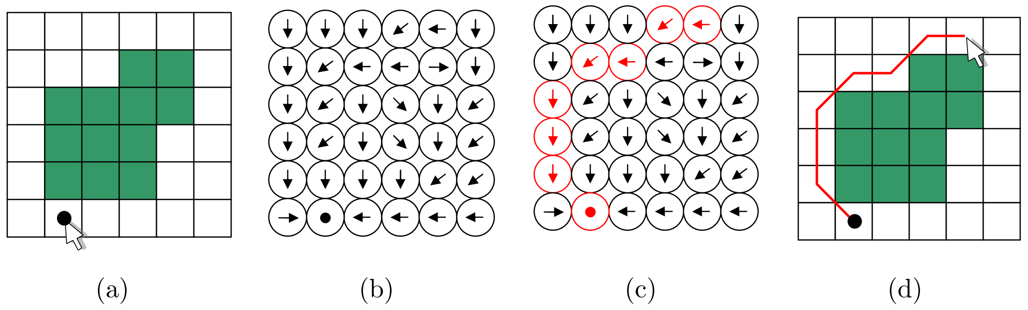


Figure 6.2: Path Map. *From left to right:* (a) A simple example image with a seed point placed by the user. (b) The corresponding path map with respect to the seed. (c) When the user moves the cursor, the path pointers are followed from the free point back to the seed. (d) The optimal path is interactively displayed.

respective shortest path between the seed and the node. The list L is sorted based on the costs of the nodes and is updated appropriately if the cost of a node is decreased.

After initialization the graph search then iteratively generates a minimum cost spanning tree based on the local cost function l. In each iteration the node p with the minimum cost is removed from L and "expanded" by computing the cost to each of p's unexpanded neighbors. For each neighbor q of p, its costs is computed as the cost of p plus the local cost from p to q, thus

$$c'(q) = c(p) + l(p, q).$$

If the newly computed cost of q is less than the current cost, then $c(q)$ is assigned the new, lower cost $c'(q)$ and its optimal path pointer is set to point to p. The process is repeated until all nodes have been expanded (i.e., when L is empty). The described path map computation is summarized in Algorithm 3.

Algorithm 3 ComputePathMap

```
Input:
  $l(p,q)$     // Local edge costs from $p$ to $q$
  $e(p)$    // Boolean function indicating if $p$ has been expanded
  $c(p)$    // Costs from seed to $p$
  $L$    // List of active nodes sorted by their cost $c$

Output:
  $ptr(p)$    // Pointers indicating next node on shortest path

while $L \neq \emptyset$ do
  $p \leftarrow min(L)$
  $e(p) = true$
  for each neighbor $q$ of $p$ with $\neg e(q)$ do
    $c'(q) = c(p) + l(p,q)$
    if $q \in L$ and $c'(q) < c(q)$ then
      $q \leftarrow L$     // remove q from L
    end if
    if $q \notin L$ then
      $c(q) = c'(q)$
      $ptr(q) = p$
      $L \rightarrow q$     // (re-)insert q into L
    end if
  end for
end while
```

6.3.2 Intelligent Scissors on Textured DEMs

Despite the fact that the original Intelligent Scissors technique provides a powerful segmentation tool for images up to a moderate size, its speed and memory consumption constrain its feasibility when dealing with large images. In the case of geospatial data typically very large images, such as high resolution aerial imagery and digital elevation models, are involved that often can not even be completely loaded into main memory. For such images, the direct application of the original Intelligent Scissors approach is not possible at all.

In order to permit interactive performance on out-of-core datasets and to ensure interactive performance, an extension to the basic Intelligent Scissors algorithm is presented that works on tiled images. The main idea is to carry out the segmentation on quadtree representation of the imagery that is used for real-time rendering by the terrain rendering engine. The quadtree representation is utilized to drastically improve performance with respect to memory requirements and computational costs so that interactive response is achieved even on very large datasets. The quadtree data structure is exploited in two ways: First, the search domain is localized, which means that the domain in which an optimal path is sought is restricted to a region around the user input. Second, a multilevel banded heuristic to exploit the hierarchical structure is employed. As this multi-level approach is also used to speed up the graph cut method presented in the next section, it is presented for both in section 6.5.

Localizing the Boundary Search

The presented algorithm takes up the idea of Enhanced Lane [Kang & Shin 2002] of localizing the search domain and incrementally updating the optimal path and adapts it to work on tiled images, as for example given by a quadtree. In addition to that, the idea of an incremental update of the optimal path is extended to the loading of the necessary tiles of the image, to the computation of the edge features and to the construction of the corresponding parts of the graph.

While previous Intelligent Scissors based techniques hold the entire image graph in memory, the presented approach only holds the parts corresponding to the current search domain in memory. When the user moves the mouse so that the search domain needs to be extended, the necessary tiles are loaded on-demand, the graph is extended accordingly and the path map is updated. If the user places a new seed, all currently loaded data is released again.

Let $\mathcal{T}$ denote the set of tiles (in the current implementation a tile size of

128×128 pixels is used) constituting the image, and let

$$\mathcal{S} = \{v(t_0), \ldots, v(t_n)\}$$

be a sequence of free points $v(t_i)$ denoting the position of the mouse cursor at time indices t_i. The seed point corresponds to $v(t_0)$ while the final goal pixel (i.e., the next seed point) corresponds to $v(t_n)$. Further, let $\mathcal{T}_a(t_i)$ denote the set of active tiles currently hold in main memory, and $G(t_i)$ and $D(t_i)$ the graph and the corresponding path map at time index t_i, respectively. Finally, let $\mathcal{T}_n(t_i)$ be the set of tiles that becomes active at time t_i and $G_n(t_i)$ the corresponding graph.

When the user starts the segmentation starts with the placement of the seed point $v(t_o)$, the tile $T(t_0) \in \mathcal{T}$ is determined that contains the seed point $v(t_0)$. This tile is then activated and added to the set of active tiles $\mathcal{T}_a(t_0)$, which is initially empty. The activation of a tile involves the following steps: First, it has to be loaded from disk. However, often tiles are already cached in main memory by the rendering engine since the parts of the terrain that are segmented are visualized at the same time. Once the image tile has been loaded, the corresponding graph $G(t_0)$ is created and the respective local edge costs are computed on-the-fly. The edge costs are computed in the same way as in the original Intelligent Scissors algorithm with one exception: The geometric distance between pixels p and q is now calculated in 3D as

$$d(p,q) = \frac{\|h(p) - h(q)\|}{\max_{(p,q)\in\mathcal{E}} \|h(p) - h(q)\|}$$

using the DEM h. After the computation of the local costs, the image tile is no longer needed and can be released from memory. Finally, the path map $D(t_0)$ is computed for the graph $G(t_0)$ with respect to the seed point $v(t_0)$. To this end, the cost of $v(t_0)$ is set to zero, inserted into the sorted list L, and Algorithm 3 is called.

As long as the free point remains inside the current search domain, the current path map stays valid and can be used to look up the shortest path and display the corresponding boundary segment. If, however, the cursor is moved out of the active search domain, the search domain has to be extended accordingly (see Figure 6.3) and the path map has to be updated. Let's now consider the situation at time t_i, and let $T(t_i)$ be the tile the current free point $v(t_i)$ is located in. Certainly, the tile $T(t_i)$ has to be activated and added to the set of active tiles. In addition to that, also the tiles that lie within a given distance d_t to $v(t_i)$ are activated (if not already active). This way it is assured that always at least an area of radius d_t centered around the current free point is active. Otherwise, object boundaries lying just outside

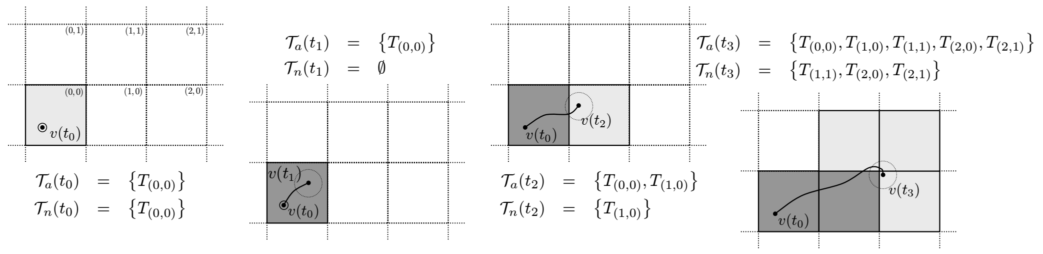

(0,1)
(1,1)
(2,1)
(0,0)
(1,0)
(2,0)
$v(t_0)$
$\mathcal{T}_a(t_0) = \{T_{(0,0)}\}$
$\mathcal{T}_n(t_0) = \{T_{(0,0)}\}$
$\mathcal{T}_a(t_1) = \{T_{(0,0)}\}$
$\mathcal{T}_n(t_1) = \emptyset$
$v(t_1)$
$v(t_0)$
$v(t_2)$
$v(t_0)$
$\mathcal{T}_a(t_2) = \{T_{(0,0)}, T_{(1,0)}\}$
$\mathcal{T}_n(t_2) = \{T_{(1,0)}\}$
$\mathcal{T}_a(t_3) = \{T_{(0,0)}, T_{(1,0)}, T_{(1,1)}, T_{(2,0)}, T_{(2,1)}\}$
$\mathcal{T}_n(t_3) = \{T_{(1,1)}, T_{(2,0)}, T_{(2,1)}\}$
$v(t_3)$
$v(t_0)$

Figure 6.3: Expansion of Search Domain. Whenever the mouse cursor comes close to the border of the current search domain, the search domain is expanded accordingly. $\mathcal{T}_a$ are the tiles that are already active at the respective time index, while $\mathcal{T}_n$ are the tiles that are activated to expand the search domain.

the search domain might be missed, although the cursor is close to them. More precisely, the set of new tiles is

$$\mathcal{T}_n(t_i) = \{T \in \mathcal{T} \mid T \notin \mathcal{T}_a(t_{i-1}) \ \wedge \ d(v(t_i), T) < d_t\},$$

where $d(v(t_i), T)$ denotes the distance between $v(t_i)$ and T, and d_t is a distance threshold (set to half the tile size in the current implementation). These tiles are then activated as described above, resulting in the graph $G_n(t_i)$. The new set of active tiles

$$\mathcal{T}_a(t_i) = \mathcal{T}_a(t_{i-1}) \ \cup \ \mathcal{T}'(t_i)$$

is then obtained as the union of the previous set of active tiles $T_a(t_{i-1})$ and the tiles $T_n(t_i)$ activated in the current step. The new graph

$$G(t_i) = G(t_{i-1}) \ \cup \ G_n(t_i) \ \cup \ \mathcal{E}(t_i).$$

is obtained in a similar fashion as the union of the old graph $G(t_{i-1})$ and the new graph $G_n(t_i)$. However, additional edges

$$\mathcal{E}(t_i) = \{(p, q) \mid p \in G(t_{i-1}), \ q \in G_n(t_i), \ p \in N(q)\}$$

that connect the graphs appropriately have to be added, where $N(\cdot)$ is the set of neighboring nodes.

Once the graph has been extended, the path map $D(t_{i-1})$ constructed so far has to be expanded and updated accordingly (see Figure 6.4). The domain for which the path map has to be computed certainly comprises the new nodes in $G_n(t_i)$ since they have not been explored yet. However, some nodes in $G(t_{i-1})$ for which a shortest path has already been computed in $D(t_{i-1})$ might have to be updated as well, This is because there now might

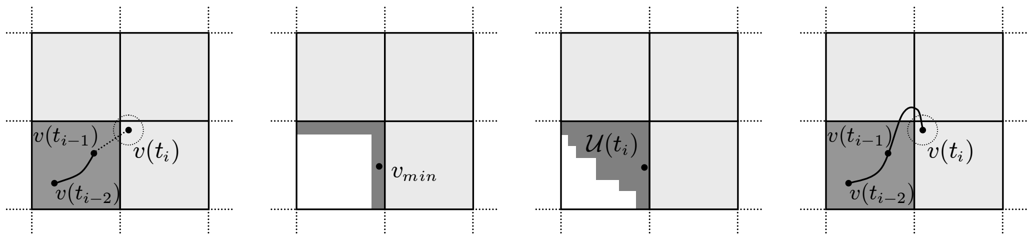


Figure 6.4: Incremental Path Map Update. If the search domain is expanded, the path map has to be expanded and updated as well. The necessary update of the path map is not restricted to the newly added nodes, but may also comprise nodes that have already been in the search domain before expansion. This is because for some nodes there might now exists a shorter path running through the new part of the graph. However, in order to be able to reduce the cost of such a node, its current cost has to be greater than that of the minimum-cost node v_{min} at the border between the old and the new graph. Therefore, all nodes $\mathcal{U}$ that have a greater cost then v_{min} are determined, and their shortest path is updated.

exist a shorter path through nodes in $G_n(t_i)$ that have not been considered until now. In order to be able to reduce the cost of a node in $G(t_{i-1})$, its current cost has to be greater than that of the minimum-cost node $v_{min}(t_i)$ at the border between the old and the new graph, otherwise it cannot be improved. Let

$$\mathcal{B}(t_i) = \{p \in G(t_{i-1}) \mid \exists\ q \in N(p) \text{ with } q \in G_n(t_i)\}$$

denote the set of border nodes. Then, the minimum-cost border node is

$$v_{min}(t_i) = \arg\min_{p \in \mathcal{B}(t_i)} c(p).$$

Therefore, in addition to the nodes of the new graph $G_n(t_i)$, all nodes

$$\{p \in G(t_{i-1}) \mid c(p) > c(v_{min}(t_i))\}$$

of the old graph $G(t_{i-1})$ with greater costs than $v_{min}(t_i)$ are inserted into L as well. This is conceptually equivalent to backtracking the path map construction to the state where $v_{min}(t_i)$ has been about to be expanded (i.e., when it was the first element in the list L). Using the such initialized list L, the path map is updated using Algorithm 3. Note that the update domain is restricted to the newly added tiles and their neighbors that have already been activated. Consequently, the time for the path map construction is independent of the size of the current search domain and the considered image.

Using Elevation Data

So far, only aerial imagery has been considered as input for the segmentation. However, aerial imagery is typically only able to reproduce the relief structure up to a certain extent. In addition to that, shadows, snow or other phenomena might occlude an object of interest. Therefore, digital elevation models, if available, are also often used as basis for landform mapping. For landform mapping purposes, a DEM is usually represented as shaded relief or using derivatives of it, such as slope and curvature. To enable such visualizations of the terrain, according shaders were implemented that allow the user to view the terrain as shaded relief, or shaded depending on slope or curvature (see Figure 6.5).

In addition to the visualization, shader output can also be used as input for the segmentation. This way the user can choose the representation that is best suited for the visualization and mapping of the object at focus. In the case of shaded relief, slope, or curvature the input image for the segmentation is just a single channel image. The segmentation algorithm is not affected by this, since it does not make any assumptions on the input data.

The shaded relief is computed using simple diffuse reflection. Each pixel is assigned a gray value proportional to the cosine of the angle between the surface normal n and the light vector l. The light direction can be interactively changed by the user, which can be used to accentuate the object of interest. Slope is calculated as the gradient magnitude

$$|\nabla h(x,y)| = \sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2},$$

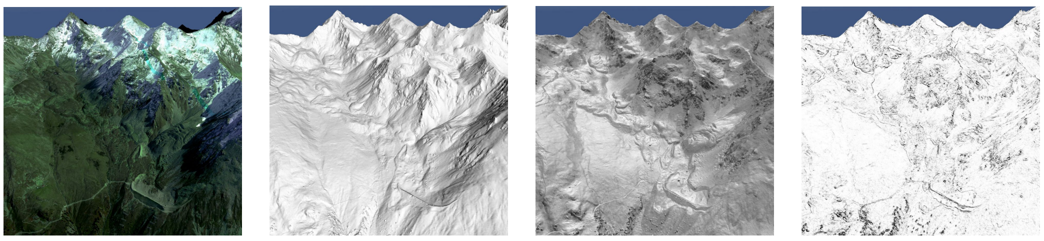

(a) texture (b) shaded relief (c) slope (d) curvature

Figure 6.5: Different Relief Representations. The user can choose how the relief is visualized and at the same time select the relief parameter landform mapping is based on. This way, the user can use the representation that is most effective to map the object at focus.

whereas curvature is computed as the Laplacian

$$\nabla^2 h(x,y) = \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}$$

of the DEM h. Although the mentioned relief parameters should meet most requirements, the system can, if required, naturally and easily be extended to support additional relief parameters (as long as they are of local nature and can be efficiently computed in a shader).

6.4 Graph Cuts on Textured DEMs

6.4.1 Basic Idea of Graph Cuts

In this section the basic idea of image segmentation based on a graph cut formulation is reviewed as introduced in the seminal work by Boykov and Jolly [Boykov & Jolly 2001]. The goal is to divide an image into two disjoint sets: foreground and background. For this purpose, the image is represented as a graph where pixels in the image correspond to nodes in the graph and weighted edges exist between neighboring pixels. The edge weights consist of two components, namely local boundary costs computed from pixel gradients and global region costs derived from intensity histograms. By marking certain regions in the image as foreground or background, respectively, the user imposes hard constraints for the segmentation. The remaining parts of the image are then segmented automatically by computing a global minimum among all segmentations that satisfy the hard constraints. The cost function is defined in terms of boundary and region properties of the resulting segments. These properties can be interpreted as soft constraints.

Let $\mathbf{x} = (x_1, \dots, x_n)$ be a vector containing the pixels of the image and let $\mathcal{N}$ be the set of all unordered pairs (x_i, x_j) of neighboring pixels in the image under a standard 8-neighborhood system (generally, the neighborhood system can be arbitrary). Further, let $G = (\mathcal{V}, \mathcal{E})$ be a graph defined by a set of nodes $\mathcal{V}$ and a set of undirected edges $\mathcal{E}$. Nodes are created corresponding to pixels x_i in the image. In addition, there are two *terminal nodes* S (source) and T (sink) that represent foreground and background labels, thus

$$\mathcal{V} = \{S, T\} \bigcup_{i=1}^{n} \{x_i\}\,.$$

The set of edges $\mathcal{E}$ consists of two types of undirected edges: *n-links* (neighborhood links) and *t-links* (terminal links). Each pair of neighboring pixels

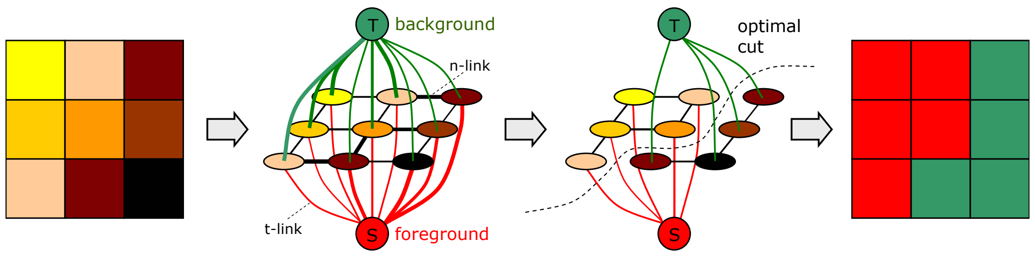


Figure 6.6: Graph Cut Example. A simple segmentation example of a 3x3 image. A graph with n-links between neighboring nodes and t-links connecting each node to the source S (red) and sink T (green) is created from the image. The cost associated with each edge is reflected by its thickness. After the graph cut minimization the graph consists of two disjoint sets.

(x_i, x_j) in $\mathcal{N}$ is connected by an n-link. In addition, each pixel x_i has two t-links (x_i, S) and (x_i, T) connecting it to each terminal, hence

$$\mathcal{E} = \mathcal{N} \bigcup_{i=1}^{n} \{(x_i, S), (x_i, T)\}\,.$$

An *s-t cut* is a subset of edges $\mathcal{C} \subset \mathcal{E}$ such that in the induced graph $G(\mathcal{C}) = (\mathcal{V}, \mathcal{E} \setminus \mathcal{C})$ the terminals S and T become completely separated and the nodes are divided between them (see Figure 6.6). Consequently, a cut corresponds to a binary partitioning of the underlying image into foreground and background with arbitrary topology. The goal with respect to image segmentation is then to compute a cut that results in an optimal segmentation. Severed n-links are located at the segmentation boundary and hence represent the cost of the segmentation boundary. In contrast, severed t-links are used to model regional properties of segments. Thus, a minimum cost cut corresponds to a segmentation with a desirable balance of boundary and regional properties. In addition to that, hard constraints can be modeled by assigning infinite costs to t-links.

Let $\mathbf{a} = (a_1, \ldots, a_n)$ be a binary vector whose components a_i specify assignments of pixels x_i to either foreground ("frg") or background ("bkg"). Then, the soft constraints imposed on the boundary and region properties of $\mathbf{a}$ are described by the cost function

$$E(\mathbf{x}, \mathbf{a}) = \lambda R(\mathbf{x}, \mathbf{a}) + B(\mathbf{x}, \mathbf{a}), \tag{6.1}$$

where

$$\begin{aligned} R(\mathbf{x}, \mathbf{a}) &= \sum_{i=1}^{n} R(x_i, a_i) \\ B(\mathbf{x}, \mathbf{a}) &= \sum_{(x_i, x_j) \in \mathcal{E}} B(x_i, x_j) \delta(a_i, a_j) \end{aligned}$$

and

$$\delta(a_i, a_j) = \begin{cases} 0 & : \quad a_i = a_j \\ 1 & : \quad a_i \neq a_j. \end{cases}$$

The coefficient λ specifies the relative importance of the region properties term $R(\mathbf{x}, \mathbf{a})$ versus the boundary properties term $B(\mathbf{x}, \mathbf{a})$. The regional term $R(\mathbf{x}, \mathbf{a})$ assumes that individual costs for assigning a pixel x_i to the foreground or background are given by $R(x_i, \text{"frg"})$ and $R(x_i, \text{"bkg"})$, respectively. In the original work by Boykov et al. pixels marked by the user are used to build histograms of pixel intensities for the foreground $p(x_i | \text{"frg"})$ and background $p(x_i | \text{"frg"})$. These histograms are then used to set region costs as negative log-likelihoods

$$\begin{aligned} R(x_i, \text{"frg"}) &= -\log p(x_i | \text{"frg"}) \\ R(x_i, \text{"bkg"}) &= -\log p(x_i | \text{"bkg"}). \end{aligned}$$

The use of negative log-likelihoods is motivated by the MAP-MRF formulations in [Greig *et al.* 1989][Boykov *et al.* 1998].

The term $B(\mathbf{x}, \mathbf{a})$ comprises the boundary properties of segmentation $\mathbf{a}$. It can be interpreted as a penalty for a discontinuity between x_i and x_j. Typically, $B(x_i, x_j)$ is large when pixels x_i and x_j are similar, and small when they are very different. $B(x_i, x_j)$ may be based on local intensity gradient, Laplacian zero-crossing, gradient direction or similar criteria. In the original implementation boundary costs are modeled using intensity differences

$$B(x_i, x_j) = \frac{1}{dist(x_i, x_j)} e^{-\frac{\left(I(x_i) - I(x_j)\right)^2}{2\sigma^2}},$$

where $I(x_i)$ denotes the intensity of pixel x_i and σ is the standard deviation of pixel intensities in the image. This function penalizes a lot for discontinuities between pixels of similar intensities if $|I(x_i) - I(x_j)| < \sigma$. Otherwise, if pixels are very different, the penalty is small. Intuitively, this function corresponds to the distribution of noise among neighboring pixels of an image.

Assume that $\mathbf{f} = (f_1, \ldots, f_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$ are binary vectors containing the hard constraints imposed by the user input. $f_i = 1$ indicates that the user has marked the pixel x_i as foreground and $b_i = 1$ as background, respectively, where $f_i \neq b_i, \forall i$. Then the edge weights $\mathcal{E}$ are defined as

edge	cost	case
(x_i, x_j)	$B(x_i, x_j)$	$(x_i, x_j) \in \mathcal{N}$
(x_i, S)	$\lambda R(x_i, \text{“bkg”})$	$f_i = b_i = 0$
	K	$f_i = 1$
	0	$b_i = 1$
(x_i, T)	$\lambda R(x_i, \text{“frg”})$	$f_i = b_i = 0$
	0	$f_i = 1$
	K	$b_i = 1$

where

$$K = 1 + \max_{x_i} \sum_{(x_i, x_j)\ :\ x_j \in \mathcal{N}(x_i)} B(x_i, x_j).$$

Once the graph has been completely defined, the goal is to compute the global minimum of Equation 6.1 among all segmentations **a** satisfying the hard constraints

$$\begin{aligned} \forall f_i = 1, \quad a_i &= \text{“frg”} \\ \forall b_i = 1, \quad a_i &= \text{“bkg”}. \end{aligned}$$

A globally minimum *s*-*t* cut can be computed efficiently in low-order polynomial time [Ford & Fulkerson 1962][Goldberg & Tarjan 1988][Cook *et al.* 1998]. The corresponding algorithms work on any graph and are therefore not restricted to 2D images but can also be used to compute globally optimal segmentation on volumes of any dimension. Apart from the theoretical considerations, an efficient implementation of graph cut algorithms can be an issue in practice. The most straightforward implementations of the standard graph cut algorithms can be slow. Experiments in [Boykov & Kolmogorov 2004] compared several well-known versions of these standard algorithms in the context of graph based methods in vision. The same paper also described a new version of the max-flow algorithm that (on typical vision examples) significantly outperformed the standard techniques.

6.4.2 Graph Cuts on Textured DEMs

In the following a graph cut based approach for the segmentation of textured DEMs is presented. The user interface is similar to that of Li et al. [Li *et al.* 2004], where the user marks parts of the foreground and background background by brushing with the mouse. In the presented approach brushing is directly performed on the textured DEM. This user input is used as hard constraints as well as to initialize the region model as in the standard approach.

Previous graph cut based approaches for image segmentation run the graph cut optimization on the whole image. While this can already become slow for moderate-sized images, it is far from providing interactive feedback for huge aerial images. However, it is not necessary to include the whole aerial image into the optimization. Instead, only a small area around the user input is used. To derive this area of interest, the convex hull of the user hints is computed and all tiles of the quadtree that are at least partially contained in the convex hull are loaded. Then, these tiles are sewed together and a graph is created from them the same way as in the original approach.

Region Term

In early approaches histograms of pixel intensity or color were used to model foreground and background regions. However, it is often impractical to build adequate histograms. For small sample sizes it is usually better to estimate solely marginal histograms. Although information about the joint occurrence of features in the different dimensions is lost, bin contents in the marginals may be significant where those in the full distribution would be too sparse. Moreover, for high-dimensional feature spaces a regular binning often results in poor performance. While a coarse binning degrades resolving power, a fine binning leads to statistically insignificant sample sizes for most of the bins. A partial solution is offered by adaptive binning, whereby the histogram bins are adapted to the distribution. The binning is induced by a set of prototypes, which can be determined by vector quantization, for example k-means [Duda *et al.* 2000].

An alternative is to use *Gaussian mixture models* (GMM), which have already been used for soft segmentation [Ruzon & Tomasi 2000][Chuang *et al.* 2001] as well as for color and texture segmentation [Permuter & Francos 2003]. If x_i is an observation vector (of random variables) of dimension d and θ is a vector of unknown parameters, then the likelihood $p(x_i|\theta)$ can be represented as a marginal over hidden variables

$$p(x_i|\theta) = \sum_{j=1}^{K} p(x_i|\theta_j)p(\theta_j).$$

Assuming that the $p(x_i|\theta_j)$ are Normally distributed with mean μ_j and covariance matrix Σ, they can be written as

$$p(x_i|\theta_j) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_j|^{\frac{d}{2}}} e^{-\frac{1}{2}\left((x_i-\mu_j)^{\mathsf{T}}\Sigma_j^{-1}(x_i-\mu_j)\right)},$$

where $\theta_j = (\mu_j, \Sigma_j)$, and $|\Sigma|$ denotes the determinant of the covariance matrix. The expectation maximization (EM) algorithm [Dempster *et al.* 1977] can be used to estimate θ_j and the mixture weights $p(\theta_j)$.

The used region modeling follows the approach by Rother et al. [Rother *et al.* 2004] that models foreground and background regions using Gaussian mixture models. The foreground and background regions are represented using a GMM with $K = 5$ components. Instead of a soft assignment of probabilities for each component to a given pixel, each pixel is assigned a unique GMM component. To this end, an additional vector $\mathbf{k} = (k_1, \ldots, k_N)$ with $k_i \in \{1, \ldots, K\}$ is introduced that assigns each pixel a GMM component either from the foreground or background. Although a soft assignment of probabilities might seem preferable as it would allow for expectation maximization, it involves significant additional computational expense while at the same time the practical benefit turns out to be negligible. The region costs are then defined as

$$\begin{aligned} R(x_i, a_i, k_i, \theta) &= -\log p(c(x_i)|a_i, k_i, \theta) \\ &= -\log \left(\sum_{j=1}^{K} p(c(x_i)|a_i, k_i, \theta_j) p(\theta_j|a_i, k_i) \right) \\ &= \sum_{j=1}^{K} -\log p(c(x_i)|a_i, k_i, \theta_j) - \log p(\theta_j|a_i, k_i) \end{aligned}$$

where $p(c(x_i)|a_i, k_i, \theta_j)$ is a multivariate Gaussian distribution, $p(\theta_j|a_i, k_i)$ are the mixture weighting coefficients and $c(x_i)$ denotes the RGB color of pixel x_i.

Boundary Term

To set the boundary penalties, a contrast dependent distance of the respective color values scaled by the pixel distances is used

$$B(x_i, x_j) = \frac{1}{\|h(x_i) - h(x_j)\|} e^{-\beta \|c(x_i) - c(x_j)\|^2},$$

where $c(x_i)$ denotes the RGB color of pixel x_i, and h is the DEM. When β is zero, the smoothness term is simply the well-known Ising prior, encouraging smoothness everywhere, to a degree determined by the constant λ. However, it has been shown [Boykov & Jolly 2001] that it is more effective to set $\beta > 0$ as this reduces the tendency to smoothness in high contrast regions. Therefore, the constant β is chosen as

$$\beta = \frac{1}{2 \left\langle \|c(x_i) - c(x_j)\|^2 \right\rangle},$$

where $\langle\cdot\rangle$ denotes expectation over the image. This choice of β ensures that the exponential term switches appropriately between high and low contrast.

Energy Function and Minimization

As in the previous section let $\mathbf{x} = (x_1, \ldots, x_n)$ be a vector containing the image pixels and $\mathbf{a} = (a_1, \ldots, a_n)$ be the corresponding binary assignment vector. Further, let $\theta = \{\mu(a,k), \Sigma(a,k)\}$ with $a \in \{0,1\}$ and $k \in \{1, \ldots, K\}$ denote the parameters of the GMM model. Then, the energy function for the segmentation is defined as

$$E(\mathbf{x}, \mathbf{a}, \mathbf{k}, \theta) = \sum_{i=1}^{n} R(x_i, a_i, k_i, \theta) + \lambda \sum_{(x_i, x_j) \in \mathcal{E}} B(x_i, x_j) \delta(a_i, a_j)$$

Instead of the previous one-shot algorithm [Boykov & Jolly 2001], the iterative energy minimization scheme presented in [Rother *et al.* 2004] is used. This has the advantage of allowing automatic refinement of the GMM parameters using newly labeled pixels. First, pixels are assigned GMM components. Then, GMM parameters are learned from the data. To this end, subsets

$$\mathcal{S}(a,k) = \{x_i \in \mathbf{x} \mid a = a_i,\ k = k_i\}$$

of clustered pixels are determined. The means $\mu(a,k)$ and covariance $\Sigma(a,k)$ are estimated in standard fashion as the sample mean and covariance of pixel values in $\mathcal{S}(a,k)$. The corresponding mixing weights are estimated as

$$p(\theta|a,k) = \frac{\|\mathcal{S}(a,k)\|}{\sum_k \|\mathcal{S}(a,k)\|}.$$

Finally, a global optimization using the minimum cut algorithm is carried out. This process is iterated until the change in the overall energy is below a threshold. The procedure is summarized in Algorithm 4.

6.5 Hierarchical Approach

In this section a multilevel banded heuristic for the presented Intelligent Scissors as well as graph cut based landform mapping approaches is proposed. The basic idea is to exploit the hierarchical structure of the quadtree to accelerate the segmentation. The segmentation is started on a coarse level of the quadtree hierarchy and then successively refined until the finest level is reached. On the starting level the segmentation is carried out using the algorithms presented in the previous sections. The obtained boundary is then

Algorithm 4 Iterative Graph Cut

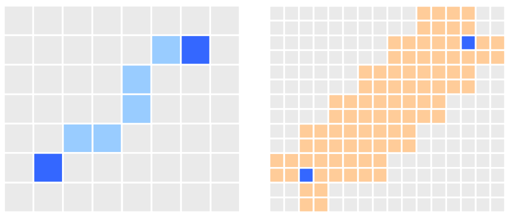
```
initialize GMMs from user input
repeat
    assign GMM components to pixels:
        $k_i^{t+1} \leftarrow \arg\min_{k \in \{0,\dots,K\}} R(x_i, a_i^t, k, \theta^t)$

    learn GMM parameters from data:
        $\theta^{t+1} \leftarrow \arg\min_\theta \sum_i R(x_i, a_i^t, k_i^{t+1}, \theta)$

    estimate segmentation: use min cut to solve
        $\mathbf{a}^{t+1} \leftarrow \arg\min_{\mathbf{a}} E^t(\mathbf{x}, \mathbf{a}, \mathbf{k}^{t+1}, \theta^{t+1})$

until $E^{t-1} - E^t < threshold$
```

propagated to the next higher resolution level. There, the segmentation is performed only within a narrow band surrounding the propagated boundary from the coarser level. This procedure is repeated until the highest resolution level (or a user-defined level) is reached. Since the algorithms only run on the subgraph that comprises the narrow band, the necessary computations at the fine resolution are significantly lower than running it on the full graph. Since only the subgraph has to be established, memory consumption is reduced as well.

This approach has already been used to accelerate graph cut based segmentation on images [Lombaert *et al.* 2005]. It is inspired by the well-known narrow band algorithm in level set methods [Adalsteinsson & Sethian 1995] as well as the multilevel graph partition method [Karypis & Kumar 1998]. The approach is particularly suited for working on a quadtree representation of image data because in this case a multiresolution representation of the data already exists and does not need to be created in a preprocessing step.

Assume there are $\mathbf{l} = (1, \dots, L)$ levels in the quadtree, where 1 corresponds the finest resolution level and L to the coarsest. Let l_s and l_e denote the level, where the segmentation starts and ends, respectively, with $1 \leq l_e \leq l_s \leq L$. Further, let $G^l(\mathcal{V}^l, \mathcal{E}^l)$ denote the graph that is constructed on level l and B^l the boundary resulting from the segmentation on the l-th level. In the Intelligent Scissors approach the boundary consists of the nodes corresponding to the optimal path. In the graph cut method the boundary is defined by the nodes that are at the border between foreground and background regions.

The segmentation starts by propagating the user inputs down to each level until l_s. For the Intelligent Scissors method this comprises the seed point and the free point, whereas for the graph cut method the user hints


(a) Banded Intelligent Scissors

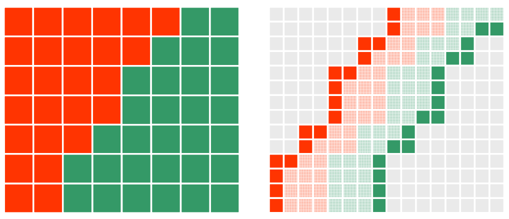

(b) Banded Graph Cut

Figure 6.7: Hierarchical Refinement. (a) A shortest path (blue) together with a narrow band surrounding it is propagated to the next higher level of the quadtree hierarchy. (b) The result of a graph cut segmentation is propagated to the next level. On the next level only pixels close to the border of foreground and background regions are considered. The outermost pixels are used as hard constraints.

marking foreground and background regions, respectively. After that, the respective segmentation algorithm is performed on l_s.

Suppose now that the current level is l and the boundary $B^l \subset \mathcal{V}^l$ is given. The boundary is then propagated to the next higher level. For a node $v \in \mathcal{V}^l$, let $prop(v) \subset \mathcal{V}^{l-1}$ denote the four corresponding nodes on the next higher resolution level. However, in addition to the boundary itself, also pixels inside a narrow band of with d surrounding the boundary are propagated to the next level

$$prop(B^l) = \left\{ prop(v) \in \mathcal{V}^{l-1} \mid \exists v' \in B^l \ : \ \|v - v'\|^2 < d, \ v \in \mathcal{V}^l \right\}.$$

From this band then the graph $G^{l-1}(\mathcal{V}^{l-1}, \mathcal{E}^{l-1})$ is built, where nodes are created for the vertices of $prop(B^l)$ and edges are created between neighboring nodes as in the standard algorithm. The distance parameter plays an important role in practice. If d is small, the algorithm may not be able to recover the full details of object boundaries with high shape complexity or high curvature. By contrast, if d is large, the computational benefits are reduced and the wider band may also introduce additional outliers far away from the desired object boundary. In practice, the use of $d = 1$ presents a reasonable compromise.

While on the start level the incremental Intelligent Scissors based approach is performed as presented in previous section, on the higher levels the optimal path from the seed to the free point is computed on the graph created from the propagated result. For the graph cut additional hard constraints at the higher levels are provided by assigning the nodes at the inner border to the foreground and the nodes at the outer border to the background (see Figure

Algorithm 5 Hierarchical Segmentation

```
propagate user input to level $l_s$
create $G^{l_s}(V^{l_s}, E^{l_s})$
**for** $l = l_s, \ldots, 1_e - 1$ **do**
    $\mathcal{B}^l \leftarrow$ solve on $G^l(V^l, E^l)$
    $prop(\mathcal{B}^l) \leftarrow$ project boundary pixel $\mathcal{B}^l$ to next level $l - 1$
    $G^{l-1} \leftarrow$ create from $prop(\mathcal{B}^l)$
    (graph cut: add additional hard constraints at borders)
**end for**
$\mathcal{B}^{l_e} \leftarrow$ solve on $G^{l_e}(V^{l_e}, E^{l_e})$
```

6.7). This procedure is repeated until the finest resolution level is reached, yielding the final boundary B^{l_e}. The method is summarized in Algorithm 5.

6.6 Revised Landform Mapping

A limitation of all landform mapping tools working on aerial imagery and elevation data only is that they can not be used to perform an accurate mapping at steep slopes. The reason for this is that steep slopes are only sparsely sampled in orthoimages and therefore lack information needed for a detailed mapping. However, even if additional imagery from different perspective is available, a suitable representation for landform mapping is required. In view of this, the landform mapping tools presented in the previous sections are revised in order to be able to incorporate imagery in addition to the aerial photographs.

To provide additional information at steep slopes, photos are registered to the textured DEM with the method described in chapter 4. The different georegistered photos are then composited on the DEM as presented in chapter 5. The segmentation is then performed on the texture patches resulting from the composition.

Since the reparameterized texture patches of a quadtree tile are also represented as images on a regular grid, creating the graph structure for them is as straightforward as in the standard case. However, since neighboring patches are parameterized separately, adjacent patch borders do not coincide exactly (although they are shaped similarly). Therefore, finding adjacent nodes along the patch borders and creating edges between them has to be performed explicitly.

Let $S \subset \mathbb{R}^3$ be the surface patch of a quadtree tile and

$$\mathcal{M}_S = \{\mathcal{V}_S, \mathcal{T}_S, (p_v)_{v \in \mathcal{V}_S}\}$$

its respective triangulation with a set of vertices $\mathcal{V}_S$, triangles $\mathcal{T}_S \in \mathcal{V}_S^3$ and vertex locations $(p_v) \in S$. Further, let $\mathcal{M}_o$ be the triangulation obtained by ortho-projection in the parameter domain $\Omega_o \subset \mathbb{R}^2$, and $\mathcal{M}_r$ be the triangulation obtained by the reparameterization in the parameter domain $\Omega_r \subset \mathbb{R}^2$, respectively. Further, let

$$\begin{array}{rcl} \phi_o : \Omega_o \subset \mathbb{R}^2 & \rightarrow & S \\ (u,v) & \mapsto & \phi_o(u,v) \end{array}$$

and ϕ_r, respectively, denote the mappings from the respective parameter spaces into S. Given a point $t = (u,v)$ in the parameter domain Ω_r at the border of patch S, the objective is then to find its neighbor t' in the parameter domain Ω_r' of the adjacent patch S' (see Figure 6.8). Since t is a border vertex it is located on an edge and can therefore be represented as the linear combination of the respective edge vertices. Assuming the edge vertices are $t_i, t_j \in \Omega_r$ of the triangulation $\mathcal{M}_r$, then t can be expressed as

$$t = \lambda t_i + (1-\lambda) t_j.$$

Since the parameterization is linear over the triangles, t can be transformed into S as

$$\begin{array}{rcl} \phi_r(t) & = & \lambda\phi_r(t) + (1-\lambda)\phi_r(t) \\ & = & \lambda p_i + (1-\lambda) p_j = p, \end{array}$$

where $p_i, p_j \in S$ are the corresponding edge vertices in world space and $p \in S$ is the interpolated vertex. From the world space representation p can be transformed into the parameter space Ω_o as

$$\begin{array}{rcl} \phi_o^{-1}(p) & = & \lambda\phi_o^{-1}(p_i) + (1-\lambda)\phi_o^{-1}(p_j) \\ & = & \lambda v_i + (1-\lambda) v_j = v, \end{array}$$

where $v_i, v_j \in \mathcal{V}_o$ are vertices of the triangulation $\mathcal{M}_o$ of the parameter domain Ω_o.

Let this mapping from Ω_o to Ω_r be denoted by

$$\gamma : \Omega_o \rightarrow \Omega_r$$

and γ^{-1} be the inverse mapping.

To perform these mappings efficiently, they are implemented as a lookup

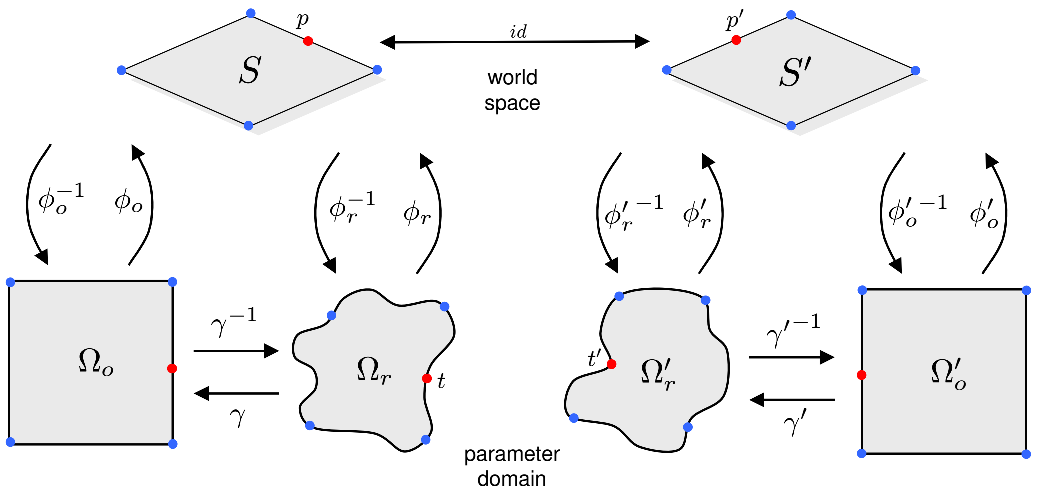


Figure 6.8: Patch Representations. Two neighboring tiles of the quadtree are shown in the different domains and with the corresponding mapping functions between them.

table. The lookup tables are generated by rendering the respective triangulation color coded into an offscreen buffer. To create the lookup table for γ, the triangulation $\mathcal{M}_r$ is rendered and each vertex is assigned its orthotexture coordinates as color as color. The colors are interpolated over the triangles. Analogously, the lookup table for γ^{-1} is created by rendering the triangulation $\mathcal{M}_o$ and each vertex is assigned the texture coordinates obtained by the reparameterization as color.

Let Ω_r and Ω_r' denote are parameter domains of the respective graphs G and G' of neighboring patches. Further, let $\mathcal{B}$ and $\mathcal{B}'$ denote the set of border nodes along the common border of G and G'. The edges of the joint graph are then obtained as the union of the edges of both graphs and additional edges between the closest border nodes. Thus,

$$\mathcal{E} \cup \mathcal{E}' \bigcup_{b_i \in \mathcal{B}} \{(b_i, c(b_i, \mathcal{B}'))\} \bigcup_{b_i' \in \mathcal{B}} \{(b_i', c(b_i', \mathcal{B}))\},$$

where

$$c(b_i, \mathcal{B}') = \operatorname*{arg\,min}_{b_j' \in \mathcal{B}'} \left\|\gamma(b_i) - \gamma'(b_j')\right\|^2.$$

This is similar to using a 4-neighborhood system. To mimic an 8-neighborhood system, additional edges to the second and third closest border nodes have to be inserted. Although there is a no longer a one-to-one mapping of pixels at patch borders, the number of nodes at corresponding patch borders should be

Algorithm 6 ConnectPatchBorders

```
Input:
  graphs $G$ and $G'$

Output:
  set of edges $\mathcal{E}(G, G')$

$\mathcal{E}(G, G') \leftarrow \emptyset$
$\mathcal{B}, \mathcal{B}' \leftarrow$ detect border nodes of $G$ and $G'$
$\gamma, \gamma' \leftarrow$ create lookup tables
for $b_i \in \mathcal{B}$ do
  $\mathcal{E}(G, G') \leftarrow \mathcal{E}(G, G') \cup \{(b_i, c(b_i, \mathcal{B}'))\}$
end for
for $b'_i \in \mathcal{B}'$ do
  $\mathcal{E}(G, G') \leftarrow \mathcal{E}(G, G') \cup \{(b'_i, c(b'_i, \mathcal{B}))\}$
end for
```

similar as the reparameterization aimed at minimizing stretch. The irregular graph structure at the borders does not affect the segmentation algorithms, since neither of them demands a fixed number of neighbors per node or a grid graph structure.

6.7 Results and Discussion

For Turtmann Valley a detailed geomorphological map and a GIS database of landform polygons exists [Otto & Dikau 2004][Otto 2006]. The map is based on field work and manual mapping on the HRSC data using ArcGIS. In order to asses the applicability of the presented landform mapping tools, different landform types have been re-mapped with them and compared to the manually mapped objects.

In evaluating the performance of boundary construction methods, typically three factors are considered to be of prime importance: speed, repeatability and accuracy. Accuracy measures the degree of agreement of the extracted boundary with the actual boundary. In general, the use of better cost functions in the algorithms could increase accuracy in tracking strong edge features. Unfortunately, it is hard to define the actual boundary itself since it is not always composed of the edges with the strongest features. Thus, a manually traced boundary is often considered to be a good approximation of the target boundary and is therefore often used as ground truth. However, the credibility of manual tracing is still questionable, since its re-

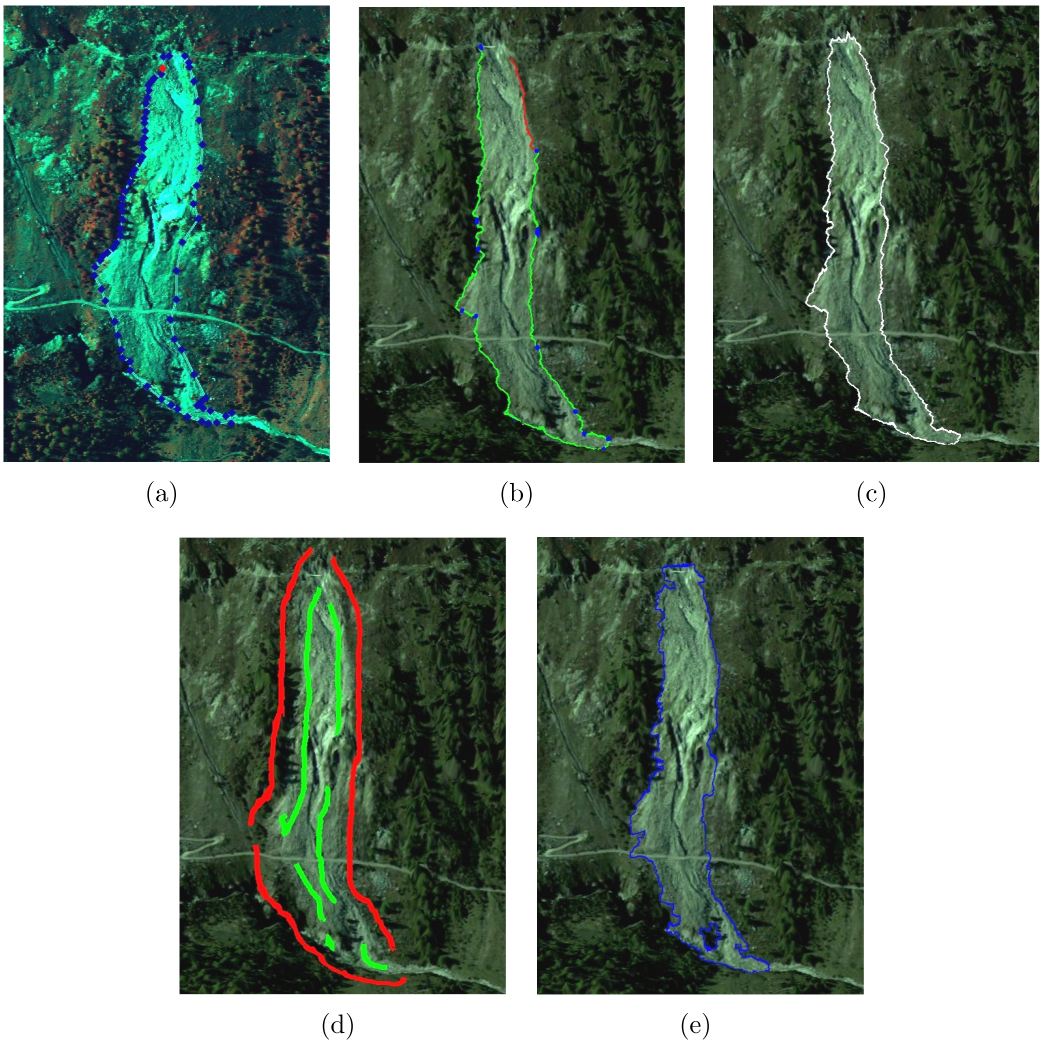


Figure 6.9: **Comparison**. Traditional manual mapping in 2D compared and the presented mapping tools. (a) Manual Mapping in ArcGIS. (b) Intelligent Scissors (in process). (c) Intelligent Scissors (finished). (d) Graph Cut Initialization. (e) Graph Cut Result.

peatability is usually worse than that of (semi-)automatic methods. What is more, manual tracing is usually never performed with pixel accuracy. Thus, it typically presents only a simplified version of the actual boundary that is too coarse to serve as ground truth when evaluating accuracy of pixel exact methods.

In view of this, a qualitative (i.e., visual) comparison of the object boundaries defined with the different methods is presented, combined with the

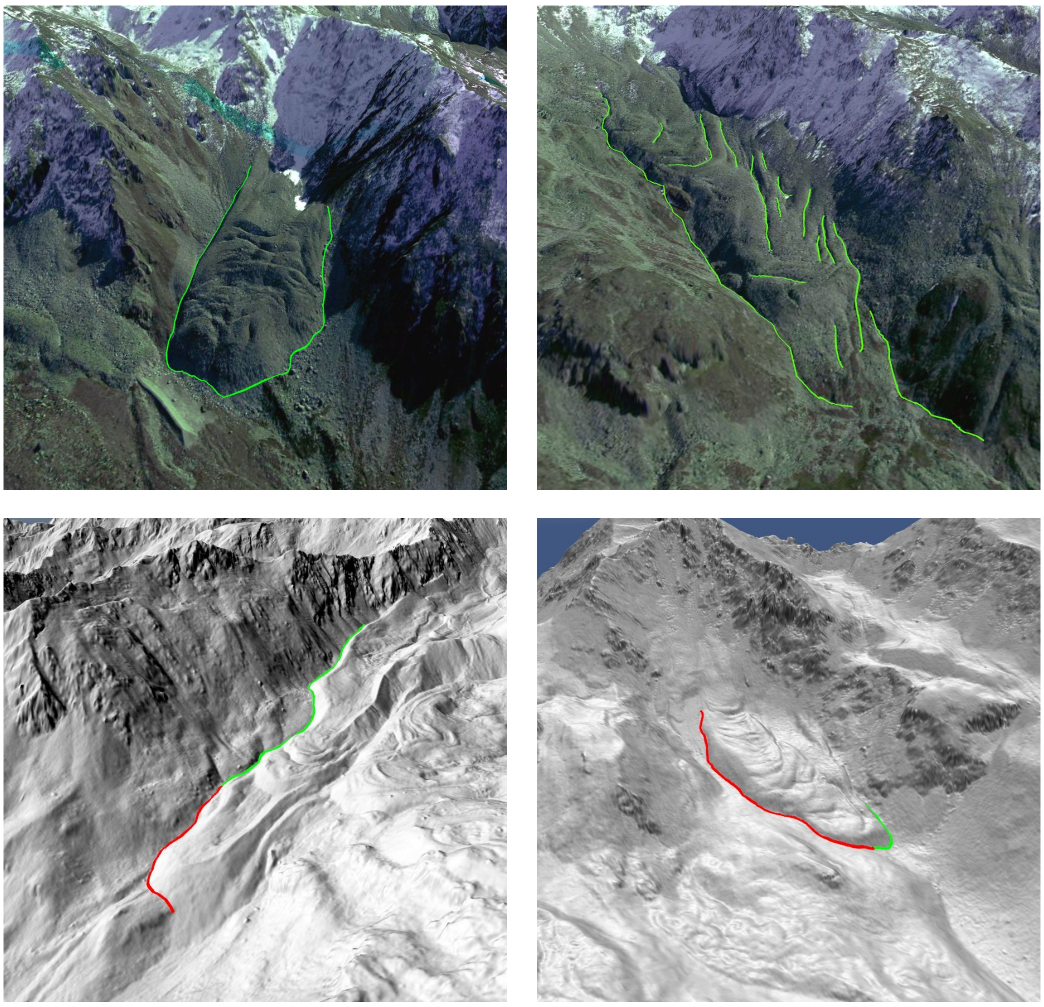


Figure 6.10: Examples of the Intelligent Scissors Based Method. The top row shows results obtained by mapping on the textured DEM. By contrast, the bottom row depicts mapping based on shaded relief (left) and slope (right).

number of nodes in the resulting boundary as an indicator of accuracy. The amount of user effort in terms of time and mouse clicks required to extract an object boundary is used as an indicator for digitizing speed (i.e., the number of nodes placed in ArcGIS, the number of seed points planted with Intelligent Scissors and the number of mouse strokes used for the graph cut initialization).

Figure 6.9 depicts a shallow landslide on the western trough slope of the valley caused by the failure of a drainage pipe. The landslide area in Figure 6.9 (a) was digitized manually in ArcGIS with 86 nodes. Using the Intelligent

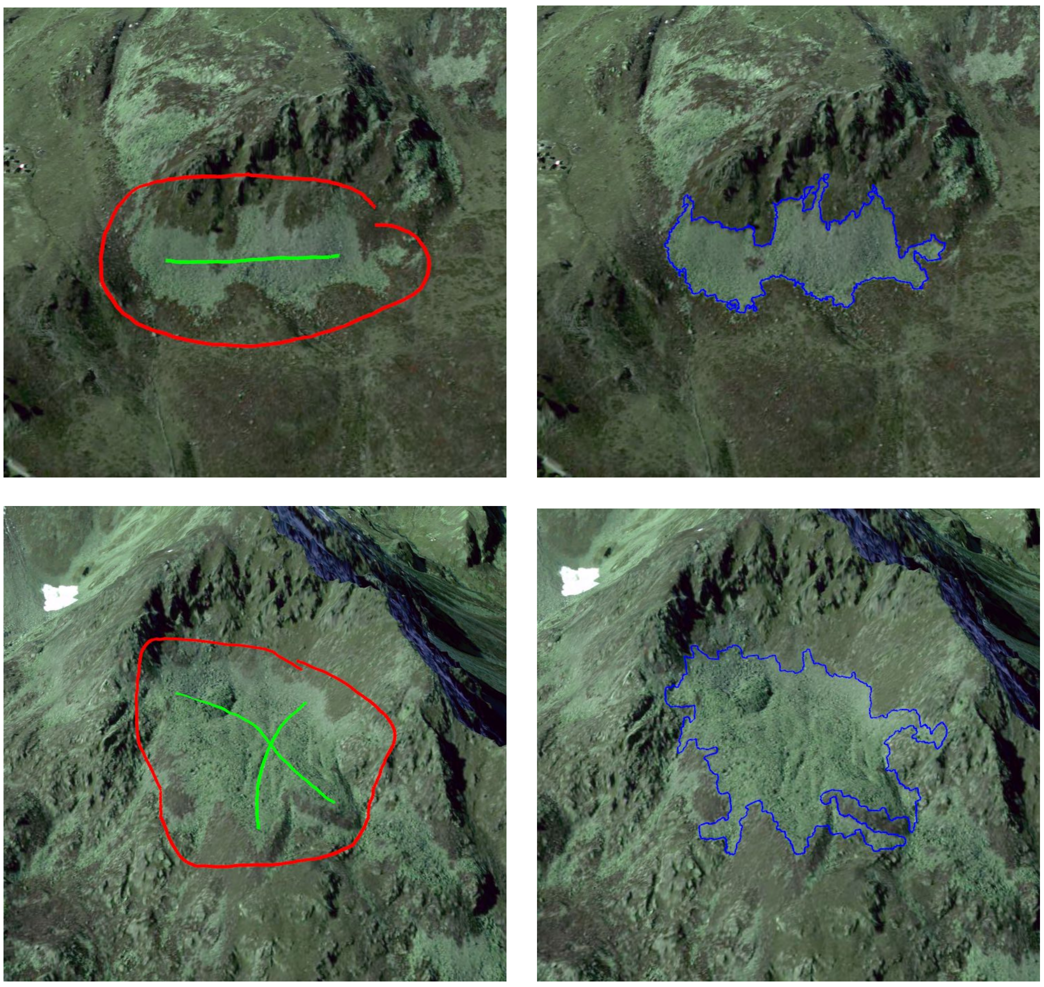


Figure 6.11: Examples of the Graph Cut Based Method. The left column shows the initialization by the user and the right column the obtained segmentation results.

Scissors tool, only about 12 nodes (seed points) need to be placed as shown in Figure 6.9 (b) and (c). Using the graph cut method, as shown in Figure 6.9 (d), the user marks the object of interest with green and the background with red, respectively. Both tools make full use of the high-resolution of the dataset resulting in polygons with up to several thousand nodes. The high number of nodes may be impractical in terms of memory consumption, especially when the number of objects is high as well. To account for that, the number of nodes can be reduced using an error-controlled polyline simplification [Douglas & Peucker 1973].

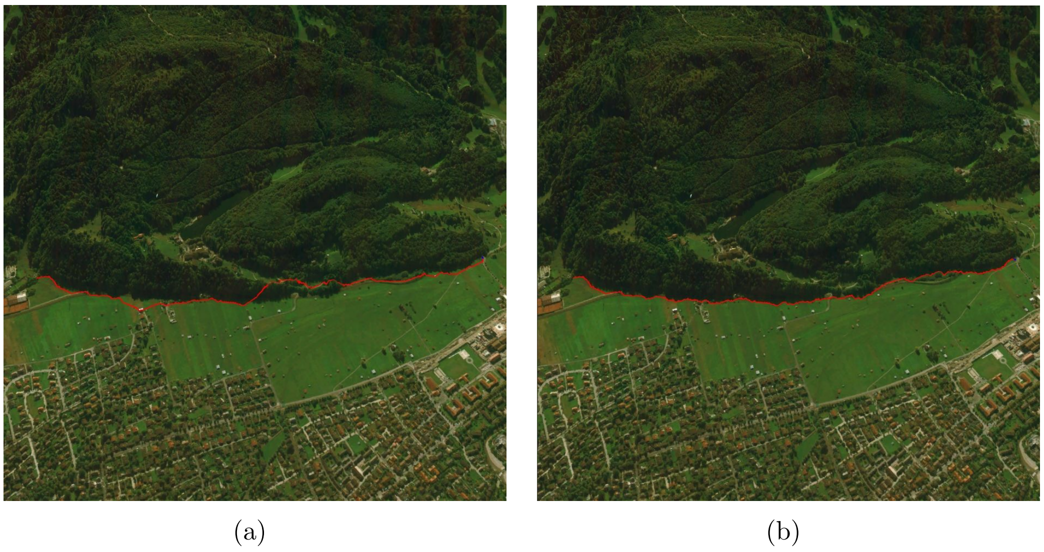


Figure 6.12: Hierarchical Mapping. In (a) the segmentation is performed using the finest resolution level only. The proposed boundary segment deviates from the desired boundary due to effects of high frequency features. In (b) the segmentation is started on the third finest level and is then successively refined. The proposed boundary is as desired because on the starting level the disturbing high frequency features do not exists and during the refinement on the higher levels they lie outside the considered band.

From visual inspection, the Intelligent Scissors tool defines the most precise boundaries of the landslide area compared to the manual mapping. In contrast, the object boundary created by the graph cut method appears to be more jagged, which however depends on the chosen influence of the smoothing term and can easily be adapted by changing it (although this requires a re-run of the graph cut optimization it is very fast because the current segmentation can be used as initialization that is already very close to the optimum). Some areas are included within the polygon that have not been mapped manually, for example parts of the gravel road. Other areas, for example the darker parts within the polygon are not considered, although they clearly belong to the landslide area. In this case this behavior is unwanted, however, this does not always have to be the case. Thus, some post-editing of the graph cut derived object is required in this case to obtain a satisfying result. Both methods produce a jagged section in middle part of the left boundary (Figure 6.9 (c) and (e)), where the landslide area crosses the forest. Here, the tools suffer from the lighting situation causing the left boundary to be defined by the shadow of the trees. This effect does not occur when

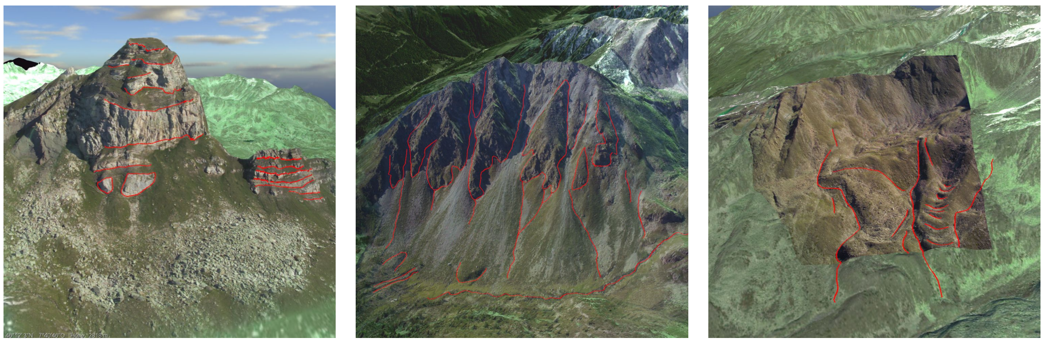


Figure 6.13: Mapping in steep slopes. Examples of the mapping at steep slopes using registered photos.

digitized manually, as the user would notice the shadowing effects and compensate for them.

Despite the aforementioned limitations, the mapping of landforms with the two tools proves to be very accurate and fast in general. Within a few steps, high resolution objects, represented by a high number of nodes and accurate boundaries can be generated within short time. Additionally, the 3D visualization environment offers numerous ways to perceive the landform and its boundaries not only on the aerial image, but including the landform surface structure. However, apart from complex geomorphological landforms, a segmentation of other objects or features like houses, roads, lakes or fields with clear boundaries can very easily be performed. Restrictions observed in full automatic classifications using remote sensing techniques do not apply, as the user is in full control of the segmentation steps.

Due to technical differences each tool has individual advantages. The Intelligent Scissors approach allows a more supervised mapping, as the location of the automatic path can be controlled by the placement of intermediate seed points. Due to its segmentation principle, the Intelligent Scissors approach is especially suited for the extraction of linear features and boundaries, like gullies, debris flow tracks, ridges, steps, or moraine and rock glacier boundaries. However, a limitation of Intelligent Scissors is that for highly textured or un-textured regions many alternative "minimal" path exists. Thus, in such regions many seed points may be necessary. In contrast, the graph cut tool is less controlled and very efficient for mapping of closed objects with complex and fine-grained borders. Another advantage of the graph cut method is the possibility to extract more than one object at the same time. This way, several debris fields or snow patches, for example, can be extracted simultaneously with a few strokes.

Both methods allow for efficient post-editing. Object boundaries can be changed by editing arbitrary single nodes after the segmentation is terminated. In addition to that, the graph cut method allows further refinements of the segmentation result by applying additional hints followed by a re-run of the optimization.

In general, boundary identification on aerial photography is influenced to a great extent by the lighting conditions during the acquisition process. As a result, segmentation results are biased, especially by shadows, causing the segmentation algorithm to follow shadow boundaries instead of the true object boundaries. One approach to overcome this problem is to consider the elevation data in the boundary estimation procedure in addition to the aerial photography. For the Intelligent Scissors method, the mapping can also be performed based on shaded relief, slope or curvature computed from the DEM (see Figure 6.10). However, high resolution elevation data is required for this which is not as often available as high resolution aerial imagery.

As a result of our incremental and hierarchical algorithm, segmentation can be performed at interactive speed nearly independent of the size of the underlying dataset. Despite accelerating the segmentation, another advantage of the hierarchical approach is shown in Figure 6.12. A forest boundary is segmented twice, starting at different levels of the quadtree. In (a) the segmentation is performed solely on the finest resolution level resulting in several deviances from the desired boundary caused by fine scale features. Many seed points need to be placed in order to obtain the desired result. However, in (b) the segmentation is started from three levels above the finest resolution leading to the correct boundary since the disturbing small scale features are not present at the higher level.

Figure 6.13 shows results obtained using the revised landform mapping. Using georegistered photos, a detailed mapping becomes possible at steep slopes which was not possible using the aerial imagery only.

CHAPTER 7

Conclusions and Future Work

7.1 Conclusions

The presented thesis used computer graphic methodologies to enrich data modeling and analysis in the geoscience domain with a particular focus on geomorphological applications.

Chapter 3 dealt with the *visualization of vector data* on textured DEMs. In contrast to traditional 2D map representations commonly used in GIS, the visualization of vector data on digital elevation models provides a more natural and intuitive representation. Particular advantages of this representation are that the size and shape of landforms is not distorted as in the 2D case, and that the morphology of landforms becomes directly apparent.

The presented texture-based approach rasterizes the vector shapes into an offscreen buffer in each frame, which is then bound as a texture and projected onto the terrain geometry. The on-the-fly generation of the texture has the advantage that neither a texture nor a complete texture pyramid has to be precomputed and loaded into memory. Instead, only the much more compact polygonal representation of the vector data is required. By applying a view-dependent reparameterization, perspective aliasing is significantly reduced and thus quality superior to a uniform parameterization is achieved.

In contrast, the presented stencil-based approach extrudes vector shapes to polyhedra. Then, these polyhedra are used to create a mask in the stencil buffer indicating the position of the vector shape with respect to the current view. In comparison to the texture-based technique, more primitives have to be rendered which makes the method slightly slower. However, since the method works in image space, it is per-pixel exact and does not suffer from projective aliasing artifacts that can, despite the reparameterization, still pose problems for the texture-based approach in very steep slopes. Both methods allow for interactive editing of vector data.

In chapter 4 an algorithm for the *registration* of uncalibrated photos to textured DEMs was presented. The proposed registration algorithm is independent of user-defined marker points on-site, which is especially important in high alpine environments where many places are difficult to access. However, if additional marker points are available, they can naturally and easily be included in the registration process. The presented method allows the user a targeted enhancement of existing terrain datasets using photos of areas of interest. Apart from increasing texture resolution significantly, particularly at steep slopes, it is also possible to add information in areas that are blurred or occluded in the aerial imagery due to shadows, snow, or clouds, for example. The main idea of the registration algorithm is to make use of the given textured DEM during an incremental structure from motion optimization. This way, the robustness and effectiveness of the registration is improved.

Chapter 5 presented a method for *compositing* georegistered photos on a textured DEM. The main idea is to reparameterize the terrain surface and perform the blending in texture space. The reparameterization is carried out in a preprocessing step. After that, visibility is determined for each view in order to identify the regions on the terrain surface valid for texturing with the respective image. With this information at hand, the registered photos are then combined in a two-step procedure. First, color distributions in the images are adapted in order to remove large scale color and lightness shifts. Second, images are blended using a weighted pyramid blending approach in texture space induced by the reparameterization of the terrain geometry. Finally, the new textures are inserted into the quadtree data structure of the terrain engine to allow real-time rendering. Using the described compositing approach smooth transitions between the images on the terrain surface despite illumination differences are realized while at the same time high frequency details are preserved.

Chapter 6 dealt with *semi-automatic landform mapping* tools that enable the mapping of geomorphological objects directly on the textured DEM. Using a 3D visualization allows the user to perceive the landform structure in its natural form similar to the perception in the field during the mapping process. Moreover, by navigating in the 3D environment landforms can be inspected from arbitrary views including perspectives hardly possible in nature. In addition to an enhanced landform visualization, the proposed landform mapping tools assist the user in specifying geomorphological objects quickly and accurately by using semi-automatic image segmentation techniques. This relieves the user from defining landform borders exactly, instead only vague hints that roughly indicate the location of the boundary have to be provided. Using incremental and hierarchical techniques, the presented methods achieve interactive performance even on very large textured DEMs.

Standard landform mapping tools working on aerial imagery and elevation data only are not capable of performing an accurate mapping at steep slopes. The main reasons for this is that orthoimages neither provide detailed enough information at steep slopes nor are they suited for representing them. With this in mind, the presented landform mapping tools were revised in order to be able to incorporate additional imagery from different perspectives. To provide the additional information at steep slopes, photos are registered to the textured DEM using the method described in chapter 4. The different georegistered photos are then composited on the DEM as presented in chapter 5. The actual segmentation is then performed on the resulting reparameterized texture patches. Using the presented method, a detailed and accurate mapping of steep slopes becomes feasible.

7.2 Future Work

Possible future work concerning vector data could aim at consistency issues of vector data and digital elevation models. Depending on the application, vector data might be used to impose constraints on the terrain geometry (e.g., areas marked as lakes have to be flat). On the other hand, 2D vector shapes could be augmented with height values from the DEM and this way extended to 2.5D surface. Combined with additional information about the volumes of sediment storages, for example from geophysical surveying, even volumetric representations of geomorphological objects could be obtained. Within this context, the development of suitable algorithms for the combined visualization of volumetric geomorphological objects and the terrain surface is an interesting topic for future research.

The focus in the presented registration approach was to align the photos as best as possible to a given terrain dataset. However, the ability to compute accurate camera parameters opens the door for techniques that compute dense surface shape models, such as multi-view stereo [Goesele *et al.* 2007]. In the future it might be interesting to investigate to what extend the geometric resolution of a DEM could be improved with such methods. Also, removing the required manual initialization by the user in order to end up with a fully automatic registration poses a challenge for future research.

Regarding the compositing of images an interesting direction for future work is the opportunity to build time-varying 3D models that can serve to pull together large collections of images pertaining to the appearance, evolution, and events surrounding one place over time. Also, a detection and removal of artifacts caused by moving objects or unmodeled geometry poses a challenge for future work.

APPENDIX A

Appendix: Registration

A.1 Projective Geometry

A point in projective n-space $\mathbb{P}^n$ is a $(n+1)$-*homogeneous* vector of coordinates $\mathtt{x} = (x_1, \ldots, x_{n+1})^\mathsf{T}$ with $x_i \in \mathbb{R}$ and at least one $x_i \neq 0$. Two homogeneous vectors $\mathtt{x}$ and $\mathtt{y}$ are called equivalent $\mathtt{x} \sim \mathtt{y}$ if and only if there exists a nonzero scalar λ, such that $x_i = \lambda y_i$, for every i $(1 \leq i \leq n+1)$. The affine space $\mathbb{R}^n$ can be embedded isomorphically in $\mathbb{P}^n$ by the standard injection $(x_1, \ldots, x_n,)^\mathsf{T} \mapsto (x_1, \ldots, x_n, 1)^\mathsf{T}$. Affine points can be recovered from projective points with $x_{n+1} \neq 0$ by the mapping

$$(x_1, \ldots, x_{n+1})^\mathsf{T} \sim \left(\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}, 1\right)^\mathsf{T} \mapsto \left(\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}\right)^\mathsf{T}.$$

A projective point with $x_{n+1} = 0$ corresponds to an ideal *point at infinity* in the $(x_1, \ldots, x_n)$ direction in affine space. The set of all such "infinite" points satisfying the homogeneous linear constraint $x_{n+1} = 0$ acts like a hyperplane, called the *hyperplane at infinity.*

A.1.1 Projective plane

The projective plane is the projective space $\mathbb{P}^2$. A point of $\mathbb{P}^2$ is represented by a 3-vector $\mathtt{x} = (x, y, w)^\mathsf{T}$. A line $\mathtt{l}$ is also represented by a 3-vector. A point is located on a line if and only if

$$\mathtt{l}^\mathsf{T}\mathtt{x} = 0.$$

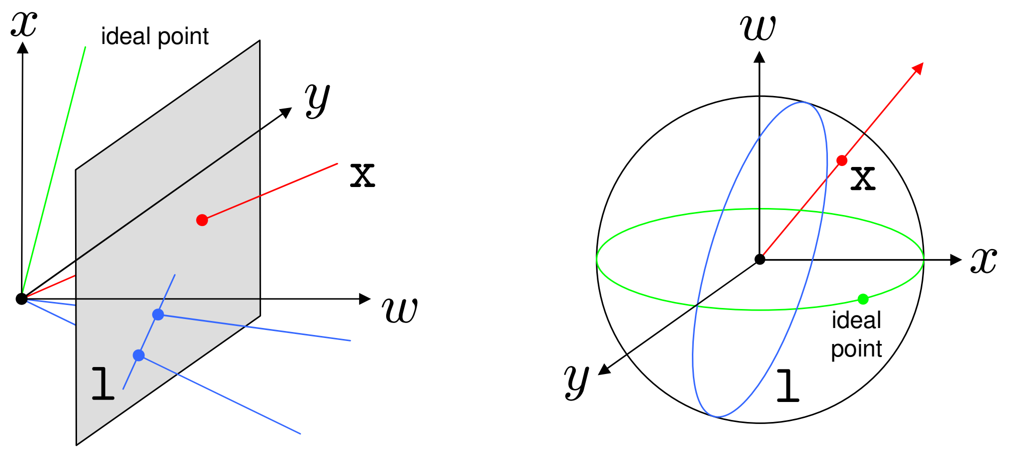


Figure A.1: Models of the Projective Plane. Points $\mathtt{x}$ and lines $\mathtt{l}$ of $\mathbb{P}^2$ can be represented by rays and planes through the origin in $\mathbb{R}^3$. *Left*: By projecting onto $w = 1$ points and lines in the projective plane can be interpreted as points and lines in the $w = 1$ plane, whereas the xy-plane represent the *hyperplane at infinity*. *Right*: By projection onto the unit sphere, points in the projective plane can be visualized as points on the unit sphere and lines can be represented as great circles obtained from the intersection of the corresponding plane with the sphere.

However, this equation can also be interpreted as expressing that the line $\mathtt{l}$ passes through the point $\mathtt{x}$. The symmetry in the equation reveals that there is no formal differences between points and lines in projective space, which is known as the *principle of duality*. A line $\mathtt{l}$ passing through points $\mathtt{x}_1$ and $\mathtt{x}_2$ is given by their vector product

$$\mathtt{l} \sim [\mathtt{x}_1]_\times \mathtt{x}_2 \text{ with } [\mathtt{x}_1]_\times = \begin{bmatrix} 0 & w_1 & -y_1 \\ -w_1 & 0 & x_1 \\ y_1 & -x_1 & 0 \end{bmatrix}.$$

A.1.2 Projective 3-space

Projective 3-space is the projective space $\mathbb{P}^3$. A point of $\mathbb{P}^3$ is represented by a 4-vector $\mathtt{X} = (x, y, z, w)^\mathsf{T}$. In $\mathbb{P}^3$ the dual entity of a point is a plane. A point $\mathtt{X}$ is located on a plane π if and only if

$$\pi^\mathsf{T}\mathtt{X} = 0.$$

A line can be given by the linear combination of two points $\lambda_1\mathtt{X}_1 + \lambda_2\mathtt{X}_2$ or by the intersection of two planes $\pi_1 \cap \pi_2$.

A.1.3 Transformations

A *planar projective transformation* or *homography* of $\mathbb{P}^2 \to \mathbb{P}^2$ is a linear transformation on homogeneous 3-vectors represented by a nonsingular 3×3 matrix $\mathbf{H}$

$$\mathtt{x} \mapsto \mathtt{x}' \sim \mathbf{H}\mathtt{x}.$$

The matrix $\mathbf{H}$ is homogeneous and has eight degrees of freedom (nine elements minus one for overall scaling). Similarly, a projective transformation acting on $\mathbb{P}^3$ is a linear transformation on homogeneous 4-vectors represented by a nonsingular 4×4 matrix $\mathbf{P}$

$$\mathtt{X} \mapsto \mathtt{X}' \sim \mathbf{P}\mathtt{X}.$$

The matrix $\mathbf{P}$ is homogeneous and has 15 degrees of freedom (16 elements minus one for overall scaling).

Group	DOF 2D/3D	Matrix	Transformations	Invariants
Projective	8/15	$\begin{pmatrix} \mathbf{A} & \mathtt{t} \\ \mathtt{v}^\mathsf{T} & 1 \end{pmatrix}$	perspective projection	cross-ratio, incidence
Affine	6/12	$\begin{pmatrix} \mathbf{A} & \mathtt{t} \\ \mathtt{0}^\mathsf{T} & 1 \end{pmatrix}$	non-uniform scaling, shear	parallelism
Similarity	4/7	$\begin{pmatrix} s\mathbf{R} & \mathtt{t} \\ \mathtt{0}^\mathsf{T} & 1 \end{pmatrix}$	uniform scaling	ratio of length, angles
Euclidean	3/6	$\begin{pmatrix} \mathbf{R} & \mathtt{t} \\ \mathtt{0}^\mathsf{T} & 1 \end{pmatrix}$	translation, rotation	length

Table A.1: Hierarchy of Transformations. The matrix $\mathbf{A}$ is an invertible 2×2 or 3×3 matrix, $\mathbf{R}$ is a 2D/3D rotation matrix, $\mathtt{t}$ a 2D/3D translation vector and s a scalar. Transformations listed higher in the table are able to produce all transformations of the ones below. The last column summarizes the invariants a specific group has in addition to the ones below.

The projective transformations form a group called *projective linear group*. It is possible to further subdivide this group into specializations with certain geometric properties (see Table A.1). Important subgroups are the *affine group*, consisting of matrices for which the last row is $(0, \ldots, 0, 1)$, and the *Euclidean group* for which the upper left hand matrix is orthogonal.

A.2 Camera Models

A camera describes a mapping from 3D world space to a 2D image. In the following cameras modeling *central projection* (see Figure A.2) are considered. All such cameras are specializations of the *general projective camera*. It is a linear function that maps world points to image points

$$\mathtt{x} \sim \mathbf{P}\mathtt{X},$$

where $\mathtt{x} \in \mathbb{P}^2$, $\mathtt{X} \in \mathbb{P}^3$, and $\mathbf{P}$ is the corresponding 3×4 projection matrix of rank 3. The most simple camera is the basic *pinhole camera*. The geometric process for image creation in a pinhole camera has been nicely illustrated by Dürer (see Figure A.3). In a Euclidean frame, the basic pinhole camera maps 3D points to 2D image points using central projection. This mapping

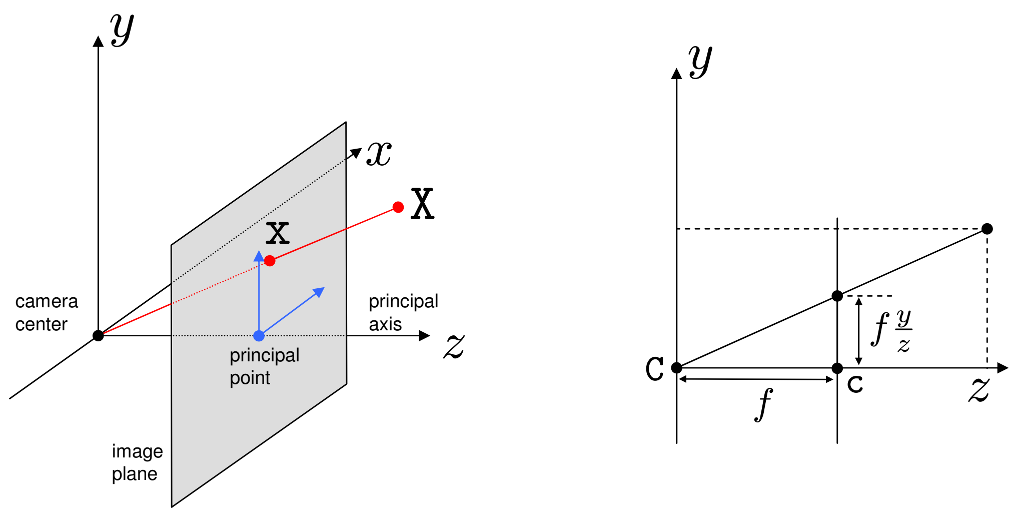


Figure A.2: Central Projection. A world point $\mathtt{X}$ is mapped by central projection to the image point $\mathtt{x}$ on the *image plane*. The *principal axis* passes through the center of projection $\mathtt{C}$ and is orthogonal to the image plane. It intersects the image plane in the *principal point* $\mathtt{c}$. The distance f between the center of projection $\mathtt{C}$ and the image plane is called *focal length*.

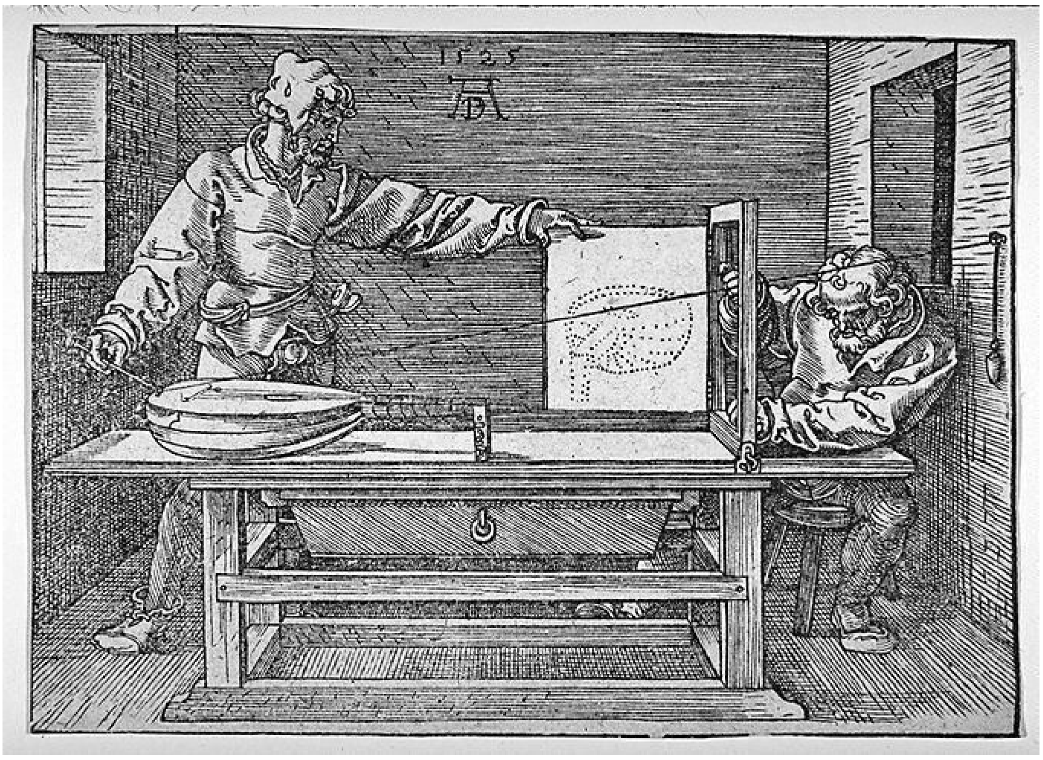


Figure A.3: Pinhole Camera Principle. "Man drawing a lute" by Albrecht Dürer (woodcut, 1525)

is completely defined by the camera center $\mathtt{C}$ and the image plane (or focal plane). Note that in Figure A.2 the image plane is located in front of the camera center. However, this is equivalent to having it behind the center and reflecting the image's x and y-axis. Assuming that the camera center lies in the origin of the world frame and that the image plane is located at $Z = 1$, central projection can be described algebraically as a non-linear mapping

$$(X, Y, Z)^\mathsf{T} \mapsto (x, y)^\mathsf{T} = (X/Z, Y/Z)^\mathsf{T},$$

called *perspective division*. In homogeneous coordinates, however, this can be expressed as a linear transformation

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{A.1}$$

that maps world coordinates to canonical camera coordinates on the image plane. Real cameras have, due to their construction, internal properties

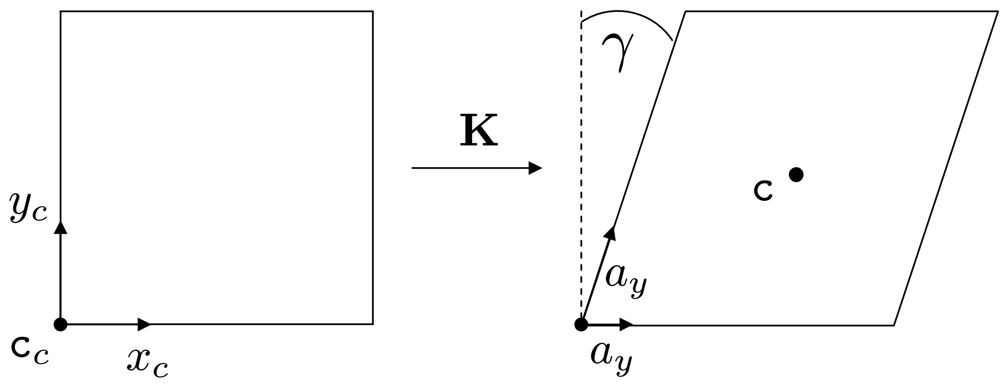


Figure A.4: Mapping from Canonical to Image Coordinates.

that further alter the canonical camera coordinates $(x_c, y_c, 1)^\mathsf{T}$ to real image coordinates $(x, y, 1)$. In the following possible transformations are described.

For real cameras the *focal length* f can vary, which is equivalent to an isotropic scaling along the image axes. Thus, the mapping becomes

$$(x_c, y_c)^\mathsf{T} \mapsto (x, y)^\mathsf{T} = (f x_c, f y_c)^\mathsf{T}.$$

The ratio a_x/a_y of a pixel's width a_x and height a_y is described by the *aspect ratio.* To account for that, an additional scaling by a_x and a_y has to be performed. Scaling by pixel dimensions can be combined with scaling by focal length as $\alpha_x = f/a_x$ and $\alpha_y = f/a_y$, which results in the mapping

$$(x_c, y_c)^\mathsf{T} \mapsto (x, y)^\mathsf{T} = (\alpha_x x_c, \alpha_y y_c)^\mathsf{T}.$$

In general the *principal point* c is not fixed at the origin but at some point $(\tilde{c}_x, \tilde{c}_y)$. The image coordinates of c are $c_x = \tilde{c}_x/\alpha_x$ and $c_y = \tilde{c}_y/\alpha_y$ and hence

$$(x_c, y_c)^\mathsf{T} \mapsto (x, y)^\mathsf{T} = (\alpha_x x_c + c_x, \alpha_y y_c + c_y)^\mathsf{T}.$$

Assuming that the image's x-axis is aligned with the canonical x-axis, the *skew* γ quantifies the angle between the image's y-axis and the canonical y-axis. This results in the final transformation

$$(x_c, y_c)^\mathsf{T} \mapsto (x, y)^\mathsf{T} = (\alpha_x x_c + c_x, \alpha_y y_c + s x_c + c_y)^\mathsf{T},$$

where $s = \alpha_y \tan \gamma$.

The complete set of transformations is illustrated in Figure A.4. In homogeneous coordinates the equation describing the transformation from canonical camera coordinates to image coordinates is

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix}. \tag{A.2}$$

This upper triangular, affine transformation matrix $\mathbf{K}$ is often called *calibration matrix* and contains five independent *intrinsic parameters*. For most CCD cameras it is reasonable to assume that the pixels are square and thus $\alpha_x = \alpha_y$. Furthermore, skew can be assumed zero for all practical purposes. Although the principle point can slightly vary around the center of the image, it is often assumed to be fixed in practice. For a standard camera with zoom and focus capabilities, focal length, however, can change from image to image. It should be mentioned that there might be additional effects that must be considered. The most prominent example is radial distortion, often encountered in cheap cameras and short focal lengths. Such distortion effects cannot be described by the above simple model and require higher dimensional equations.

Typically, points are expressed in a world coordinate frame instead of the camera frame. These two coordinate frames are related via a rotation and a translation. The corresponding Euclidean transformation matrix between the world and camera coordinate frame is

$$\mathtt{X}' \sim \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathtt{C}} \\ 0^\mathsf{T} & 1 \end{bmatrix} \mathtt{X}. \tag{A.3}$$

The parameters of $\mathbf{R}$ and $\mathtt{t} = -\mathbf{R}\tilde{\mathtt{C}}$ that relate the camera position and orientation to a world coordinate system are called *extrinsic parameters*. There are six degrees of freedom, three for the rotation $\mathbf{R}$ and three for the translation $\mathtt{t}$.

Concatenating the transformations described by Equations A.2, A.1 and A.3 yields

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}^{3\times 3} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathtt{C}} \\ 0^\mathsf{T} & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

which can be simplified to

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathtt{t} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

and further with $\mathbf{P} \sim \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathtt{t} \end{bmatrix}$ to

$$\mathtt{x} \sim \mathbf{P}\mathtt{X}.$$

The matrix $\mathbf{P}$ describes a finite projective camera. It has 11 degrees of freedom, the total of five intrinsic and six extrinsic parameters. In contrast to the finite projective camera, the pinhole camera model has only nine degrees of freedom because it assumes that pixels are square and that the skew is zero. The left 3×3 submatrix of $\mathbf{P}$ is nonsingular and conversely any 3×4 matrix $\mathbf{P}$ where the left 3×3 submatrix is nonsingular is the camera matrix of some finite projective camera. Being nonsingular, the submatrix can be decomposed by RQ decomposition into a product of matrices $\mathbf{KR}$, where $\mathbf{K}$ is an upper triangular matrix of the form in Equation A.2 and $\mathbf{R}$ is orthogonal. The remaining ambiguity in the decomposition can be resolved by demanding that $\mathbf{K}$ has positive diagonal entries.

A.3 Camera Estimation

The following section describes methods for estimating the camera projection matrix $\mathbf{P}$ from corresponding world space and image entities. This computation of the camera matrix is known as *resectioning*. The simplest kind of such a correspondence is that between a 3D point $\mathtt{X}$ and its image $\mathtt{x}$ under the unknown camera mapping. Given n correspondences $\mathtt{X}_i \leftrightarrow \mathtt{x}_i$ between 3D points $\mathtt{X}_i$ and 2D image points $\mathtt{x}_i$, a 3×4 camera matrix $\mathbf{P}$ is sought such that $\mathtt{x}_i = \mathbf{P}\mathtt{X}_i$. For each correspondence a system of equations

$$\begin{bmatrix} 0^\mathsf{T} & -w_i\mathtt{X}_i^\mathsf{T} & y_i\mathtt{X}_i^\mathsf{T} \\ w_i\mathtt{X}_i^\mathsf{T} & 0^\mathsf{T} & -x_i\mathtt{X}_i^\mathsf{T} \\ -y_i\mathtt{X}_i^\mathsf{T} & -x_i\mathtt{X}_i^\mathsf{T} & 0^\mathsf{T} \end{bmatrix} \begin{pmatrix} \mathbf{P}^1 \\ \mathbf{P}^2 \\ \mathbf{P}^3 \end{pmatrix} = 0$$

can be derived, where each ${\mathbf{P}^i}^\mathsf{T}$ is the i-th row of $\mathbf{P}$. Since the three equations are linearly dependent, usually only two of them are used. By stacking up the equations of the n correspondences a $2n \times 12$ matrix $\mathbf{A}$ can be obtained. The projection matrix $\mathbf{P}$ is computed from it by solving the set of equations $\mathbf{Ap} = 0$, where $\mathbf{p}$ is the vector containing the entries of $\mathbf{P}$.

The matrix $\mathbf{P}$ has 12 entries defined up to a scale and hence has 11 degrees of freedom. Thus, at least 11 equations are needed to solve for $\mathbf{P}$. Since each correspondence leads to two equations at least six correspondences are necessary. If a minimum number of correspondences are given, an exact solution exists that projects each $\mathtt{X}_i$ on the corresponding $\mathtt{x}_i$. However, if the data is not exact, for example due to noise in the point coordinates, and

$n \geq 6$ correspondences are given, then there will not be an exact solution to $\mathbf{Ap} = 0$. Therefore, instead of demanding an exact solution, typically an approximate solution is computed that minimizes an algebraic or geometric error. In the case of algebraic error a popular approach is to minimize $\|\mathbf{Ap}\|$ subject to the normalization constraint $\|\mathbf{p}\| = 1$. The solution can then be obtained as the right singular vector of $\mathbf{A}$ corresponding to the smallest eigenvalue. This algorithm is known as the *DLT algorithm*.

The result of the DLT algorithm depends on the coordinate frame in which the points are expressed. In fact, the result is not invariant under similarity transformations of the image. Therefore it is important to apply a *data normalization* in advance. The normalization makes the DLT invariant to similarity transformations and results in improved accuracy. To normalize the points $\mathtt{x}_i$ in the image, they are translated so that their centroid is at the origin and scaled so that their average distance from the origin is $\sqrt{2}$. The 3D points $\mathtt{X}_i$ are normalized similarly by translating their centroid to the origin and applying a scaling so that their average distance to the origin is $\sqrt{3}$.

The particular advantages of the DLT algorithm are a linear and thus unique solution and its computational cheapness. Often, the result of the DLT is used as a starting point for a non-linear minimization subject to an error function based on the geometric error

$$\sum_i \|\mathtt{x}_i - \mathbf{P}\mathtt{X}_i\|^2$$

to further refine the solution. Minimizing the geometric error requires the use of iterative techniques such as Levenberg-Marquardt. The normalized DLT algorithm followed by a nonlinear optimization, for example via Levenberg-Marquardt, is the *Gold Standard algorithm* for the estimation of the camera matrix.

The DLT algorithm described so far, computes a general projective camera matrix $\mathbf{P}$ from a set of 3D to 2D correspondences. Often, it is desirable to find a camera matrix subject to restrictive conditions of the camera parameters. Common assumptions are zero skew, square pixels and a known principle point. In this case, typically a linear DLT is used to find an initial, unconstrained camera matrix. Then, the fixed camera parameters are set to their desired values and a Levenberg-Marquardt optimization is performed in which only the variable parameters are optimized. Ideally, the assumed values of the fixed parameters will be close to the values obtained by the DLT. In practice, however, this is not always the case. Altering these parameters to their desired values in such a case might lead to difficulties in converging. Therefore, instead of setting the fixed parameters to their desired values, an additional cost function can be introduced that penalizes differences of the parameters from the desired values. The influence of these terms to the overall error is increased during the optimization so that the parameters are smoothly drawn to their desired values.

A.4 Two-view Geometry

The intrinsic projective geometry between two views is called *epipolar geometry.* The geometric entities involved in the epipolar geometry are illustrated in Figure A.5. Given two cameras the base line is the line joining the two camera centers $\mathtt{C}$ and $\mathtt{C}'$. Its intersections with the two image planes are the epipoles $\mathtt{e} \sim \mathbf{P}\mathtt{C}'$ and $\mathtt{e}' \sim \mathbf{P}'\mathtt{C}$. The image points $\mathtt{x} = \mathbf{P}\mathtt{X}$ and $\mathtt{x}' = \mathbf{P}'\mathtt{X}$ together with their respective camera centers $\mathtt{C}$ and $\mathtt{C}'$ define two rays that intersect in the common world point $\mathtt{X}$. These two rays and the base line lie in a common plane called epipolar plane. The intersections of the epipolar

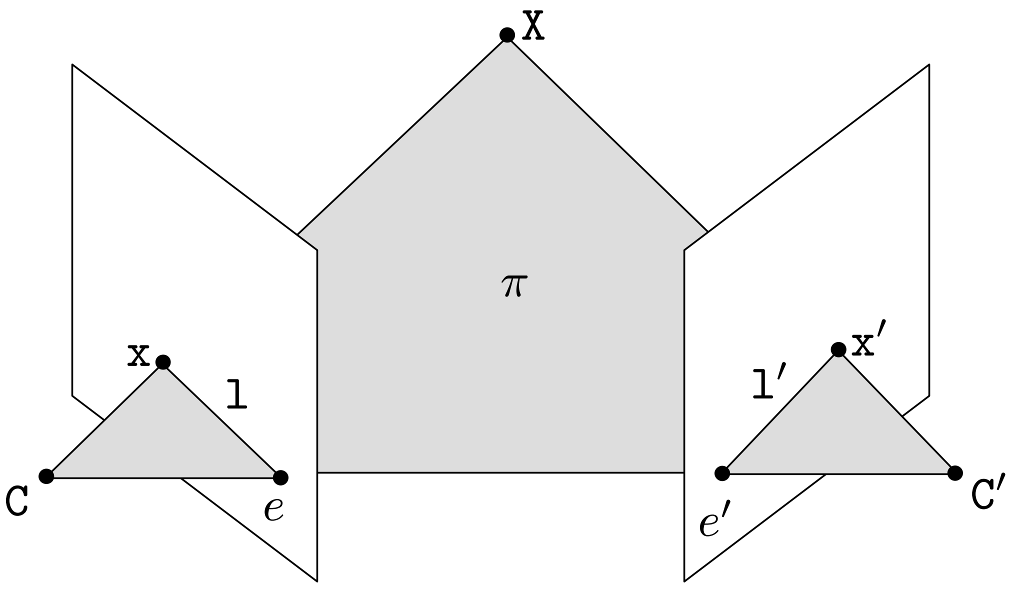


Figure A.5: Epipolar Geometry. The two cameras are indicated by their camera centers $\mathtt{C}$ and $\mathtt{C}'$ and image planes. The camera centers $\mathtt{C}$ and $\mathtt{C}'$, world point $\mathtt{X}$ and its image projections $\mathtt{x}$ and $\mathtt{x}'$ lie in a common plane π called *epipolar plane.* The intersection of the epipolar plane with the image planes defines the *epipolar lines* $\mathtt{l}$ and $\mathtt{l}'$. The line connecting the camera centers is called *baseline* and its intersections with the image planes are the *epipoles* $\mathtt{e}$ and $\mathtt{e}'$.

plane with the two image planes result in the epipolar lines $\mathtt{l}$ and $\mathtt{l}'$. The epipolar line $\mathtt{l}$ joins the image point $\mathtt{x}$ with the epipole $\mathtt{e}$ while $\mathtt{l}'$ joins $\mathtt{x}'$ with $\mathtt{e}'$.

The *fundamental matrix* is the algebraic representation of epipolar geometry. Given a pair of images, there exists to each point $\mathtt{x}$ in one image a corresponding epipolar line $\mathtt{l}'$ in the other image. Any point $\mathtt{x}'$ in the second image corresponding to the point $\mathtt{x}$ in the first image must lie on the epipolar line $\mathtt{l}'$. The epipolar line $\mathtt{l}'$ is the projection of the ray from the point $\mathtt{x}$ through the camera center $\mathtt{C}$ of the first camera in the second image. The mapping

$$\mathtt{x} \mapsto \mathtt{l}'$$

is represented by the fundamental matrix $\mathbf{F}$

$$\mathtt{l}' = \mathbf{F}\mathtt{x}.$$

The fundamental matrix is a homogeneous 3×3 matrix of rank 2 that maps points to lines. It satisfies the condition that for any pair of corresponding points $\mathtt{x} \leftrightarrow \mathtt{x}'$ in the two images

$$\mathtt{x}'\mathbf{F}\mathtt{x} = 0.$$

This relation characterizes the fundamental matrix with respect to image correspondences only, without referencing the camera matrices. This is an important observation since it enables $\mathbf{F}$ to be computed from image correspondences alone. There are many ways to compute the fundamental matrix from point correspondences that differ in parameterization and optimality criterion. A review of many algorithms, together with a comparison of their performance, can be found in [Zhang 1998]. A popular and simple to implement algorithm that provides fairly good results is the normalized 8-point algorithm [Hartley 1997]. Longuet-Higgins was the first to propose a linear 8-point algorithm for the computation of the essential matrix in [Longuet-Higgins 1981]. However, it is susceptible to noise and therefore was considered useless in practice until Hartley pointed out that the sensitivity to noise was mainly due to badly conditioned equation systems that arise from improper scaling of the input data. He proposed a modification to the algorithm that included a normalization of the data.

A.5 Feature Extraction

In most matching or tracking applications, especially for camera calibration or 3D reconstruction, local features (or keypoints) are especially important since they provide a limited set of well localized and individually identifiable anchor points. Despite their recent success, the foundations of modern feature detection and matching techniques were being laid in the 1980s, even though different terminology was used and the level of invariance was less than today. Already in 1981, Lucas and Kanade [Lucas & Kanade 1981] developed a patch tracker based on two-dimensional image statistics, while Moravec [Moravec 1983] introduced the concept of "corner-like" feature points. Förstner [Förstner 1986] and then Harris [Harris & Stephens 1988] both proposed finding keypoints using measures based on eigenvalues of smoothed outer products of gradients, which are still widely used today.

These early techniques relied on matching patches around the detected keypoints, which limited their range of applicability to scenes seen from similar viewpoints such as aerial photogrammetry applications. If features are being tracked from frame to frame, an affine extension of the basic Lucas-Kanade tracker has been shown to perform well [Shi & Tomasi 1994]. However, for true wide baseline matching (i.e., the automatic matching of images taken from widely different views) affine-invariant feature descriptors have to be used. In the following some work in the field of local invariant features is briefly reviewed. A more comprehensive survey on local invariant features can be found in [Tuytelaars & Mikolajczyk 2008] while Mikolajczyk et al. [Mikolajczyk & Schmid 2005] review some recently developed view-invariant local image descriptors and experimentally compare their performance.

The class of *distribution-based descriptors* uses histograms to represent different characteristics of appearance or shape. Probably the most simple descriptor is the distribution of the pixel intensities represented by a histogram. A more expressive representation was introduced by Johnson et al. [Johnson & Hebert 1997] for 3D object recognition in the context of range data. Their representation, called spin-image, is a histogram of the point positions in the neighborhood of a 3D interest point. Recently, it was adapted to images in [Lazebnik *et al.* 2003]. The two dimensions of the histogram are the distance from the center and the intensity value. Zabih and Woodfill [Zabih & Woodfill 1994] developed an approach that is robust to illumination changes. It relies on histograms of ordering and reciprocal relations between pixel intensities which are more robust than raw pixel intensities. This descriptor is suitable for texture representation, but requires a large number of dimensions to obtain a robust descriptor [Ojala *et al.* 2002]. Lowe [Lowe 2004] proposed a scale invariant feature transform (SIFT), which combines a scale invariant region detector and a descriptor based on the gradient distribution in the detected regions. The descriptor is represented by a 3D histogram of gradient locations and orientations. The contribution to the location and orientation bins is weighted by the gradient magnitude. The quantization of gradient locations and orientations makes the descriptor robust to small geometric distortions and small errors in the region detection. Geometric histograms [Ashbrook *et al.* 1995] and shape contexts [Belongie *et al.* 2002] implement the same idea and are very similar to the SIFT descriptor. Both methods compute a histogram describing the edge distribution in a region.

Another class of techniques are *spatial-frequency techniques* that describe the frequency content of an image. The Fourier transform is one alternative to decompose the image content into basis functions. However, spatial relations between points are not explicit and basis functions are infinite. Therefore it is difficult to adapt to a local approach. The Gabor transform [Gabor 1946] overcomes these problems, but a large number of Gabor filters are required to capture small changes in frequency and orientation. Wavelets and Gabor filters are frequently investigated in the context of texture classification.

The class of *differential descriptors* represents the neighborhood of a point using local derivatives. The properties of local derivatives (local jet) were investigated by Koenderink and van Doorn [Koenderink & van Doom 1987]. Later, Florack et al. [Florack *et al.* 1991] derived differential invariants, which combine components of the local jet to obtain rotation invariance. Freeman and Adelson [Freeman & Adelson 1991] developed steerable filters, which steer derivatives in a particular direction given the components of the local jet. Steering derivatives in the direction of the gradient makes them invariant to rotation. A stable estimation of the derivatives is obtained by convolution with Gaussian derivatives. Instead of using Gaussian filters, Baumberg [Baumberg 2000] and Schaffalitzky et al. [Schaffalitzky & Zisserman 2002] proposed using complex filters that differ from the Gaussian derivatives by a linear coordinate change in the filter response domain.

A.6 Feature Matching

In the previous section methods to extract feature points and corresponding descriptors from images were presented. In order to be able to use the features to compute the fundamental matrix, for example, it is necessary to determine corresponding or matching features (i.e., image projections of the same world point) in the different images. Let $\mathcal{F}_I$ and $\mathcal{F}_J$ denote the set of features found in images I and J, respectively. Then, the problem of matching the features of I and J is to find a (possibly empty) set of correspondences

$$\mathcal{M}_{IJ} = \{(f_i, g_j) \mid f_I \in \mathcal{F}_I \text{ and } g_J \in \mathcal{F}_J \text{ corresponding}\}\,.$$

In order to match the features, they have to be compared with respect to some similarity criterion. Typically, the distance between two features f_i and

g_j is measured using the Euclidean distance

$$\|d(f_i) - d(g_j)\|^2$$

between their descriptor vectors $d(f_i)$ and $d(g_j)$. An alternative way to compare two features is two measure the cosine of the angle between their descriptors

$$\cos(d(f_i), d(g_j)) = \frac{\langle d(f_i), d(g_j) \rangle}{\|d(f_i)\| \, \|d(g_j)\|}.$$

Given some distance function $dist(\cdot)$ to compare two features, several strategies for matching a set of features $\mathcal{F}_I$ to a set $\mathcal{F}_J$ exist. In *threshold-based matching*, a feature $f_i \in \mathcal{F}_I$ is assumed to be in correspondence to all features in $\mathcal{F}_J$ whose distance between their descriptors is below a threshold

$$\mathcal{M}_t = \{(f_i, g_j) \in \mathcal{F}_I \times \mathcal{F}_J \mid dist(f_i, g_j) < \epsilon\}.$$

With this approach each feature $f_i \in \mathcal{F}_I$ can have several or none corresponding features in $\mathcal{F}_J$. In contrast to that, *nearest neighbor-based matching* assumes a feature f_i to be in correspondence with the feature with the closest descriptor

$$\mathcal{M}_{nn} = \left\{(f_i, g_j) \in \mathcal{F}_I \times \mathcal{F}_J \mid g_j = \underset{g_k \in \mathcal{F}_J}{\arg\min}\, d(f_i, g_k)\right\}.$$

Using nearest neighbor-based matching each feature in $\mathcal{F}_I$ has exactly one match. The *closest-to-next-closest matching* [Lowe 2004] is similar to nearest neighbor matching, except that an additional thresholding is applied to the distance ratio between the closest and the second closest neighbor. Thus, the features f_i and g_j are matched if

$$\frac{dist(f_i, g_j)}{dist(f_i, g_k)} < \epsilon, \tag{A.4}$$

and g_j is the nearest and g_k is the second-nearest neighbor to f_i. Thus, each feature has exactly one match or none. This measure performs well because correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching. For false matches, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space.

The matching strategies presented above are not symmetric. If features of $\mathcal{F}_I$ are matched to features of $\mathcal{F}_J$ the resulting set of correspondences is in general different from the one obtained by matching $\mathcal{F}_J$ to $\mathcal{F}_I$. However, symmetry can be enforced by performing the matching twice, exchanging I and

J, and keeping only matches found in both runs. In addition to that, uniqueness (i.e., a one-to-one mapping) can be efficiently enforced after matching (demanding uniqueness during matching has high computational complexity) by removing features for that more then one corresponding feature was found.

The simplest way to find all corresponding features in an image pair is to compare all the features in one image to all the features in the other. Unfortunately, this is quadratic in the number of features, which makes it impractical for some applications. More efficient matching algorithms can be devised using different kinds of indexing schemes, many of which are based on the idea of finding nearest neighbors in high-dimensional spaces. A vast number of data structures for fast nearest neighbor searching have been proposed in literature. One of them are *kd-trees* that are based on a hierarchical decomposition of space in multidimensional rectangles. Unfortunately, kd-trees have proven to be inefficient for dimensions larger than 10. Given that the most powerful descriptors have many more dimensions, one would not consider kd-trees as a possible solution. However, an approach to overcome the dimensionality problem is to search for approximate nearest neighbors [Arya *et al.* 1998][Liu *et al.* 2004]. Vision applications have found this solution to be sufficient in practice [Lowe 2004][Mori *et al.* 2001], as other parts of the recognition systems are highly inaccurate too.

A.7 Outlier Removal

Although feature detection and matching algorithms have made substantial progress in recent years, the resulting feature correspondences almost always contain a significant amount of false matches. Since these *outliers* can severely disturb subsequent estimations, for example of a homography, camera or fundamental matrix, they have to be identified and removed. Methods for outlier removal are typically applied before any estimation takes place. From the remaining set of correspondences the sought quantity can then be computed using robust estimation techniques as presented in the next section.

The *disparity gradient* is a measure of the geometric compatibility of two feature pairs. Given two feature pairs (f, f') and (g, g') their disparity gradient can be defined as

$$dpg(f, f', g, g') = \frac{|dp(f, f') - dp(g, p')|}{|d_{cs}(f, f', g, g')|}, \tag{A.5}$$

where the disparity

$$dp(f, f') = p(f) - p(f')$$

between two features f and f' is the difference vector between their image locations $p(\cdot)$. The cyclopean separation

$$d_{cs}(f, f', g, g') = \frac{p(f) + p(f')}{2} - \frac{p(g) + p(g')}{2}$$

is defined as the vector joining the midpoints of the line segments connecting the features (see Figure A.6). The idea behind the disparity gradient is that if f and g are close together in both images, they should have a similar parallax (a small numerator in Equation A.5). Consequently, the smaller the disparity gradient, the more the two correspondences are in agreement. The performance of outlier removal based on the disparity gradient can be further improved if the process is performed locally instead of on the whole image. To this end, the image is typically divided in patches (usually six or eight, according to the image size) and for each patch the sum of disparity gradients for each correspondence inside the patch is computed. Those matches that have a disparity gradient sum greater than the median of the sums are rejected. Alternatively, the disparity gradient can be used in an iterative process, where incompatible matches are successively removed until all pairs have a similar disparity gradient. This simple test on the local geometric consistency of the matches typically removes about 80% of the false correspondences at low computational time.

An alternative to the disparity gradient is to consider the *relative position of correspondences*. The disparity of a correct match should be similar to the disparity of neighboring matches. Typically, the angle between the disparities and the ratio of their magnitudes are used for comparison and have to be smaller than a given threshold in order to be accepted. For a given correspondence this test is usually performed with the n closest neighbors (typically three to five), and the match is considered valid if the majority

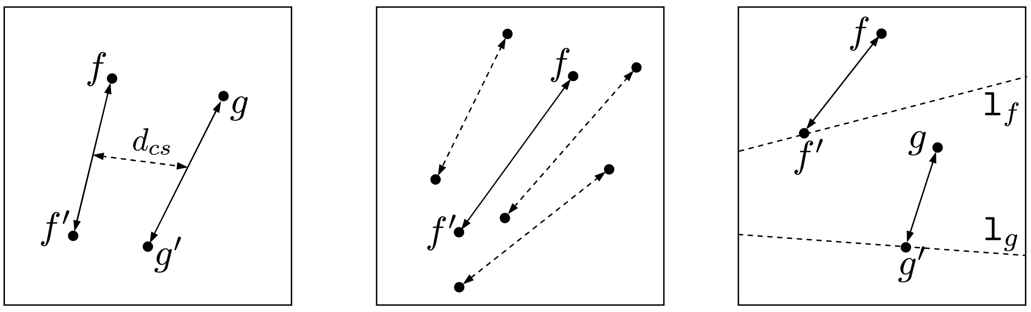


Figure A.6: Outlier Removal. Disparity gradient (left), relative positions of neighbors (middle) and epipolar constraint (right).

of the tests vote for accepting it. As long as the image pair's baseline is relatively small, these simple constraints give similar results as the disparity gradient. However, it is more difficult to select good thresholds on the disparity angle and magnitude.

Another popular method to remove outliers is to utilize the *epipolar constraint* imposed by the epipolar geometry and the corresponding fundamental matrix. The fundamental matrix can be computed robustly from feature correspondences (e.g., using the normalized 8-point algorithm) in a RANSAC procedure (see next section). Given the fundamental matrix $\mathbf{F}$ describing the epipolar geometry between image I and J, the feature pair (f, f') is compatible with the epipolar geometry if

$$\|\mathbf{F}p(f) - p(f')\| < \epsilon.$$

A.8 Robust Estimation

The goal of robust estimation is to determine a set of *inliers* from given input correspondences so that the sought quantity can be computed from them in an optimal manner using the standard estimation algorithms. A popular and very successful robust estimator is *RANSAC (Random Sample Consensus)* [Fischler & Bolles 1981]. The idea is to randomly select a sample of data points and to instantiate the model (compute an estimation of the sought quantity) from it. The support for this model is measured by the number of data points that lie within a distance threshold. This random selection is repeated a number of times and the model with the most support is deemed the robust fit. All data points that support this model constitute the set of inliers.

In most cases, it is computationally infeasible and unnecessary to try every possible sample. Instead the number of samples N is chosen sufficiently high to ensure with a probability p (often set to 0.99), that at least one of the random samples of s points (where s has to be large enough to instantiate the model) does not contain any outliers. Let w denote the probability that a selected data point is an inlier and thus $\epsilon = 1 - w$ the probability that it is an outlier. Then at least

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \tag{A.6}$$

selections of s points are required. In practice, the fraction of data consisting of outliers is often unknown. In such cases ϵ can be initialized with a worst case estimate that is updated as larger consistent sets are found. A new

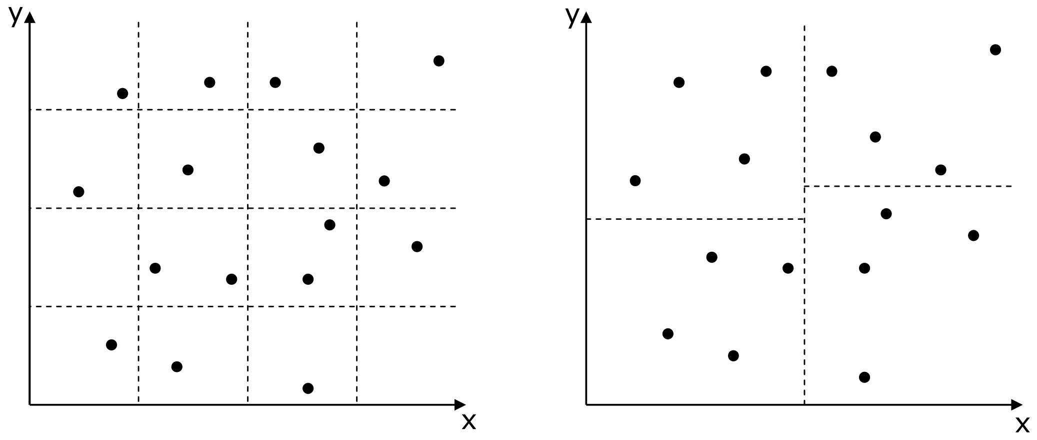


Figure A.7: Bucketing Strategies. Example of two different bucketing strategies. Regular buckets (right) and adaptive buckets (left).

(lower) estimate of ϵ also implies a reduced N according to Equation A.6. The algorithm terminates as soon as N samples have been performed. This adaptive approach works very well in practice and also covers the questions of both number of samples and terminating the algorithm.

Bucketing techniques use clusters of measures (buckets) that cover the whole data space to drive the sampling stage of RANSAC. The idea behind bucketing techniques is to ensure that the estimated model is valid on the whole working space. For example, a camera calibration has to be valid for the entire image. It would be useless to obtain a perfect camera model from a set of local correspondences which would be valid only on a small part of the image. There are several ways of building subsets of the data points (see Figure A.7). Points can be grouped into equally sized buckets as proposed in [Zhang *et al.* 1995]. Alternatively, they can also be spread across buckets that split the space arbitrarily along one or several dimensions. There are also different strategies of selecting sample points from the buckets. For example, a point may be randomly chosen from each bucket. Although this densely samples the whole working space (if the buckets are small enough) there might exist buckets that contain only outliers. Another approach is to first randomly select a subset of (usually s) buckets from which then again the points are randomly chosen.

A.9 Bundle Adjustment

Bundle adjustment is the process of refining a given scene reconstruction in order to obtain jointly optimal estimates for structure and viewing parameters. The name refers to the bundles of light rays leaving each 3D feature and converging on each camera center (see Figure A.8). Suppose a set of 3D points $\mathtt{X}_j$ is viewed by a set of cameras $\mathbf{P}^i$. Let $\mathtt{x}_j^i$ denote the coordinates of the j-th point as seen by the i-th camera. The reconstruction problem can then be formulated as follows: Given the set of image coordinates $\mathtt{x}_j^i$, find the set of world points $\mathtt{X}_j$ and the set of camera matrices $\mathbf{P}^i$ such that $\mathbf{P}^i\mathtt{X}_j = \mathtt{x}_j^i$. Without further restrictions on the $\mathbf{P}_i$ or $\mathtt{X}_j$ such a reconstruction is a projective reconstruction. If the image measurements are noisy, then the equations $\mathbf{P}^i\mathtt{X}_j = \mathtt{x}_j^i$ will not be satisfied exactly. Typically, Gaussian noise is assumed in the measurements and the Maximum Likelihood solution is sought. To this end, projection matrices $\hat{\mathbf{P}}^i$ and 3D points $\hat{\mathtt{X}}_j$ are estimated that project exactly to image points $\hat{\mathtt{x}}_j^i$. Thus,

$$\hat{\mathtt{x}}_j^i = \hat{\mathbf{P}}^i\hat{\mathtt{X}}_j, \ \forall i, j$$

while at the same time the image distance between the reprojected points and measured image points for every view in which the 3D point appears is minimized

$$\sum_{i,j} \left\| \hat{\mathbf{P}}^i\hat{\mathtt{X}}_j - \mathtt{x}_j^i \right\|^2 \to \min_{\hat{\mathbf{P}}^i, \hat{\mathtt{x}}_j}!.$$

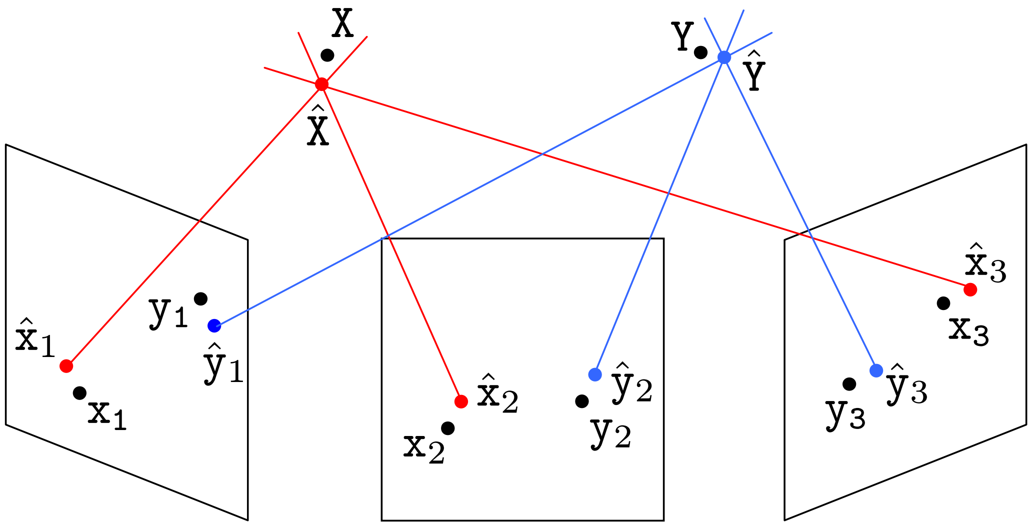


Figure A.8: Bundle Adjustment.

Since each camera matrix has 11 degrees of freedom and each 3-space point three degrees of freedom, a reconstruction involving n points over m views requires a minimization over $3n+11m$ parameters. If the Levenberg-Marquardt algorithm is used for minimization, matrices of dimension $(3n+11m)\times(3n+11m)$ must be factored. As m and n increase this becomes extremely costly. Fortunately, the matrices arising in bundle adjustment have a sparse block structure due to the lack of interaction among parameters for different 3D points and cameras. Therefore, considerable computational benefit can be gained by using sparse variants of the Levenberg-Marquardt algorithm that explicitly take advantage of the matrix structure [Lourakis & Argyros 2004].

APPENDIX B

Appendix: Compositing

B.1 Mesh Parameterization

A parameterization of a surface can be viewed as a bijective mapping from a suitable parameter domain to a surface. In general, the parameter domain itself will be a surface and so constructing a parameterization means mapping one surface into another. In the case that these surfaces are represented by triangle meshes, the problem of computing such a mapping is referred to as *mesh parameterization*. The map is piecewise linear, associating each triangle of the original mesh with a triangle in the parameter domain. Parameterizations between surface meshes and a variety of domains have numerous applications in computer graphics and geometry processing. In recent years numerous methods for parameterizing meshes were developed, targeting diverse parameter domains and focusing on different parameterization properties [Floater & Hormann 2005][Sheffer *et al.* 2006].

For computer graphics applications, such as texture mapping, planar parameterization of meshes with disk-like topology are of particular interest. Since planar parameterization is only applicable to surfaces with disk-like topology, closed surfaces and surfaces with genus greater than zero have to be cut prior to planar parameterization. Planar parameterization of 3D surfaces inevitably creates distortion in all but special cases. Greater surface complexity usually increases parameterization distortion, independent of the parameterization technique used. To allow parameterizations with low distortion, the surfaces must be cut to reduce the complexity. Since cuts introduce discontinuities into the parameterization, a delicate balance between the conflicting goals of small distortion and short cuts has to be achieved.

B.1.1 Measuring Distortion

Given an orientable 2-manifold surface patch $S \subset \mathbb{R}^k$, a parameterization is defined as a homeomorphism

$$\begin{aligned} \phi : \Omega \subset \mathbb{R}^2 &\rightarrow S \subset \mathbb{R}^k \\ (u,v) &\mapsto \phi(u,v) \end{aligned}$$

from the parameter space Ω into S. If ϕ is a differentiable parameterization, the first fundamental form $\mathbf{I}_\phi$, which captures the metric structure of S, is defined as

$$\mathbf{I}_\phi = \nabla^\mathsf{T}\phi \cdot \nabla\phi = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

$$\text{with } a = \left\|\frac{\partial\phi}{\partial u}\right\|^2,\ b = \left\langle \frac{\partial\phi}{\partial u}, \frac{\partial\phi}{\partial v} \right\rangle \text{ and } c = \left\|\frac{\partial\phi}{\partial v}\right\|^2.$$

Since $\mathbf{I}_\phi$ is a symmetric positive definite 2×2 matrix in every $\omega \in \Omega$, it induces a scalar product on $\mathbb{R}^2$. It describes the lengths and angles of vectors in $\mathbb{R}^2$ after being mapped by $\mathbf{I}_\phi$.

A parameterization is called *conformal* (or angle-preserving) if for every $\omega \in \Omega$

$$\mathbf{I}_\phi = \lambda(\omega) \cdot \mathbf{I}.$$

Consequently, derivatives of the iso-u and iso-v curves passing through $\phi(\omega)$ are orthogonal and of the same magnitude. Thus, conformal mappings preserve angles. If the maximal and minimal eigenvalue of $\mathbf{I}_\phi$ are denoted by λ_{min} and λ_{max}, respectively, conformality can also be expressed as

$$\frac{\lambda_{max}}{\lambda_{min}} = 1.$$

Since $0 < \lambda_{min} \leq \lambda_{max}$, minimizing this ratio optimizes angular distortion.

Conformality allows the directional derivatives to be uniformly scaled by a factor $\lambda(\omega)$ that may vary from point to point on the surface. If this factor does not equal one, a shape in the domain appears stretched or shrinked when mapped onto the surface and its area is distorted. If in addition to angles, area is to be preserved globally, the magnitude of the directional derivatives has to be fixed resulting in an isometry. A parameterization is said to be *isometric* (or length-preserving) if

$$\lambda(\omega) = 1,\ \forall\omega \in \Omega.$$

In other words the first fundamental form equals the identity matrix in every point. An isometric parameterization preserves angles and area globally. Unfortunately, isometric parameterizations exist only for developable surfaces (i.e., surfaces with zero Gaussian curvature), such as a cylinder. In the general case of non zero Gaussian curvature, angle and area preservation have to be traded off. To measure area deformation imposed by a map ϕ, a sufficiently small axis aligned square in Ω of area A can be considered. The image of this square is a trapezoid spanned between the directional derivatives in u and v whose area is given by $A \cdot \sqrt{\det(\mathbf{I}_\phi)}$. Thus, ϕ preserves area if and only if

$$\det(\mathbf{I}_\phi) = 1.$$

B.1.2 The Used Parameterization

In this thesis the parameterization method by Degener et al. [Degener *et al.* 2003] is applied. It uses an energy functional that quantifies angle and global area deformations simultaneously while the relative importance between angle and area preservation can be controlled by the user through a parameter. In the presented thesis this parameter is chosen to obtain a parameterization that is optimized for a uniform sampling of the terrain surface.

To enforce the area and angle preservation $f(x) = x + \frac{1}{x}$ is used as objective function. Angle deformation is measured by the ratio of eigenvalues $\frac{\lambda_{max}}{\lambda_{min}}$ of the first fundamental form. This yields

$$E_{angle}(\omega) = f(\sqrt{\frac{\lambda_{max}}{\lambda_{min}}}) = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}} + \sqrt{\frac{\lambda_{min}}{\lambda_{max}}}.$$

Area deformation is defined as

$$E_{area}(\omega) = f(\sqrt{\det(\mathbf{I}_\phi)}) = \sqrt{\det(\mathbf{I}_\phi)} + \frac{1}{\sqrt{\det(\mathbf{I}_\phi)}}.$$

The deformation energies are then combined as

$$E(\omega) = E_{angle}(\omega) \cdot (E_{area}(\omega))^\theta$$

where the parameter θ varies between 0 and ∞ and controls the relative importance of area and angle preservation. For the special choice of $\theta = 1$, the combined energy becomes the simple product

$$E_{angle} \cdot E_{area} = f(\lambda_{max}) + f(\lambda_{min}).$$

As the eigenvalues λ_{max} and λ_{min} measure the greatest and the smallest stretch that the parameterization ϕ imposes on a vector of unit length, the energy obtained for $\theta = 1$ enforces an uniform sampling of the surface, and penalizes oversampling ($\lambda_{min} < 1$) as well as undersampling ($\lambda_{max} > 1$).

A parameterization ϕ can now be assigned a combined area and angle distortion by integrating over the surface patch S

$$E(\phi) = \int_S E(\phi^{-1})dp$$

To minimize the non-linear isometric energy the hierarchical parameterization algorithm proposed by Hormann et al. [Hormann *et al.* 1999] is used.

APPENDIX C

Appendix: Digital Landform Mapping

C.1 Edge Detection

Edge detection is one of the most common operations applied in image analysis, in particular in image segmentation. An *edge* is the boundary between an object and the background, and indicates the boundary between overlapping objects. This means that if the edges in an image can be identified accurately, all of the objects can be located and basic properties such as area, perimeter, and shape can be measured.

Physical edges correspond to discontinuities in the physical, geometrical and photometrical properties of scene objects. The principal physical edges correspond to significant variations in reflectance, illumination, position and orientation of scene surfaces. *Image edges*, however, are characterized as discontinuities in image intensity, color or texture. Since image intensity is often proportional to scene radiance, image edges often correspond to physical edges. Therefore, the most common approach to local boundary detection is to look for discontinuities in image brightness using image derivates. Typically, edges are localized as positive maxima or negative minima of the first-order derivative or as zero-crossings of the second-order derivative.

C.1.1 Convolution

Convolution is a mathematical concept that is the basis for sampling, reconstruction and filtering. A convolution takes two functions f and g as input and combines them to produce a new function $f * g$. The functions can be continuous or discrete, and may be defined on a one-dimensional, two-dimensional or higher-dimensional domain. Since many of the important applications of sampling and reconstruction in graphics are applied to 2D functions, in particular to images, only convolution in 2D is considered in the following.

The convolution of two *discrete* sequences in 2D is defined as

$$(a * b)[i, j] = \sum_{i'} \sum_{j'} a[i', j']b[i - i', j - j'],$$

whereas the *continuous* convolution is defined as

$$(f * g)(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x', y')g(x - x', y - y')dx'dy'.$$

These definitions can be easily extended to higher dimensions in the same way. Convolution is commutative and associative, and it is distributive over addition

$$\begin{aligned} f * g &= g * f \\ f * (g * h) &= (f * g) * h \\ f * (g + h) &= f * g + f * h. \end{aligned}$$

Taking the derivative of a convolution is commutative and associative

$$\frac{\partial}{\partial x}(f * g) = \frac{\partial}{\partial x} f * g = f * \frac{\partial}{\partial x} g.$$

A function $f(x, y)$ is said to be *separable* if there are two functions f_1 and f_2 of one variable such that

$$f(x, y) = f_1(x) f_2(y).$$

For the Gaussian this is a consequence of the fact that

$$e^{x+y} = e^x e^y.$$

The key advantage of separable filters over non-separable 2D filters has to do with efficiency in implementation. A convolution with a 2D separable filter can be expressed by two convolutions with 1D filters. If the filter has radius r this reduces computational complexity from $O(r^2)$ to $O(r)$.

C.1.2 Image Derivatives

Given a function $f(x, y)$ describing the image, the vector

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

is called the *image gradient.* The *gradient magnitude*

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2}$$

describes the amount of the difference between pixels in the neighborhood (the strength of the edge). The *gradient orientation*

$$\phi\left(\nabla f(x,y)\right) = \arctan \left(\frac{\frac{\partial f}{\partial x}}{\frac{\partial f}{\partial y}} \right)$$

gives the direction of the greatest change, which presumably is the direction across the edge (the edge normal). The *Laplacian* operator

$$\nabla \cdot \nabla \left(f(x,y) \right) = \nabla^2 \left(f(x,y) \right) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

measures second derivatives. Edges can be found by looking for zero-crossings in the Laplacian of the image. Since edges represent high-frequency image content, edge detectors are in general susceptible to noise. The difficulty is to distinguish noise from image edges that correspond to relevant edges. While smoothing, for example with a Gaussian kernel, reduces noise and therefore improves robustness of edge detection, it also causes information loss and degrades the localization of edges. The ultimate goal is therefore to find detectors that ensure a good compromise between noise reduction and edge conservation. Blurring and differentiating can be combined to a single convolution as

$$\begin{aligned} \nabla \left(G * f \right) &= \left(\nabla G \right) * f \\ \nabla^2 \left(G * f \right) &= \left(\nabla^2 G \right) * f. \end{aligned}$$

References

Adalsteinsson, D. and Sethian, J. A. (1995). A fast level set method for propagating interfaces. *J. Comput. Phys.*, **118**(2), 269–277. 116

Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D. and Cohen, M. (2004). Interactive digital photomontage. *ACM Trans. Graph.*, **23**(3), 294–302. 75

Agathos, A. and Fisher, R. (2003). Colour texture fusion of multiple range images. *Pages 139–146 of: 3DIM03.* 76

Agrawal, A., Radhakrishna, M. and Joshi, R. (2006). Geometry-based Mapping and Rendering of Vector Data over LOD Phototextured Terrain Models. *In: Proceedings of WSCG.* 28

Akbarzadeh, A., Frahm, J.-M., Mordohai, P., Clipp, B., Engels, C., Gallup, D., Merrell, P., Phelps, M., Sinha, S., Talton, B., Wang, L., Yang, Q., Stewenius, H., Yang, R., Welch, G., Towles, H., Nister, D. and Pollefeys, M. (2006). Towards Urban 3D Reconstruction from Video. *Pages 1–8 of: 3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06).* Washington, DC, USA: IEEE Computer Society. 53

Alliez, P., Ucelli, G., Gotsman, C. and Attene, M. (2005). *Recent Advances in Remeshing of Surfaces.* Research Report. AIM@SHAPE Network of Excellence. 15

Amini, A. A., Weymouth, T. E. and Jain, R. C. (1990). Using Dynamic Programming for Solving Variational Problems in Vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, **12**(9), 855–867. 95

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. and Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, **45**(6), 891–923. 56, 147

Ashbrook, A. P., Thacker, N. A., Rockett, P. I. and Brown, C. I. (1995). Robust recognition of scaled shapes using pairwise geometric histograms. *Pages 503–512 of: BMVC '95: Proceedings of the 6th British conference on Machine vision (Vol. 2)*. Surrey, UK, UK: BMVA Press. 145

Ballard, D. H. and Sklansky, J. (1973). Tumor Detection in Radiographs. *Computers and Biomedical Research*, **6**(4), 299–321. 95

Ballard, D. H. and Brown, C. M. (1982). *Computer Vision*. Prentice Hall Professional Technical Reference. 95

Bannai, N., Agathos, A. and Fisher, R. B. (2004). Fusing Multiple Color Images for Texturing Models. *Pages 558–565 of: 3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium on (3DPVT'04)*. Washington, DC, USA: IEEE Computer Society. 76

Baumberg, A. (2000). Reliable Feature Matching across Widely Separated Views. *Pages 774–781 of: Proceedings of Conference on Computer Vision and Pattern Recognition*. 145

Baumberg, A. (2002). Blending Images for Texturing 3D Models. *Page 3D and Video of: BMVC02*. 75

Beauchesne, E. and Roy, S. (2003). Automatic relighting of overlapping textures of a 3D model. *Pages II: 166–173 of: Conference on Computer Vision and Pattern Recognition (CVPR) 2003*. 76

Beers, A. C., Agrawala, M. and Chaddha, N. (1996). Rendering from compressed textures. *Pages 373–378 of: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM. 12

Belongie, S., Malik, J. and Puzicha, J. (2002). Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **2**(4), 509–522. 145

Bernardini, F., Martin, I. M. and Rushmeier, H. (2001). High-Quality Texture Reconstruction from Multiple Scans. *IEEE Transactions on Visualization and Computer Graphics*, **7**(4), 318–332. 75

Blake, A., Rother, C., Brown, M., Perez, P. and Torr, P. (2004). Interactive Image Segmentation Using an Adaptive GMMRF Model. *Pages Vol I: 428–441 of: European Conference in Computer Vision (ECCV) 2004*. 97

Borodin, P., Guthe, M. and Klein, R. (2003). Out-of-Core Simplification with Guaranteed Error Tolerance. *Pages 309–316 of:* Ertl, T., Girod, B., Greiner, G., Niemann, H., Seidel, H.-P., Steinbach, E. and Westermann, R. (eds), *Vision, Modeling and Visualisation 2003*. Akademische Verlagsgesellschaft Aka GmbH, Berlin. 17

Boykov, Y. and Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. *Proceedings of ICCV*, **1**, 105–112. 95, 97, 109, 114, 115

Boykov, Y. and Kolmogorov, V. (2004). An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, **26**(9), 1124–1137. Member-Yuri Boykov and Member-Vladimir Kolmogorov. 112

Boykov, Y., Veksler, O. and Zabih, R. (1998). Markov Random Fields with Efficient Approximations. *Page 648 of: CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society. 111

Brown, M. and Lowe, D. G. (2005). Unsupervised 3D Object Recognition and Reconstruction in Unordered Datasets. *Pages 56–63 of: 3DIM '05: Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*. Washington, DC, USA: IEEE Computer Society. 52

Bruneton, E. and Neyret, F. (2008). Real-time rendering and editing of vector-based terrains. *In: Eurographics*. 29

Burt, P. J. and Adelson, E. H. (1983a). A Multiresolution Spline with Application to Image Mosaics. *ACM Transactions on Graphics*, **2**(4), 217–236. 75, 83

Burt, P. J. and Adelson, E. H. (1983b). The Laplacian Pyramid as a compact image code. *IEEE Transactions on Communications*, **COM-31,4**, 532–540. 75

Cappelletti, J. D. and Rosenfeld, A. (1989). Three-dimensional boundary following. *Comput. Vision Graph. Image Process.*, **48**(1), 80–92. 95

Chen, C.-Y. and Klette, R. (1999). Image Stitching - Comparisons and New Techniques. *Pages 615–622 of: CAIP '99: Proceedings of the 8th International Conference on Computer Analysis of Images and Patterns*. London, UK: Springer-Verlag. 73

Chen, S. E. (1995). QuickTime VR: an image-based approach to virtual environment navigation. *Pages 29–38 of: SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM. 72

Chien, Y. and Fu, K. (1974). A Decision Function Method for Boundary Detection. *CGIP*, **3**(2), 125–140. 95

Chuang, Y.-Y., Curless, B., Salesin, D. H. and Szeliski, R. (2001). A Bayesian Approach to Digital Matting. *Pages 264–271 of: Proceedings of IEEE CVPR 2001*, vol. 2. IEEE Computer Society. 113

Cignoni, P., Montani, C. and Scopigno, R. (1998). A comparison of mesh simplification algorithms. *Computers & Graphics*, **22**(1), 37–54. 16

Cohen, J., Olano, M. and Manocha, D. (1998). Appearance-Preserving Simplification. *Pages 115–122 of: SIGGRAPH 98*. 16

Cohen, L. D. (1991). On active contour models and balloons. *CVGIP: Image Underst.*, **53**(2), 211–218. 95

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R. and Schrijver, A. (1998). *Combinatorial optimization*. New York, NY, USA: John Wiley & Sons, Inc. 112

Cooke, R. and Doornkamp, J. (1974). Geomorphology in environmental management. *ITC Journal*, 352–397. 91

Crow, F. (1977). Shadow Algorithms for Computer Graphics. *Pages 242–248 of: Proceedings of SIGGRAPH*. 37

Daneels, D. (1993). Interactive Outlining: An Improved Approach Using Active Contours. *Pages 226–233 of: in SPIE Proceedings of Storage and Retrieval for Image and Video Databases*. 95

Debevec, P. E., Taylor, C. J. and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics*, **30**(Annual Conference Series), 11–20. 53

Degener, P., Meseth, J. and Klein, R. (2003). An Adaptable Surface Parametrization Method. *The 12th International Meshing Roundtable*. 78, 155

Dempster, A., Laird, N. and Rubin, D. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *J. Roy. Statist. Soc.*, **39**, 1–38. 113

Dick, A. R., Torr, P. H. S. and Cipolla, R. (2004). Modelling and Interpretation of Architecture from Several Images. *Int. J. Comput. Vision*, **60**(2), 111–134. 53

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, **1**, 269–270. 101

DIKAU, R. (1989). The application of a digital relief model to landform analysis in geomorphology. *Three dimensional applications in Geographical Information Systems*, 51–77. 92

DING, M., LYNGBAEK, K. AND ZAKHOR, A. (2008). Automatic registration of aerial imagery with untextured 3D LiDAR models. *In: CVPR*. 54

DOUGLAS, D. AND PEUCKER, T. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, **10**(2), 112–122. 124

DUDA, R. O., HART, P. E. AND STORK, D. G. (2000). *Pattern Classification*. Wiley-Interscience Publication. 97, 113

ERIKSON, C., MANOCHA, D. AND WILLIAM V. BAXTER, I. (2001). HLODs for faster display of large static and dynamic environments. *Pages 111–120 of: I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*. New York, NY, USA: ACM. 17

ESRI. (1998). *ESRI Shapefile Technical Description - An ESRI White Paper*. 25

EVERITT, C. AND KILGARD, M. (2002). Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. *Published on-line at developer.nvidia.com*. 37

FALCAO, A. X., UDUPA., J. K., SAMARASEKERA, S., SHARMA, S., E., B. AND LOTUFO, R. (1998). User-steered image segmentation paradigms: live wire and live lane. *Graphical Models and Image Processing*, **60**, 233–260. 95, 96

FALCAO, A. X., UDUPA., J. K. AND MIYAZAWA, F. K. (2000). An ultra-fast user-steered image segmentation paradigm: live wire on the fly. *IEEE Transactions on Medical Imaging*, **19**(1), 55–62. 96

FISCHLER, M. A. AND BOLLES, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, **24**(6), 381–395. 56, 149

FITZGIBBON, A. W. AND ZISSERMAN, A. (1998). Automatic Camera Recovery for Closed or Open Image Sequences. *Pages 311–326 of: ECCV '98: Proceedings of the 5th European Conference on Computer Vision-Volume I*. London, UK: Springer-Verlag. 53

FLOATER, M. S. AND HORMANN, K. (2005). Surface parameterization: a tutorial and survey. *Advances in Multiresolution for Geometric Modelling*, 157–186. 153

FLORACK, L., TER HAAR ROMENY, B., KOENDERINK, J. AND VIERGEVER, M. (1991). General intensity transformations and second order invariants. *In: Proceedings of the 7th Scandinavian Conference on Image Analysis*. 145

Floriani, L., L.Kobbelt and Puppo, E. (2004). A Survey on Data Structures for Level-Of-Detail Models. *Advances in Multiresolution for Geometric Modelling, Series in Mathematics and Visualization*, 49–74. 16

Ford, L. and Fulkerson, D. (1962). *Flows in Networks*. Princeton University Press. 112

Freeman, W. T. and Adelson, E. H. (1991). The Design and Use of Steerable Filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, **13**(9), 891–906. 145

Förstner, W. (1986). A feature-based correspondence algorithm for image matching. *International Archives Photogrammetry and Remote Sensing*, **26**(3), 150–166. 144

Gabor, D. (1946). Theory of Communication. *JIEE*, **93**, 429–459. 145

Garland, M. (1999). *Multiresolution Modeling: Survey & Future Opportunities*. Eurographics State of The Art Report (STAR). 16

Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. *Pages 209–216 of: SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. 15

Garland, M. and Heckbert, P. S. (1998). Simplifying surfaces with color and texture using quadric error metrics. *Pages 263–269 of: VIS '98: Proceedings of the conference on Visualization '98*. Los Alamitos, CA, USA: IEEE Computer Society Press. 16

Geiger, D., Gupta, A., Costa, L. A. and Vlontzos, J. (1995). Dynamic Programming for Detecting, Tracking, and Matching Deformable Contours. *IEEE Trans. Pattern Anal. Mach. Intell.*, **17**(3), 294–302. 95

Goesele, M., Snavely, N., Curless, B., Hoppe, H. and Seitz, S. M. (2007). Multi-View Stereo for Community Photo Collections. *In: ICCV 2007*. 70, 131

Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *J. ACM*, **35**(4), 921–940. 112

Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.*, **6**(11), 21–29. 72

Greig, D., Porteous, B. and Seheult, A. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, **51**(2), 271–279. 111

Gumhold, S. and Strasser, W. (1998). Real time compression of triangle mesh connectivity. *Pages 133–140 of: SIGGRAPH 98 Conference Proceedings.* 11

Gustavsson, M., Kolstrup, E. and Seijmonsbergen, A. (2006). A new symbol-and-GIS based detailed geomorphological mapping system: Renewal of a scientific discipline for understanding landscape development. *Geomorphology*, **77**, 90–111. 91

Guthe, M., Borodin, P. and Klein, R. (2005a). Fast and accurate Hausdorff distance calculation between meshes. *Journal of WSCG*, **13**(2), 41–48. 15

Guthe, M., Balázs, A. and Klein, R. (2005b). GPU-based trimming and tessellation of NURBS and T-Spline surfaces. *ACM Transactions on Graphics*, **24**(3), 1016–1023. 31

Harris, C. and Stephens, M. J. (1988). A combined corner and edge detector. *In: A combined corner and edge detector.* 144

Hartley, R. I. (1997). In Defense of the Eight-Point Algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, **19**(6), 580–593. 56, 143

Hauber, E., Slupetzky, H., Jaumann, R., Wewel, F., Gwiner, K. and Neukum, G. (2000). Digital and automated high resolution stereo mapping of the Sonnblick glacier (Austria) with HRSC-A. *In: Proceedings of EARSeL-SIG-Workshop Land Ice and Snow.* 10

Heckbert, P. and Garland, M. (1997). *Survey of polygonal surface simplification algorithms.* SIGGRAPH 97 Course Notes 25. 16

Heidmann, T. (1991). Real Shadows Real Time. *IRIS Universe*, **18**, 28–31. 37

Herley, C. (2005). Automatic Occlusion Removal from Minimum Number of Images. *Pages II: 1046–1049 of: IEEE International Conference on Image Processing (ICIP) 2005.* 90

Hoppe, H. (1996). Progressive meshes. *Pages 99–108 of: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM. 17

Hoppe, H. (1998). Smooth view-dependent level-of-detail control and its application to terrain rendering. *Pages 35–42 of: VIS '98: Proceedings of the conference on Visualization '98.* Los Alamitos, CA, USA: IEEE Computer Society Press. 17

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W. (1993). Mesh optimization. *Pages 19–26 of: SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM. 14

Hormann, K., Greiner, G. and Campagna, S. (1999). Hierarchical Parametrization of Triangulated Surfaces. *Pages 219–226 of:* Girod, B., Niemann, H. and Seidel, H.-P. (eds), *Proceedings of Vision, Modeling, and Visualization 1999*. Erlangen, Germany: infix. 156

Hsu, S. C., Samarasekera, S., Kumar, R. and Sawhney, H. S. (2000). Pose Estimation, Model Refinement, and Enhanced Visualization Using Video. *Pages 1488–1495 of: CVPR*. IEEE Computer Society. 54

Hüttner, T. (1998). High Resolution Textures. *In: Visualization '98 - Late Breaking Hot Topics Papers*. 19

Hu, J., You, S. and Neumann, U. (2006). Automatic Pose Recovery for High-Quality Textures Generation. *Pages 561–565 of: ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*. Washington, DC, USA: IEEE Computer Society. 54

Inc., S. (1998). *S3TC DirectX 6.0 Standard Texture Compression*. 12

Isenburg, M. and Snoeyink, J. (2000). Face fixer: compressing polygon meshes with properties. *Pages 263–270 of: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. 12

Johnson, A. and Hebert, M. (1997). Object Recognition by Matching Oriented Points. *In: CVPR*. 144

Juan, O. and Boykov, Y. (2007). Capacity Scaling for Graph Cuts in Vision. *Pages 1–8 of: IEEE 11th International Conference on Computer Vision (ICCV) 2007*. 98

Kang, H. W. and Shin, S. Y. (2002). Enhanced lane: interactive image segmentation by incremental path map construction. *Graphical Models*, **64**(5), 292–303. 96, 104

Kang, H. W. (2005). G-wire: a livewire segmentation algorithm based on a generalized graph formulation. *Pattern Recogn. Lett.*, **26**(13), 2042–2051. 97

Karypis, G. and Kumar, V. (1998). Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, **48**, 96–129. 116

Kass, M., Witkin, A. and Terzopoulos, D. (1988). Snakes: Active Contour Models. *International Journal of Computer Vision*, **1**(4), 321–331. 95

Kersting, O. and Döllner, J. (2002). Interactive 3D visualization of vector data in GIS. *Pages 107–112 of: GIS '02: Proceedings of the 10th ACM international symposium on Advances in geographic information systems*. New York, NY, USA: ACM. 29

Klein, R. and Liebich, G. (1996). Mesh reduction with error control. *Pages 311–318 of: Visualization 96. ACM.* 15, 22

Klein, R. and Schilling, A. G. (2001). Efficient Multiresolution Models for Progressive Terrain Rendering. *Chap. 7, pages 109–130 of:* Schilling, A. G. (ed), *Festschrift zum 60. Geburtstag von Wolfgang Straßer*. Wilhelm-Schickard-Institut f. Informatik. 19

Koenderink, J. J. and van Doom, A. J. (1987). Representation of local geometry in the visual system. *Biol. Cybern.*, **55**(6), 367–375. 145

Kugler, H. (1964). Die geomorphologische Reliefanalyse als Grundlage großmaßstäbiger geomorphologischer Kartierung. *Wissenschaftliche Veröffentlichungen des Deutschen Instituts für Länderkunde*, **21/22**, 541–655. 91

Lazebnik, S., Schmid, C. and Ponce, J. (2003). *Sparse texture representation using affine-invariant neighborhoods.* 144

Lee, S. C., Jung, S. K. and Nevatia, R. (2001). Automatic Integration of Facade Textures into 3D Building Models with a Projective Geometry Based Line Clustering. *Computer Graphics Forum*, **21**(3), 511–519. 54

Lensch, H. P., Heidrich, W. and Seidel, H.-P. (2001). A silhouette-based algorithm for texture registration and stitching. *Graph. Models*, **63**(4), 245–262. 74

Levin, A., Zomet, A., Peleg, S. and Weiss, Y. (2004). Seamless Image Stitching in the Gradient Domain. *Pages Vol IV: 377–389 of: 8th European Conference on Computer Vision (ECCV) 2004.* 76

Li, Y., Sun, J., Tang, C.-K. and Shum, H.-Y. (2004). Lazy snapping. *ACM Transaction on Graphics*, **23**(3). 97, 112

Liu, L., Stamos, I., Yu, G., Wolberg, G. and Zokai, S. (2006). Multiview Geometry for Texture Mapping 2D Images Onto 3D Range Data. *Pages 2293–2300 of: CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Washington, DC, USA: IEEE Computer Society. 54

Liu, T., Moore, A., Gray, A. and Yang, K. (2004). An investigation of practical approximate nearest neighbor algorithms. *In: NIPS.* 147

Lloyd, B. and Egbert, P. (2002). Horizon occlusion culling for real-time rendering of hierarchical terrains. *Pages 403–410 of: VIS '02: Proceedings of the conference on Visualization '02.* Washington, DC, USA: IEEE Computer Society. 23

LOHMAR, F. J. (1988). *World geodetic system 1984 - geodetic reference system of GPS orbits*. Springer Berlin / Heidelberg. Pages 476–486. 7

LOMBAERT, H., SUN, Y., GRADY, L. AND XU, C. (2005). A multilevel banded graph cuts method for fast image segmentation. *Proceedings of ICCV*, 259–265. 98, 116

LONGUET-HIGGINS, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, **293**, 133–135. 52, 143

LOURAKIS, M. AND ARGYROS, A. (2004). *The design and implementation of a generic sparse bundle adjustment software package based on the Levenberg–Marquardt algorithm*. 59, 152

LOWE, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Distinctive image features from scale-invariant keypoints*, **62**(2), 91–110. 51, 55, 56, 144, 146, 147

LUCAS, B. D. AND KANADE, T. (1981). An iterative image registration technique with an application in stereo vision. *Pages 674–679 of: International joint conference on artificial intelligence*. 144

LUEBKE, D., WATSON, B., COHEN, J. D., REDDY, M. AND VARSHNEY, A. (2002). *Level of Detail for 3D Graphics*. New York, NY, USA: Elsevier Science Inc. 16

LUEBKE, D. P. (2001). A Developer's Survey of Polygonal Simplification Algorithms. *IEEE Comput. Graph. Appl.*, **21**(3), 24–35. 16

MANJUNATH, B. S. AND MA, W. Y. (1996). Texture Features for Browsing and Retrieval of Image Data. *Pages 837–842 of: IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18. Washington, DC, USA: IEEE Computer Society. 82

MARTINEC, D. AND PAJDLA, T. (2006). 3D reconstruction by gluing pair-wise euclidean reconstructions, or "How to achieve good reconstruction from bad image". *In: 3DPVT*. 53

MARTINEC, D. AND PAJDLA, T. (2007). Robust rotation and translation estimation in multiview reconstruction. *In: CVPR*. 53

MASLENNIKOVA, A. AND VEZHNEVETS, V. (2007). Interactive Local Color Transfer Between Images. *In: GraphiCon*. 82

MCGUIRE, M., HUGUES, J. F., EGAN, K. T., KILGARD, M. AND EVERITT, C. (2003). Fast, Practical and Robust Shadows. *Technical Report CS03-19*. 37

Miano, J. (1999). *Compressed image file formats: JPEG, PNG, GIF, XBM, BMP*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. 12

Mikolajczyk, K. and Schmid, C. (2005). A Performance Evaluation of Local Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, **27**(10), 1615–1630. 55, 144

Moravec, H. (1983). The Stanford cart and the CMU rover. *Proceedings of the IEEE*, **71**(7), 872–884. 144

Mori, G., Belongie, S. and Malik, H. (2001). Shape contexts enable efficient retrieval of similar shapes. *In: CVPR*. 147

Mortensen, E. N. and Barrett, W. A. (1995). Intelligent scissors for image composition. *SIGGRAPH 95 Proceedings*, 191–198. 95, 98

Mortensen, E. N. and Barrett, W. A. (1998). Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*, **60**, 349–384. 95

Mortensen, E. N. and Barrett, W. A. (1999). Toboggan-based intelligent scissors with a four parameter edge model. *Proceedings of IEEE Computer Vision and Pattern Recognition*, **2**, 452–458. 96

Neukum, G. (2001). The airborne HRSC-AX cameras: evaluation of the technical concept and presentation of application results after one year of operations. *Photogrammetric Week*, **1**, 117–130. 9

Neumann, U., You, S., Hu, J., Jiang, B. and Lee, J. (2003). Augmented Virtual Environments (AVE): Dynamic Fusion of Imagery and 3D Models. *Page 61 of: VR '03: Proceedings of the IEEE Virtual Reality 2003*. Washington, DC, USA: IEEE Computer Society. 54

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga. 101

Nistér, D. (2000). Reconstruction from Uncalibrated Sequences with a Hierarchy of Trifocal Tensors. *Pages 649–663 of: ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part I*. London, UK: Springer-Verlag. 53

Ojala, T., Pietikäinen, M. and Mäenpää, T. (2002). Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, **24**(7), 971–987. 144

Oliensis, J. (1999). A Multi-Frame Structure-from-Motion Algorithm under Perspective Projection. *Int. J. Comput. Vision*, **34**(2-3), 163–192. 52

Otto, J.-C. (2006). *Paraglacial sediment storage quantification in the Turtmann Valley, Swiss Alps*. Ph.D. thesis, University of Bonn. 121

Otto, J.-C. and Dikau, R. (2004). Geomorphologic System Analysis of a high mountain valley in the Swiss Alps. *Zeitschrift für Geomorphologie*, **48**(3), 323–341. 91, 121

Otto, J.-C., Kleinod, K., König, O., Krautblatter, M., Nyenhuis, M., Roer, I., Schneider, M., Schreiner, B. and Dikau, R. (2007). HRSC-A data: a new high-resolution data set with multipurpose applications in physical geography. *Progress in Physical Geography*, **31**(Otto2006), 179–197. 2, 7

Pajarola, R. (2002). *Overview of Quadtree-based Terrain Triangulation and Visualization*. Tech. rept. UCI-ICS Technical Report No. 02-01. University of California, Irvine. 21

Pajarola, R. and Gobbetti, E. (2007). Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, **23**(8), 583–605. 21

Pérez, P., Gangnet, M. and Blake, A. (2003). Poisson image editing. *ACM Trans. Graph.*, **22**(3), 313–318. 75

Permuter, H. and Francos, J. (2003). Gaussian Mixture Models of Texture and Colour for Image Database Retrieval. *Pages 25–88 of: in Proc . ICASSP*. 113

Polis, M. F., Gifford, S. J. and Jr., D. M. M. (1995). Automating the Construction of Large-Scale Virtual Worlds. *Computer*, **28**(7), 57–65. 29

Pollefeys, M., Koch, R. and Gool, L. V. (1999). Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. *International Journal of Computer Vision*, **32**(1), 7–25. 52

Pollefeys, M. and Gool, L. V. (2002). From images to 3D models. *Commun. ACM*, **45**(7), 50–55. 52

Puppo, E. (1998). Variable resolution triangulations. *Computational Geometry*, **11**, 219–238. 17

Reinhard, E., Ashikhmin, M., Gooch, B. and Shirley, P. (2001). Color transfer between images. *IEEE Computer Graphics and Applications*, **21**(5), 34–41. 76, 81

Rocchini, C., Cignoni, P., Montani, C. and Scopigno, R. (2004). Multiple Textures Stitching and Blending on 3D Objects. *Pages 119–130 of: Eurographics Rendering Workshop*. 74

Roman, A., Garg, G. and Levoy, M. (2004). Interactive Design of Multi-Perspective Images for Visualizing Urban Landscapes. *Pages 537–544 of: VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society. 53

Rossignac, J. (1999). Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, **5**, 47–61. 11

Rossignac, J. and Borrel, P. (1993). Multi-resolution 3D approximations for rendering complex scenes. *Pages 455–465 of: Geometric Modeling in Computer Graphics*. 14

Rothenbühler, C. (2003). Erfassung und Darstellung der Geomorphologie im Gebiet Bernina (GR) mit Hilfe von GIS. *Physische Geographie*, **41**, 117–126. 91

Rother, C., Kolmogorov, V. and Blake, A. (2004). GrabCut - Interactive foreground extraction using iterated graph cuts. *ACM Transaction On Graphics*, **23**(3), 309–314. 97, 114, 115

Ruderman, D. L., Cronin, T. W. and chin Chiao, C. (1998). Statistics of cone responses to natural images: implications for visual coding. *Journal of the Optical Society of America A*, **15**, 2036–2045. 76, 81

Ruzon, M. and Tomasi, C. (2000). Alpha Estimation in Natural Images. *Pages I: 18–25 of: CVPR*. 113

Sander, P. V., Gu, X., Gortler, S. J., Hoppe, H. and Snyder, J. (2000). Silhouette Clipping. *Pages 327–334 of:* Akeley, K. (ed), *Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman. 41

Schaffalitzky, F. and Zisserman, A. (2002). Multi-view Matching for Unordered Image Sets, or "How Do I Organize My Holiday Snaps?". *Pages 414–431 of: ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*. London, UK: Springer-Verlag. 53, 145

Schilling, A. and Klein, R. (1998). Rendering of multiresolution models with texture. *Computer and Graphics*, **22**(6), 667–674. 16

Schilling, A., Basanow, J. and Zipf, A. (2007). Vector based mapping of polygons on irregular terrain meshes for web 3d map services. *3rd International Conference on Web Information Systems and Technologies (WEBIST)*. 29

Schindler, G., Dellaert, F. and Kang, S. B. (2007). Inferring temporal order of images from 3D structure. *In: Proceedings of the IEEE conference on computer vision and pattern recognition.* 53

Schmidt, J. and Hewitt, A. (2004). Fuzzy land element classification from DTMs based on geometry and terrain position. *Geoderma*, **121**(3-4), 243–256. 92

Schneevoigt, N. and Schrott, L. (2006). Fernerkundungsbasierte Reliefformenerkennung im Hochgebirge (Reintal, Bayerische Alpen). *Geographica Helvetica*, **3**. 92

Schneider, M. and Klein, R. (2006). A Multilevel Banded Intelligent Scissors Method for Fast Segmentation in Large Virtual Terrains. *II International Conference Remote Sensing Archaeology.* 2

Schneider, M. and Klein, R. (2007a). Efficient and Accurate Rendering of Vector Data on Virtual Landscapes. *Journal of WSCG*, **15**(1-3). 2

Schneider, M. and Klein, R. (2007b). Semi-automatic Landform Mapping at Steep Slopes. *In: Proceedings of Joint Workshop "Visualization and Exploration of Geospatial Data".* 2

Schneider, M. and Klein, R. (2008). Enhancing Textured Digital Elevation Models Using Photographs. *In: Proceedings of 3DPVT.* 2

Schneider, M. and Otto, J.-C. (2006). A new semi-automatic tool for 3d landform mapping. *9th International Symposium on High Mountain Remote Sensing Cartography.* 2

Schneider, M., Guthe, M. and Klein, R. (2005). Real-time Rendering of Complex Vector Data on 3d Terrain Models. *Pages 573–582 of: Proceedings of The 11th International Conference on Virtual Systems and Multimedia.* 2

Scholten, F., Gwinner, K. and Wewel, F. (2002). Angewandte digitale Photogrammetrie mit der HRSC. *Photogrammetrie-Fernerkundung-Geoinformation*, **5**, 317–332. 10

Schroeder, W. J., Zarge, J. A. and Lorensen, W. E. (1992). Decimation of triangle meshes. *Pages 65–70 of: SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM. 14

Seijmonsbergen, A. and de Graaff, L. (2006). Geomorphological mapping and geophysical profiling for the evaluation of natural hazards in an alpine catchment. *Natural Hazards and Earth System Science*, **6**(2), 185–193. 91

SHEFFER, A., PRAUN, E. AND ROSE, K. (2006). Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis.*, **2**(2), 105–171. 153

SHI, J. AND TOMASI, C. (1994)(June). Good Features to Track. *In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*. 144

SHIRMAN, L. A. AND ABI-EZZI, S. S. (1993). The Cone of Normals Technique for Fast Processing of Curved Patches. *Computer Graphics Forum*, **12**(3), 261–272. 13

SINOP, A. K. AND GRADY, L. (2006). Accurate Banded Graph Cut Segmentation of Thin Structures Using Laplacian Pyramids. *Pages 896–903 of:* LARSEN, R., NIELSEN, M. AND SPORRING, J. (eds), *Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2006*. LNCS 4191, vol. IISpringer, for MICCAI Society. 98

SMITH, M. AND CLARK, C. (2005). Methods for the visualisation of digital elevation models for landform mapping. *Earth Surf. Processes Landforms*, **30**, 885–900. 92

SNAVELY, N., SEITZ, S. M. AND SZELISKI, R. (2008). Modeling the World from Internet Photo Collections. *International Journal of Computer Vision*. 52, 56

SNAVELY, N., SEITZ, S. M. AND SZELISKI, R. (2006). Photo Tourism: Exploring Photo Collections in 3D. *ACM Transactions on Graphics*. 51, 52

SONKA, M., WINNIFORD, M. D. AND COLLINS, S. M. (1995). Robust Simultaneous Detection of Coronary Borders in Complex Images. *IEEE Transactions on Medical Imaging*, **14**(1), 151–161. 95

SONKA, M., HLAVAC, V. AND BOYLE, R. (2007). *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering. 94, 95

SPETSAKIS, M. AND ALOIMONOS, J. Y. (1991). A multi-frame approach to visual motion perception. *Int. J. Comput. Vision*, **6**(3), 245–255. 52

STAMMINGER, M. AND DRETTAKIS, G. (2002). Perspective shadow maps. *ACM Trans. Graph.*, **21**(3), 557–562. 33

STAMOS, I. AND ALLEN, P. K. (2002). Geometry and texture recovery of scenes of large scale. *Comput. Vis. Image Underst.*, **88**(2), 94–118. 54

STÄBLEIN, G. (1980). Die Konzeption der Geomorphologischen Karten GMK 25 und GMK 100 im DFG-Schwerpunktprogramm. *Berliner Geographische Abhandlung*, **31**, 13–30. 91

Szeliski, R. and Kang, S. B. (1994). Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, **5**(1), 10–28. 52, 72

Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and environment maps. *Pages 251–258 of: SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. 72, 73

Tan, H. L., Gelfand, S. B. and Delp, E. J. (1992). A Cost Minimization Approach to Edge Detection Using Simulated Annealing. *IEEE Trans. Pattern Anal. Mach. Intell.*, **14**(1), 3–18. 95

Tanner, C. C., Migdal, C. J. and Jones, M. T. (1998). The clipmap: a virtual mipmap. *Pages 151–158 of: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM. 18

Teller, S., Antone, M., Bodnar, Z., Bosse, M., Coorg, S., Jethwa, M. and Master, N. (2003). Calibrated, Registered Images of an Extended Urban Area. *Int. J. Comput. Vision*, **53**(1), 93–107. 53

Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: a factorization method. *Int. J. Comput. Vision*, **9**(2), 137–154. 52

Touma, C. and Gotsman, C. (1998). Triangle mesh compression. *Pages 26–34 of: Proc. Graphics Interface*. 11, 12

Toussaint, G. (1983). *Solving geometric problems with the rotating calipers*. 79

Triggs, B., McLauchlan, P. F., Hartley, R. I. and Fitzgibbon, A. W. (2000). Bundle Adjustment - A Modern Synthesis. *Pages 298–372 of: ICCV '99: Proceedings of the International Workshop on Vision Algorithms*. London, UK: Springer-Verlag. 52

Turk, G. (1992). Re-tiling polygonal surfaces. *SIGGRAPH Comput. Graph.*, **26**(2), 55–64. 15

Tuytelaars, T. and Mikolajczyk, K. (2008). A Survey on Local Invariant Features. *Foundations and Trends in Computer Graphics and Vision*, **1**(1), 1–106. 144

Uyttendaele, M., Eden, A. and Szeliski, R. (2001). Eliminating Ghosting and Exposure Artifacts in Image Mosaics. *Pages II:509–516 of: Computer Vision and Pattern Recognition (CVPR) 2001*. 73, 76, 90 van Asselen, S. and Seijmonsbergen, A. (2006). Expert-driven semi-automated geomorphological mapping for a mountainous area using a laser DTM. *Geomorphology*, **78**(3-4), 309–320. 92

van der Zwet, P. N. J. and Reiber, J. H. C. (1992). A New Algorithm to Detect Irregular Coronary Boundaries: the Gradient Field Transform. *Pages 107–110 of: IEEE Proc. of Computers in Cardiology.* 95

Vergauwen, M. and Gool, L. V. (2006). Web-based 3D Reconstruction Service. *Mach. Vision Appl.*, **17**(6), 411–426. 53

Wahl, R., Massing, M., Degener, P., Guthe, M. and Klein, R. (2004). Scalable Compression of Textured Terrain Data. *Journal of WSCG*, **12**(3), 521–528. 21

Wartell, Z., Kang, E., Wasilewski, T., Ribarsky, W. and Faust, N. (2003). Rendering Vector Data over Global, Multiresolution 3D Terrain. *Pages 213–222 of: Proceedings on the Symposium on Data Visualization*, vol. 40. 28

Weber, A. and Benner, J. (2001). Interactive Generation of Digital Terrain Models Using Multiple Data Sources. *Pages 60–64 of: DEM '01: Proceedings of the First International Symposium on Digital Earth Moving.* London, UK: Springer-Verlag. 28

Williams, D. J. and Shah, M. (1992). A fast algorithm for active contours and curvature estimation. *CVGIP: Image Underst.*, **55**(1), 14–26. 95

Williams, L. (1978). Casting curved shadows on curved surfaces. *In SIGGRAPH 78*, 270–274. 33

Williams, L. (1983). Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, **17**(3), 1–11. 18

Wimmer, M., Scherzer, D. and Purgathofer, W. (2004). Light Space Perspective Shadow Maps. *Pages 143–151 of: Eurographics Symposium on Rendering.* 33

Wong, K. C., Heng, P. A. and Wong, T. T. (2000). Accelerating intelligent scissors using slimmed graphs. *Journal of Graphic Tools*, **5**(2), 1–13. 96

Yang, Q. H., Snyder, J. P. and Tobler, W. R. (2000). *Map Projection Transformation: Principles and Applications.* CRC Press. 7

Zabih, R. and Woodfill, J. (1994). Non-parametric Local Transforms for Computing Visual Correspondence. *Pages 151–158 of: ECCV (2).* 144

Zachmann, G. and Langetepe, E. (2003)(July). *Geometric Data Structures for Computer Graphics.* 18

Zhang, F., Sun, H., Xu, L. and Lun, L. K. (2006). Parallel-split shadow maps for large-scale virtual environments. *Pages 311–318 of: VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications.* New York, NY, USA: ACM Press. 45

Zhang, Z. (1998). Determining the Epipolar Geometry and its Uncertainty: A Review. *Int. J. Comput. Vision*, **27**(2), 161–195. 143

Zhang, Z., Deriche, R., Faugeras, O. and Luong, Q.-T. (1995). A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artif. Intell.*, **78**(1-2), 87–119. 150

Zhao, W., Nister, D. and Hsu, S. (2005). Alignment of Continuous Video onto 3D Point Clouds. *IEEE Trans. Pattern Anal. Mach. Intell.*, **27**(8), 1305–1318. Senior Member-Wenyi Zhao and Member-David Nister and Member-Steve Hsu. 54

Zucker, S. W. (1976). Region growing: Childhood and adolescence. *Comp. Graphics and Image Process.*, **5**, 382–399. 94