Interactive Visualization of Remote Sensing Data

Interaktive Visualisierung von Fernerkundungsdaten

Vom Department Elektrotechnik und Informatik
der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von
Martin Lambers

1. Gutachter:	Prof. Dr. Andreas Kolb
2. Gutachter:	Dr. Karol Myszkowski
Vorsitzender:	Prof. Dr. Roland Wismüller

Tag der mündlichen Prüfung: 1. Juli 2011

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Abstract

Remote Sensing is an important tool for the analysis and interpretation of a wide range of global and regional conditions and processes on the earth. Airborne and spaceborne Remote Sensing systems produce a rapidly growing number of data sets, and improvements in sensor technology result in continuously increasing spatial and spectral resolutions of these data sets.

To visualize Remote Sensing data for analysis and interpretation purposes, sensor data from various sources has to be processed and combined to produce geometry and color information. Since the details contained in multimodal high-resolution data sets cannot be preserved in a single static image, interactive visualization techniques are required.

A system for interactive visualization of Remote Sensing data allows to choose and adjust data processing and data fusion methods interactively. This enables the user to bring out the data details that are relevant for the current aims and objectives, and thus to gain better insight into the data.

Interactive visualization of Remote Sensing data is a challenging problem:

- Remote Sensing data sets are generally very large due to their high resolution and area coverage, and visualization tasks often need to combine multiple data sets. An interactive visualization system must be able to efficiently handle large amounts of diverse input data, and at the same time provide data processing capabilities that allow interactive adjustments.

- To produce geometry and color information from sensor data, specialized processing methods are required for different sensor systems. These specialized methods must allow interactive and intuitive adjustments.

- The dynamically generated geometry and color information resulting from interactive data processing and fusion must be rendered efficiently and accurately.

This dissertation proposes methods to address these challenges. A framework is presented that handles data management, processing, and fusion. Specialized interactive sensor data processing methods are examined based on the example of images generated by Synthetic Aperture Radar systems. Efficient geometry refinement methods are presented that are suitable for rendering dynamically generated data with guaranteed error bounds.

Zusammenfassung

Fernerkundung ist ein wichtiges Werkzeug für die Analyse und Interpretation einer Vielzahl von globalen und lokalen Zuständen und Prozessen auf der Erde. Flugzeug- und satellitengestützte Fernerkundungssysteme produzieren eine schnell wachsende Zahl an Datensätzen, und Fortschritte im Bereich der Sensortechnik führen zu immer höheren räumlichen und spektralen Auflösungen dieser Datensätze.

Für die Visualisierung von Fernerkundungsdaten müssen Sensordaten aus verschiedenen Quellen verarbeitet und kombiniert werden, um Geometrie- und Farbinformation zu produzieren. Da der Detailreichtum multimodaler, hochaufgelöster Datensätze nicht in einem einzigen statischen Bild wiedergegeben werden kann, sind interaktive Visualisierungstechniken erforderlich.

Ein System zur interaktiven Visualisierung von Fernerkundungsdaten erlaubt es, Methoden für die Datenverarbeitung und Datenfusion interaktiv auszuwählen und zu justieren. Dies ermöglicht es dem Benutzer, die für die aktuelle Fragestellung relevanten Details sichtbar zu machen, und so ein besseres Verständnis der Daten zu erlangen.

Interaktive Visualisierung von Fernerkundungsdaten ist eine Herausforderung:

- Fernerkundungsdatensätze sind aufgrund ihrer hohen Auflösung und Flächenabdeckung sehr groß, und Visualisierungsaufgaben erfordern häufig die Kombination mehrerer verschiedener Datensätze. Ein interaktives Visualisierungssystem muss daher große Mengen vielfältiger Eingabedaten effizient verwalten können, und gleichzeitig Datenverarbeitungskapazitäten bereit halten, die interaktives Eingreifen ermöglichen.

- Um Geometrie- und Farbinformation zu erzeugen, sind für verschiedene Sensorsysteme jeweils spezialisierte Datenverarbeitungsmethoden nötig. Diese Methoden müssen interaktives und intuitives Justieren erlauben.

- Die dynamisch generierte Geometrie- und Farbinformation aus den interaktiven Datenverarbeitungs- und Datenfusionsprozessen muss effizient und präzise zur Anzeige gebracht werden.

Diese Dissertation schlägt Methoden vor, diesen Herausforderungen zu begegnen. Es wird ein System vorgestellt, das Datenverwaltung, Datenverabeitung und Datenfusion übernimmt. Spezialisierte interaktive Verarbeitungsmethoden für Sensordaten werden am Beispiel von Bildern aus Radarsystemen mit synthetischer Apertur untersucht. Schließlich werden

effiziente Geometrieverfeinerungsmethoden vorgestellt, die zur Anzeige dynamischer Daten geeignet sind und dabei Fehlerobergrenzen garantieren können.

Contents

Introduction

Remote Sensing can be defined as the acquisition of information on some properties of a phenomenon or object by a recording device not in physical contact with the features under surveillance. This is a very broad definition; in common usage, the term usually refers to the acquisition of information about the earth's land surface, oceans, and atmosphere, by the use of sensing devices located at some distance from the observed target [Sho10, Cam08]. Most often, sensing devices sample electromagnetic radiation and are located on platforms such as aircrafts, satellites, ships, vehicles, or stationary facilities.

Remote Sensing is essential for the understanding of global interconnected processes, and thus takes a key role in observation, analysis, and planning tasks in important areas such as environment and ecosystems, food and agriculture, resources and trade.

Remote Sensing technologies are applied to a very broad range of problems. Examples include measuring ice shelf thickness, monitoring use of agricultural areas, locating sites of archaeological interest, estimating sea water temperature, or guiding emergency response teams, to name just a few. With this vast variety of applications comes a vast variety of specialized sensors. Examples include Light Detection And Ranging (LIDAR) sensors for distance measurements, radiometers for multispectral imaging, and weather surveillance radar systems.

A particularly versatile imaging sensor technology is Synthetic Aperture Radar (SAR). SAR systems record information about the electric field backscattered from an actively illuminated target area. This information is then transformed into a reflectivity map, the SAR image, using a special image formation process. SAR systems are independent of daylight and weather conditions, achieve high spatial resolutions, and provide rich information about the target scene. These properties make SAR systems indispensable for many Remote Sensing tasks.

With the growing importance of Remote Sensing, more and more data sets are produced, and at the same time advancements in sensor technology continue to increase their spatial and spectral resolutions. Making

these vast and fast growing amounts of sensor data accessible for analysis and interpretation requires visualization techniques.

Visualization of Remote Sensing Data

Remote Sensing data sets usually take the form of two-dimensional *maps* that cover a target area on the earth's land or ocean surface. This is true not only for data sets that provide information directly tied to the surface, such as optical image data sets or elevation models, but also for data sets whose samples can be meaningfully associated with locations on the surface, such as measurements about vegetation, information from ground penetrating radar systems, or atmospheric measurements for defined heights.

Many sensor data maps can be visualized by transforming them to two-dimensional images for display. Usually, different maps of a region form different *layers* in a visualization system, and the user can switch between layers or combine layers into a single view. Additional non-map layers can provide meta information such as area and place names, borders, or the location of roads or buildings.

A map that contains elevation data for surface locations is called a Digital Elevation Model (DEM). The elevation measures the distance between a surface point and the corresponding point on an underlying model of a planet, most often an ellipsoid. In the presence of one or more DEMs, a visualization system can render a three-dimensional view of the target scene. In this case, the elevation maps and accompanying image layers form *terrain* data, and terrain rendering methods are used to apply the image layers as textures to the geometry defined by the elevation maps.

Traditionally, the transformation of sensor data maps to textures and geometry takes place in an offline preprocessing step before the actual visualization. The visualization system then assumes that input geometry and textures are static. This allows significant optimizations and speedups in the rendering component. This approach is also used in the terrain rendering component of popular commercial applications such as Google Earth or NASA Worldwind. As a result, the only interaction available to the user during visualization is navigation through static data, and in some cases switching between different layers. The transformation process that created texture and geometry cannot be influenced anymore.

Problem Statement

The transformation of sensor data maps to textures and geometry generally depends on the transformation method and its parameters, and often is impossible without information loss. Additionally, there is no single "best" set of methods and parameters for a given data set. Instead, methods and parameters need to be chosen and adjusted to fit the needs of the current visualization task.

For these reasons, producing one static texture and geometry set for visualization is not enough. Instead, the user must be able to interactively influence the transformation process by choosing algorithms and setting parameters so that a deeper understanding of the data can be gained, and the features of interest for the task at hand can be identified and exposed more reliably.

This integration of the transformation process into the interactive visualization system poses three main challenges:

1. Large amounts of Remote Sensing data must be managed and processed fast enough for interactive use. The user must instantly get visual feedback on changes to the transformation process.

2. Specialized methods to transform sensor data to geometry and textures for visualization are required for different sensor technologies. These methods must be optimized for intuitive interactive use.

3. The dynamically generated terrain data (geometry and textures) has to be rendered accurately and fast.

In this dissertation, techniques and algorithms to address these challenges are proposed and combined into a framework for interactive visualization of Remote Sensing data.

To meet the large demand for computational power, todays programmable Graphics Processing Units (GPUs) are used not only for rendering, but also for general data processing tasks. Their highly parallel architecture is ideally suited for sensor data processing requirements. Additionally, since GPUs have become mass market articles, their price/performance ratio is very attractive.

The process of transforming sensor data to textures and geometry for visualization is highly specific to the sensor technology and its properties. This dissertation focusses on the transformation of SAR reflectivity maps, both because of the importance of SAR for Remote Sensing, and because SAR reflectivity maps are a prime example for the challenges involved in the visualization of Remote Sensing data.

Contribution

The contributions of this dissertation to the field of interactive visualization of Remote Sensing data are threefold:

1. A visualization framework, including

   - A combined data hierarchy and distributed data management using multiple cache levels.
   - A GPU-based processing pipeline for hierarchically organized sensor data.
   - Distributed and parallel rendering for Virtual Reality and Powerwall applications.
   - Visual assistance tools to help the user in interacting with the framework.

   Parts of this work were presented at the IEEE International Geoscience and Remote Sensing Symposium (IGARSS) [LKNK07, LK08b, LK09, LK10b].

2. Interactive visualization operators to transform SAR reflectivity maps into textures for display, in particular

   - Novel GPU-based global and local dynamic range reduction operators.
   - GPU-based despeckling operators.

   Large parts of this work were presented at IGARSS [LKNK07] and the European Conference on Synthetic Aperture Radar (EUSAR) [LK08a], and published in the IEEE Geoscience and Remote Sensing Letters [LNK08].

3. A novel technique for rendering dynamic terrain data with a guaranteed upper bound of the screen space error, using

   - A novel data structure to efficiently store and manage triangle mesh hierarchies.
   - Bottom up and top down refinement methods for triangle mesh hierarchies.

   This work was published in the 3D Research Journal [LK10a].

Chapter Overview

The general GPU-based visualization framework is described in Chapter 1. This framework provides the foundation for the interactive transformation of sensor data into geometry and textures, and for the subsequent rendering of this dynamic terrain data. Chapter 2 presents specialized visualization operators that transform SAR reflectivity maps to textures for rendering. These operators exemplify the challenges associated with the visualization of Remote Sensing data. Chapter 3 describes the data structures and refinement methods necessary to render the dynamically generated terrain data accurately. This dissertation concludes with a summary and outlook.

Chapter 1

Visualization Framework

1.1 Overview

A framework for interactive visualization of Remote Sensing data must be able to handle large amounts of input data, process and combine this data fast enough to allow interactive adjustments of processing and fusion methods, and render the resulting dynamically generated terrain data accurately. To achieve these goals, it is necessary to leverage the computational power of GPUs not only for rendering, but also for data processing and fusion.

An overview of such a framework is given in Fig. 1.1. The input consists of multiple hierarchically organized data sets containing georeferenced sensor data. Individual parts of these data sets are chosen depending on the current region of interest, and transferred to GPU memory. On the GPU, elevation input data is transformed to elevation maps, and image-like input data is transformed to color textures. These transformations are based on interactively adjustable processing methods tailored to the specific sensor technology. To combine the various input data sets into a single view, the resulting sets of elevation maps and textures are fused into a single set of elevation maps and a single set of textures using interactively adjustable data fusion methods. The result of this step is dynamically generated terrain data, consisting of an elevation map and texture hierarchy. This dynamic terrain data is finally rendered.

To implement such a framework, the following challenges need to be addressed:

- Data management. To be able to fuse multiple data sets for various areas of the earth's surface, a common data hierarchy must be defined that allows to manage georeferenced input data in a single, consistent representation. Since sensor data processing is performed

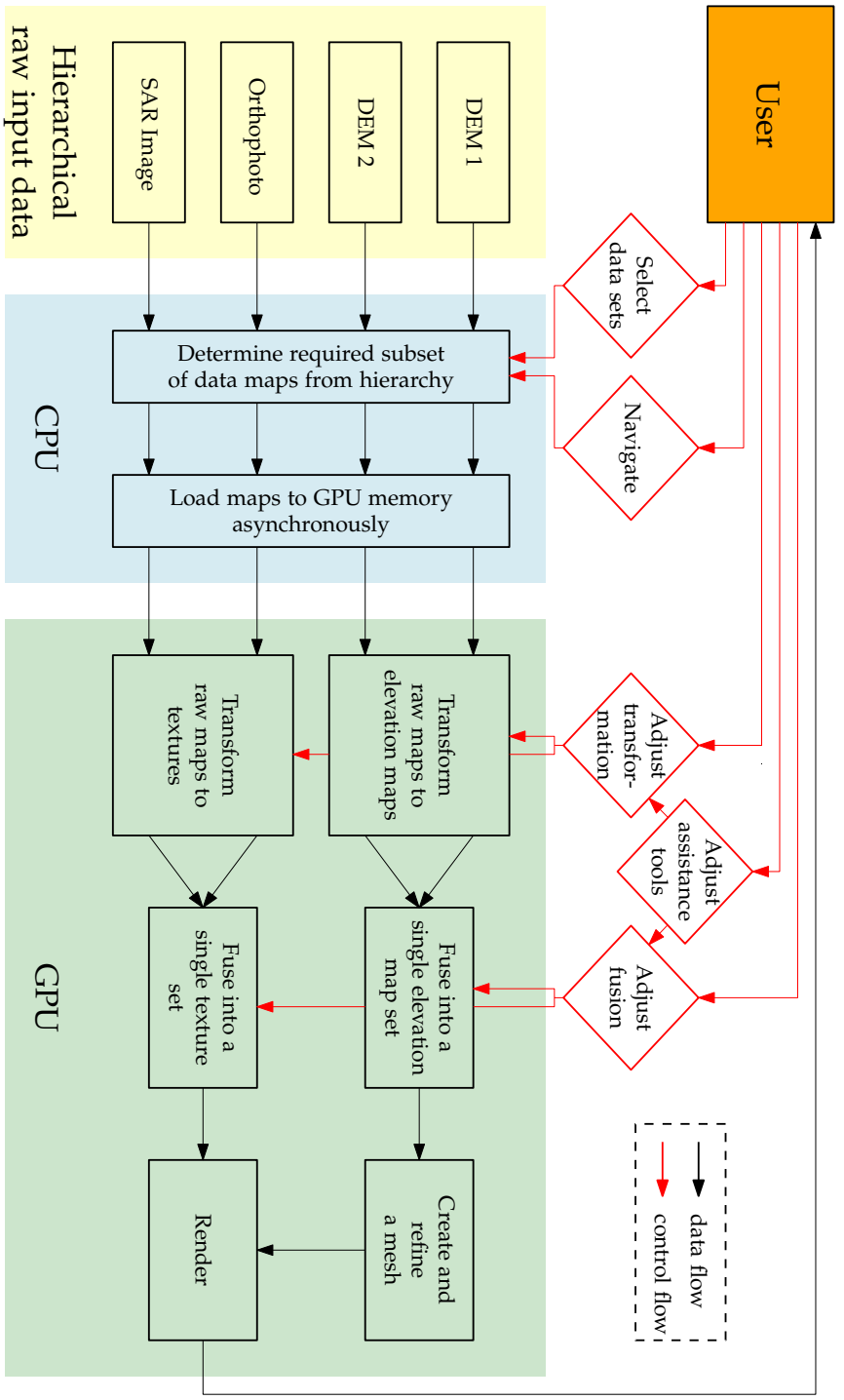

Figure 1.1: Overview of the visualization framework.

interactively and not in a preprocessing step, the data management component must be able to handle a wide range of sensor data types. Problems of storing and transferring very large amounts of data efficiently without blocking the interactive user interface need to be addressed.

- GPU-based data processing and fusion. The transformation of sensor data to elevation maps and textures, as well as subsequent fusion of multiple elevation map and texture sets, need to be performed on the GPU to achieve the processing speed that is necessary to allow interactive adjustments.

- Accurate rendering of dynamic terrain data. The dynamically generated elevation maps and textures need to be rendered efficiently. In the context of Remote Sensing data visualization, reliable and accurate rendering is important to avoid misinterpretation. Therefore, the rendering methods must guarantee upper bounds on the screen space error.

- Visual assistance tools. Since the interactive approach of visualization introduces many methods and parameters for the user to choose and adjust, additional assistance tools are required to help the user find the optimal set of parameters for the visualization task at hand.

This chapter describes framework components that address these challenges. These components form the basis for all methods and algorithms presented in this dissertation.

1.2 Data Hierarchy and Management

As described in the Introduction, this work focusses on Remote Sensing data that takes the form of two-dimensional maps for target areas on the earth's land or ocean surface.

Such maps can cover huge areas; some provide world-wide coverage. At the same time, advancements in sensor technology continue to increase the spatial and spectral resolutions of these maps. For example, for satellite-based systems, a ground resolution in the decimeter range is now common both for SAR and optical systems, and hyperspectral instruments are capable of resolving more than 200 spectral bands.

To handle the resulting huge amounts of sensor data in interactive visualization systems, hierarchical data structures must be used, and level of detail techniques must pick the required subset of data from these hierarchies to reduce the memory and processing requirements of the system.

This section describes the data hierarchy and management component of the framework, corresponding to the left side of Fig. 1.1. The underlying data structure is chosen to combine the following benefits of previously used structures:

- The data structure is suitable to store and manage huge amounts of diverse sensor data.
- The data structure provides a common reference system for georeferenced data covering different areas of the earth's surface.
- The data structure allows efficient adaptive grid triangulations for level of detail purposes.
- The data structure and resulting triangulations are regular to allow efficient parallel algorithm implementations on the GPU.

1.2.1 Related Work

Hierarchical data structures for two-dimensional maps are a fundamental component of many applications in computer graphics and computer vision, and thus have been thoroughly researched for many years.

In the context of terrain rendering, a hierarchical data structure must allow efficient construction of a single, seamless mesh for rendering. Additionally, the data hierarchy must allow efficient use of highly-parallel computing resources like the GPU. This latter requirement favors regular hierarchies over irregular hierarchies, because regular hierarchies tend to use simpler data structures that allow a higher grade of parallelization. This outweighs the greater adaptability of irregular structures: with modern GPUs, it is generally cheaper to render more triangles than strictly necessary than to compute an optimally adaptive triangulation.

A recent survey by Pajarola and Gobbetti gives an overview of data hierarchies that fulfill these requirements [PG07]. In this survey, two main classes of regular adaptive grid triangulation methods are identified: quadtree-based methods and triangle bin-tree based methods. Pajarola and Gobetti argue that both classes of methods lead to the same class of adaptive grid triangulations. Quadtree-based methods have the advantage to directly provide a hierarchical tiling suitable for storing elevation and image maps.

Originally, terrain rendering methods used such quadtree (or triangle bin-tree) structures to build the complete mesh for rendering on the CPU, including continuous level of detail computation [LKR*96, Paj98, DWS*97]. With GPU performance improvements, this approach became a bottleneck in recent years since GPUs nowadays can render far more

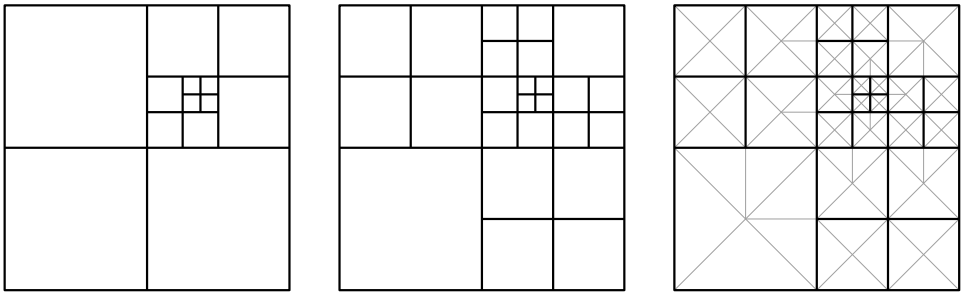

Figure 1.2: An unrestricted quadtree (left), a restricted quadtree (middle), and one possible conforming triangle mesh for the restricted quadtree (right).

triangles than these CPU-based methods can generate in a given time frame. For this reason, more recent methods tend to choose and combine larger blocks on the CPU for rendering on the GPU. Such blocks can for example consist of fixed or precomputed meshes for quadtree structures [SW06, DSW09]. Special care must be taken that the mesh is consistent at block boundaries. Some methods add special stitching geometry for this purpose [WMD*04, DSW09, LKES09], which requires special handling of block boundaries and introduces additional geometry data.

Most terrain rendering methods only handle terrain data sets that represent a limited local area on a planet's surface, and interpret height map input data relative to a local plane. Only a few methods explicitly address the problems associated with handling a complete planet surface [CGG*03, KLJ*09]. In this case, the geometry is given by interpreting elevation data relative to an ellipsoid or sphere that represents the planet. Sampling problems and distortion that arise when managing elevation and image map hierarchies for ellipsoid or sphere surfaces have to be addressed.

1.2.2 Restricted Quadtree

To combine the benefits of simple seamless triangle mesh creation with the performance advantages of block-based approaches, a *restricted quadtree* [PG07] is used as the basic hierarchical data structure, and patches of triangles are generated for each quad on the GPU.

A restricted quadtree is a quadtree in which the levels of neighboring quads differ by not more than one. See Fig. 1.2. Restricted quadtrees allow the generation of triangle meshes with two important properties:

1. The meshes consist entirely of right-angled, isosceles triangles. This regularity is important for GPU-based mesh refinement techniques,

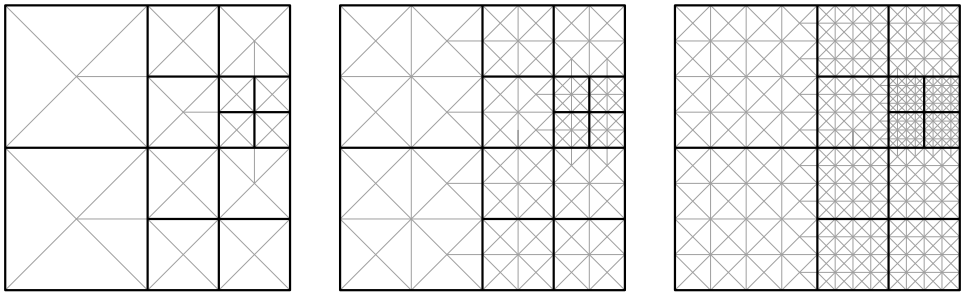

Figure 1.3: Different conforming triangle meshes for a restricted quadtree: level $l = 0$ (left) is the minimal conforming mesh. Level $l = 1$ (middle) and level $l = 2$ (right) correspond to the minimal conforming mesh after subdividing each quad of the quadtree l times.

as described in Chapter 3.

2. The meshes do not contain T-junctions that would lead to cracks in the geometry when the mesh is applied to elevation maps. See Fig. 1.2.

Restricted quadtree triangulations with these properties are called *conforming* triangulations.

There are multiple ways to create conforming triangle meshes from a restricted quadtree hierarchy [PG07]. The minimal conforming triangle mesh consists of 4 to 8 triangles per quad, depending on the levels of neighboring quads. This minimal conforming mesh is the *level* 0 triangulation of the restricted quadtree. Higher level triangulations (level $l, l > 0$) are equivalent to minimal triangulations applied after subdividing each quad of the restricted quadtree l times. See Fig. 1.3.

Triangle patches for a given level l can be generated for each quad easily and efficiently, and these patches form a conforming triangle mesh for the quadtree.

To meet the specific needs of Remote Sensing data visualization for the earth, the restricted quadtree variant used in the framework has the following additional properties:

- The quadtree is based on the World Geodetic System reference coordinate system, version WGS84. This coordinate system defines latitude/longitude coordinates relative to a reference ellipsoid that represents the earth [NGA]. The single quad in the lowest level zero of the quadtree represents the whole earth in the WGS84 coordinate system, with the latitude ranging from $+90^\circ$ (north) to -90° (south) and the longitude ranging from -180° (west) to $+180^\circ$ (east), as shown in Fig. 1.4.

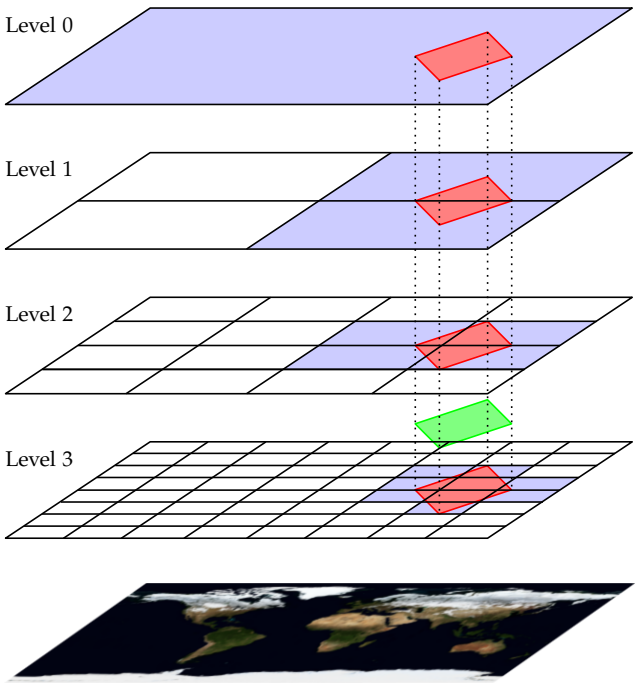

Figure 1.4: The quadtree variant used in the framework, representing the WGS84 earth map. A data set (shown in green) provides data for a subset of quads (shown in blue) in the hierarchy.

- The quadtree levels are contiguous at the $-180°/+180°$ longitude transition, i.e. the leftmost quads in a quadtree level are direct neighbors to the rightmost quads in this level. This guarantees a seamless visualization at this transition.
- To provide approximately square sample resolutions on the earth surface, each quad is twice as large in longitude than in latitude direction. With a quad size of 512×256 samples, quad level 16 would provide a ground coverage of about $1\,m^2$ per sample, and quad level 26 would provide a ground coverage of about $1\,mm^2$ per sample (near the equator). To generate conforming triangle meshes from quads that are twice as wide as high, each quad is subdivided once in longitude direction before creating a triangle patch for it. An example is given in Fig. 1.5.

Note that this choice of a common georeferencing coordinate system results in non-optimal behaviour at the north and south pole, because one side of the northernmost and southernmost quads will be reduced to a point when projecting WGS84 to cartesian coordinates. This leads to two problems. First, triangles generated for these quads will degenerate when

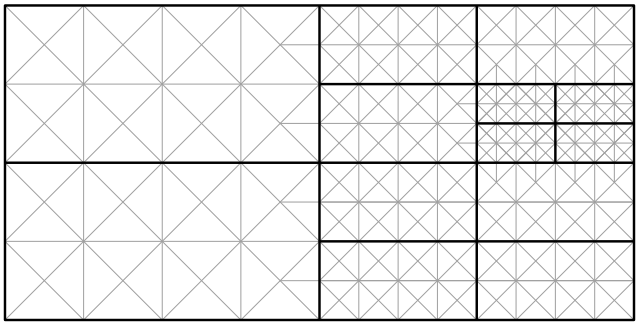

Figure 1.5: Triangulation of the quadtree variant used in the framework, with quads that are twice as wide as high.

being projected to cartesian coordinates. Second, sensor data stored in quads will exhibit sampling problems in near proximity to the poles.

Therefore, the presented data hierarchy is only suitable for a limited latitude range. In the range from -67.5° to $+67.5^\circ$ latitude, which corresponds to quadtree level 3 without the northernmost and southernmost quad rows and covers more than 92 % of the earth's surface, the degeneration of triangles is limited, and the longitude width of sensor data samples in proximity to $\pm 67.5^\circ$ is still roughly 38 % of the width of samples in proximity to the equator. See Fig. 1.6. The sampling problems and the consequential increase in data size are manageable in this range. For the pole regions outside this range, an approximative, lower-quality visualization can be achieved by artificially limiting the level of detail and ignoring the sampling problem in these regions.

An alternative quadtree-based representation of the earth's surface with the potential to avoid these precision limitations is described in Sec. 1.2.6. However, the current framework implementation is only concerned with the limited subset of the full latitude range as described above, and does not incorporate this extension.

1.2.3 Sensor Data Representation

The restricted quadtree as described in Sec. 1.2.2 allows to represent all sensor data sets in a single coordinate system and a single hierarchy. This avoids having to mix and match different projections and hierarchies at visualization time.

The hierarchy is suitable for many types of map-like sensor data. The framework currently implements support for DEMs, color images (e.g. orthophotos), and SAR images. To store a given sensor data set in the hierarchy, the following preprocessing steps have to be performed:

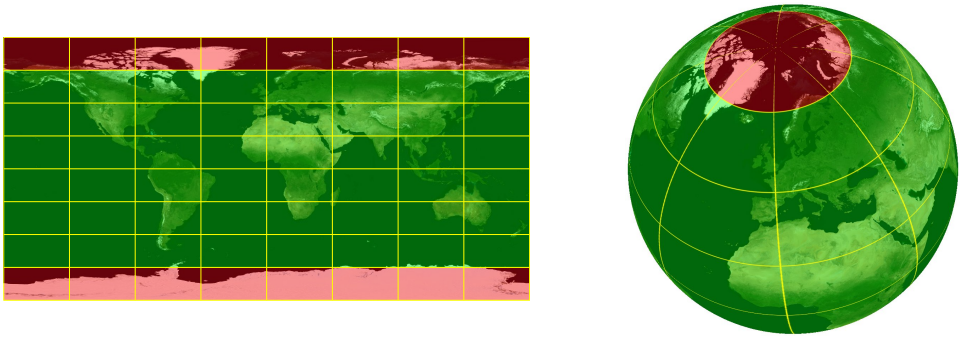

Figure 1.6: The latitude range from $-67.5°$ to $+67.5°$ (displayed in green) for which the data hierarchy is valid.

1. The data set must be projected into the WGS84 coordinate system. This coordinate system is in wide use in many applications, including the Global Positioning System (GPS). Many data sets are already available in a WGS84 representation, and data sets in other representations can be projected to WGS84 using widely available free software packages such as the Geospatial Data Abstraction Library (GDAL) [GDA].

2. The original resolution of a sensor data set in WGS84 representation will generally lie between two quadtree levels, as shown in Fig. 1.4. The data set is first resampled (using bilinear interpolation) to the next higher quadtree level, resulting in a magnification factor between 1 and 2. The representation for each quad in lower quadtree levels can then be computed by combining the four corresponding quads from the next higher level using averaging.

3. Quads that are not covered by the data set are left empty. Areas of a non-empty quad that are not covered by the data set are marked with a special `nodata` flag. This also allows to handle data sets with holes, as is common e.g. for various kinds of DEM data sets.

In addition to the unprocessed sensor data stored in the hierarchy, supplemental information about the data needs to be available to subsequent sensor data processing methods. This supplemental information is either given or can be computed while building the hierarchy. It belongs into one of the following categories:

- Global information, relevant for the complete data set. Examples for this category are the sensor data type and specifications, the area that is represented by the data set, global minimum and maximum

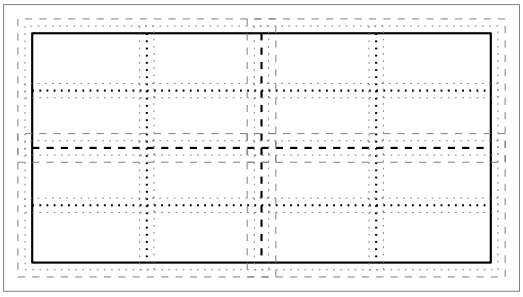

Figure 1.7: The quads of levels 0 (solid lines), 1 (dashed lines) and 2 (dotted lines) of the quadtree. The quads themselves are shown in black, and the extending borders are shown in gray.

values, and so on. This information can easily be stored alongside the quadtree hierarchy, e.g. in a structured text file.

- Quad-based information, relevant for the data stored in one quad. For example, in the case of elevation data it is useful to know the minimum and maximum elevation stored in each quad for level of detail purposes (see Sec. 1.4), and for SAR data it is useful to know some key statistical properties about the amplitude values stored in each quad (see Sec. 2.2). Such information must be stored in a metadata hierarchy that corresponds to the quadtree hierarchy.
- Local information, relevant for a limited area. For example, local data processing methods require access to a local neighborhood around one data sample. To make such data available without requiring expensive data fetches from neighboring quads, each quad is extended with an overlap area of a fixed size. See Fig. 1.7. This leads to slightly higher storage space and memory consumption for the enlarged quads in the quadtree hierarchy, but allows local processing methods direct access to local neighborhood information.

1.2.4 Data Storage and Management

Data Storage

The storage of hierarchical sensor data representations and their metadata must allow low-overhead access, for both local and network storage systems.

For this purpose, a quadtree hierarchy can be mapped directly to a directory hierarchy on a file system, with directories encoding the quadtree level l and the horizontal and vertical coordinates x and y of a quad in that

level. The most direct implementation would use directories `l/x/y`, but this can lead to low performance due to the large number of subdirectories in higher quadtree levels (level l contains 2^l quads both in horizontal and vertical direction). Therefore, other encodings that limit the number of subdirectories are preferable.

The framework currently implements a directory naming scheme of the form `l/[[[.../]`b_2`/]`b_1`/]`b_0`.dat`. The first directory is the level directory. The coordinates x and y are then combined into a single index, and this index is divided into blocks b_i, each 12 bits wide. The number of required blocks depends on the level l. These bit blocks are then transformed to hexadecimal representation and used as subdirectory names or, in case of the last block b_0, as the file name. That way, the number of entries in one directory is limited to $2^{12} = 4096$, and the number of subdirectory levels is kept low. For levels up to and including level 6, only a single block is necessary, since 12 bits are sufficient to store the combined index. For higher levels, more blocks are used, resulting in a deeper directory hierarchy.

Each data file contains the sensor data of the corresponding quad in a suitable file format that allows lossless compression. Quad-related metadata can be stored in the same file (depending on the file format) or in an adequately named additional file in the same directory.

A single file at the highest level of the directory hierarchy stores global information about the data set.

Such a simple directory hierarchy allows efficient filesystem-based access and maps directly to common network protocols such as HTTP and FTP, thus allowing network-based access with minimal overhead.

Data Transfer and Caching

An interactive visualization system must allow uninterrupted navigation and adjustment of parameters: the user interface must not block when new data has to be loaded in order to render the current scene. This results in the following requirements that a data management system must fulfill:

- If data that is necessary to render the current view is not yet available in GPU memory, a suitable approximation has to be used until the data becomes available.
- The data transfer from the storage hierarchy to GPU memory has to be done asynchronously to avoid blocking the rendering and user interface.
- Data transfers must use caching extensively to reduce the amount of time required to get data into GPU memory to a minimum.

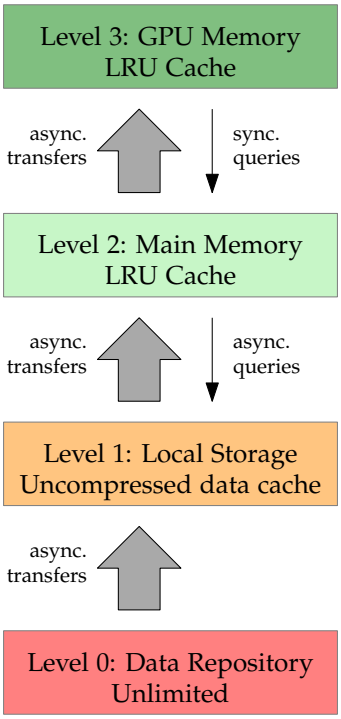

Figure 1.8: Multilevel cache hierarchy. A level can query a lower level to determine if missing data is available in that level, and subsequently initiate an asynchronous transfer of this data. Level 0 always contains all data. Cache levels with limited capacity use least-recently-used (LRU) caching strategies.

In order to fulfill these requirements, a multilevel cache hierarchy is necessary. This hierarchy must manage the following levels for each input data set (see Fig. 1.8).

Level 0: Sensor data repository. This is the storage hierarchy described in Sec. 1.2.4. The repository is accessed via network in the general case, but may also reside on a local storage system. The sensor data is stored in compressed form in this storage level.

Level 1: Local storage system cache. This cache level caches uncompressed data on the local storage system. It uses the same directory structure as the sensor data repository. Sensor data is stored uncompressed since many sensor data quads may be requested at the same time, and concurrent decompression of multiple quads would occupy too many CPU resources that are better spent on rendering and user interface tasks.

The local storage system cache may be limited in size. In this case, sensor data quads can be replaced on a least-recently-used basis.

Data transfers from level 0 to level 1 must be done asynchronously because network data transfers may potentially take a long time to complete or require multiple restarts.

Level 2: Main memory cache. This cache level stores the sensor data uncompressed and keeps it available for transfer to the GPU. The main memory cache is limited in size so that it does not congest the system's main memory pool. Sensor data quads are replaced on a least-recently-used basis.

Data transfers from level 1 to level 2 must be done asynchronously because each disk access can block the calling thread for an unknown period of time. For the same reason, a query to determine if a sensor data quad is available in cache level 1 must be done asynchronously, too.

Level 3: GPU memory cache. This cache level keeps the sensor data available for the visualization system. The available amount of GPU memory is typically much smaller than the available amount of main memory, thus the GPU memory cache is limited to a relatively small maximum size. Sensor data quads are replaced on a least-recently-used basis.

Data transfers from level 2 to level 3 can be handled in an asynchronous way by the GPU driver component. The driver guarantees that the transfer is completed when the data is first accessed, if necessary by blocking this first access for a short period of time. Since data transfers from main memory to GPU memory are usually very fast and the rendering system usually requires some CPU time before it triggers the first access to the data, this is acceptable, and it is not necessary to implement a separate thread to handle these data transfers. A query to determine if a sensor data quad is available in cache level 2 is a simple main memory access and thus can be done synchronously.

The visualization system determines which quads from which input data sets are required to render the current scene, and then requests these quads from the cache manager. A quad of a data set is identified by its quadtree level l and its horizontal and vertical coordinates x and y within that level.

The cache manager organizes data transfers, cache level queries, and quad approximations. It works as depicted in Fig. 1.9. This procedure ensures that each required quad will be transported from the lowest cache level to the highest. Special care must be taken to guarantee that multiple

```
quad q(l, x, y);

if (gpu_cache.sync_query(q) == FOUND) {
    return gpu_cache.get(q);
}
else if (memory_cache.sync_query(q) == FOUND) {
    gpu_cache.async_put(q);
    return gpu_cache.get(q); /* only a reference is returned here,
                                so the transfer does not block */
}
else {
    switch (local_storage_cache.async_query(q)) {
        case FOUND:
            memory_cache.async_put(local_storage_cache.get(q));
            break;
        case NOT_FOUND:
            local_storage_cache.async_put(data_repository.get(q));
            break;
        case UNKNOWN: /* an asynchronous query is still running */
            break;
    }
    quad r = gpu_cache.get_approximation(q);
    return r;
}
```

Figure 1.9: Management of cache data transfers and cache level queries, for a request of quad (l, x, y).

threads can access and modify the different cache levels concurrently, so that multiple transfers and queries can be active at the same time.

When the cache manager gets a request for quad (l, x, y) of a data set, and that quad is not readily available in GPU memory, the best available approximation has to be used. This approximation can be found by looking up the lower-resolution quads $(l - i, \frac{x}{2^i}, \frac{y}{2^i}), i = 1, \ldots, l$ and using the first one that is available in the GPU cache. The appropriate part of that quad provides the best available approximation for the required quad. To make sure that an approximation is always found, the single quad from level 0 of the data hierarchy is always kept in the GPU memory cache. (A quad could also be combined from its four children if these are all available in GPU memory, however since this specific case is rare and an approximation is only required for a very short period of time, it is usually not worth to implement this).

Using these caching strategies, the visualization system does not need to block the user interface when requested data is not yet available in GPU memory, as suitable approximations will be used instead. At the

same time, multilevel caching ensures that requested data is available in GPU memory as fast as possible.

The subset of sensor data quads that is required to render the current view of the scene is determined by level of detail techniques, as described in Sec. 1.4. Since the view typically varies smoothly over time, it is possible to apply heuristics that try to predict which quads will be required in the near future, and trigger transfer of these quads in advance so that they are immediately available when needed. Such techniques have been shown to improve data availability in terrain rendering applications [CGG*03, BGP09, DSW09]. However, this is currently not implemented in the framework.

1.2.5 Summary

This section described the fundamental hierarchical data structure implemented in the framework: a variant of a restricted quadtree that covers the WGS84 world map.

Restricted quadtrees are GPU friendly data structures that allow level of detail techniques to create conforming triangle meshes without the need for extra stitching geometry. This will be described in detail in Sec. 1.4. The meshes consist entirely of right-angled, isosceles triangles and are thus well-suited for GPU based mesh refinement, as described in Chapter 3.

At the same time, quadtree hierarchies can be mapped directly to file system directory hierarchies, for multi-resolution storage of remote sensing data sets. A multilevel cache manager transfers sensor data quads from the original data repository to a local storage cache, main memory, and finally GPU memory in a way that avoids interruptions of the visualization user interface.

The particular quadtree variant presented in this section is valid only for a limited latitude range. A possible extension for accurate planet-wide coverage is described in the next section.

1.2.6 Future Work

Sec. 1.2.2 describes limitations of the chosen quadtree variant with regard to accurate coverage of the earth's surface in proximity to the poles. These limitations are

1. degenerated triangles when transforming near-pole quad meshes to cartesian coordinates, and

2. sampling problems for sensor data stored in near-pole quads.

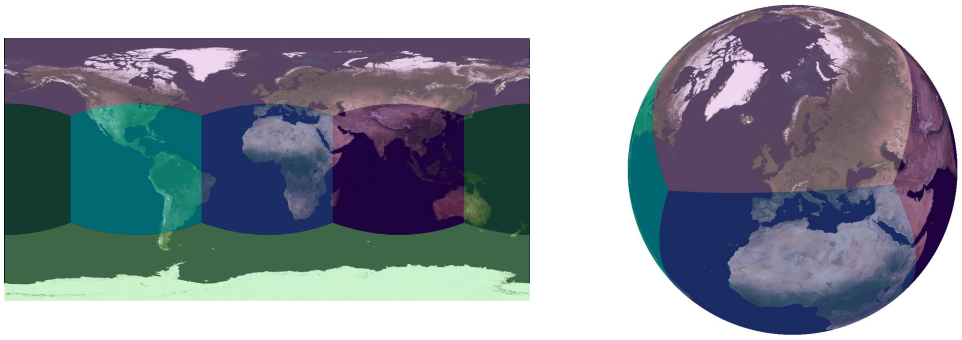

Figure 1.10: A subdivision of the planet surface into six equal interconnected areas, each managing its own quadtree hierarchy while keeping proper neighborhood relations with its four neighbor areas.

These limitations are common to all approaches based on quadtree or triangle bintree hierarchies, but are rarely investigated. Only a few works exist that explicitly address these problems.

Kooima et al. use a uniform sphere triangulation instead of a quadtree hierarchy, thereby avoiding triangle degeneration entirely. To reduce sampling problems, they interpolate sensor data from three different projections: a polar projection for each pole area and a spherical projection for the remaining area [KLJ*09]. However, their level of detail technique based on icosahedron subdivision cannot guarantee error bounds, and is therefore not suitable for accurate visualization of Remote Sensing data.

Cignoni et al. subdivide the sphere that represents the earth into six equal partitions similar to a cube map. Each partition then manages its own hierarchy structure, stores its sensor data in its own coordinate system, and generates its own triangle bin-tree based meshes [CGG*03]. This ensures that both triangle degeneration and data sampling pose no problem. Proper handling of neighbor relations at the borders of the partitions ensure a seamless visualization result.

This solution is also applicable to quadtree based methods such as the one presented in Sec. 1.2.2: the approach of subdividing a sphere into six equal areas, each holding a quadtree structure, was already used by White and Stemwedel as a means to store sky-related data in the FITS image format [WS92]. To use this approach in the framework, each of the six areas must maintain proper neighboring relations with each of its four neighbor areas, to allow consistent restricted subdivision of the six quadtrees for level of detail purposes. See Fig. 1.10.

With this approach, the creation of a single seamless mesh is still possible using the same methods that are described in the following sections. Since each of the six areas would use its own coordinate system to store

sensor data, sampling irregularities would be limited and acceptable for all areas on the earth's surface.

1.3 GPU-Based Data Processing

This section covers the infrastructure required for the GPU-based sensor data processing tasks. These tasks are the following (see also the right half of Fig. 1.1):

- The transformation of sensor data quads into elevation maps (if the input data set is a DEM) or color textures (if the input data set was produced by an imaging sensor technology).
- The fusion of multiple elevation maps or textures, from different data sets, into a single elevation map or texture.

These tasks take one or more two-dimensional maps as input and produce one two-dimensional map as output. GPUs are highly optimized for this kind of work, since one of their design goals is to efficiently apply textures (which correspond to multiple two-dimensional input maps) to two-dimensional regions of the output image (which corresponds to the two-dimensional output map).

However, to make optimal use of GPU computing resources, it is necessary to adhere to requirements resulting from the highly specialized GPU architecture.

This section starts with an overview of GPU technology advances in recent years, and then gives an overview of related work in the field of general purpose computations on GPUs. Based on this information, a GPU-based processing pipeline for sensor data is described, as well as a set of requirements that sensor data processing methods such as those described in Chapter 2 need to fulfill.

1.3.1 Background

In the last decade, graphics hardware became more and more flexible and powerful. It moved from a fixed function graphics pipeline model with a very limited set of supported input data types to a fully programmable stream processor model with full support for common input and output data types.

At the same time, thanks to the highly data-parallel nature of graphics computations, the computational power of GPUs in terms of theoretically achievable floating point operations per second (FLOPS) grew faster than that of CPUs [OLG*07]. Furthermore, GPUs have a very attractive

price/performance ratio since they have become off-the-shelf mass-market articles.

This transformation of GPU technology began with the introduction of *shaders* that allowed limited manipulation of the operations carried out by the graphics pipeline. First, *fragment shaders* allowed per-pixel computations, followed by *vertex shaders* for per-vertex computations. Initially, these shaders were only programmable using low level assembly language. Later, higher level shading languages were introduced, such as the OpenGL Shading Language (GLSL) or the DirectX High Level Shader Language (HLSL).

The classic graphics pipeline was then extended with the introduction of *geometry shaders* that allowed manipulation of the vertex stream, including the insertion and deletion of vertices. At the time of writing, the latest addition to the now fully programmable graphics pipeline are tessellation stages that can be used to add geometric detail on the fly.

In parallel to this increase in programmability, GPUs added support for more and more data types. For texture data, the $1-4$ components were originally each represented using a maximum of 8 bits. Meanwhile, texture formats include support for half, single, and double precision floating point components. Similarly, GPUs originally supported only simplified floating point computations internally (not IEEE 754 standard compliant, e.g. without infinity and not-a-number representations), but now support the same range of integer and floating point data types and computations that CPUs do (although some mathematical functions are not yet available in an IEEE 754 compliant double precision variant at the time of writing).

To be able to handle this much increased functionality, the GPU had to move from a fixed function design (where each component fulfilled a fixed role) to a general-purpose stream processor design. In the stream processing model, a set of compute cores is first configured to execute a particular task, and then processes a large set of data in parallel [Owe05]. In this model, the graphics driver can assign different tasks to different compute core sets on the GPU depending on demand.

The increased flexibility that the hardware had to offer, combined with its rapidly increasing performance at relatively low prices, made GPUs more and more interesting for applications beyond displaying graphics. Using a graphics pipeline centered API such as OpenGL or DirectX to program the GPU proved to be cumbersome for many general purpose tasks, and new ways to program GPUs independent of graphics concepts were sought after [SDK05]. As a consequence, new APIs and languages were developed to allow more flexible use of the hardware.

McCool et al. presented Sh, a C++ library that allows to program GPUs

from within the application's source code instead of using specialized languages such as GLSL [MDTP*04]. Buck et al. described a compilation and runtime system called Brook for GPUs, with which the GPU can be used as a streaming coprocessor [BFH*04]. McCormick et al. presented the Scout programming language and environment for interactive data manipulations on GPUs for analysis and visualization purposes [MIA*07]. Lefohn et al. introduced Glift, an abstraction and generic template library for complex GPU data structures which allows to separate data structures and algorithms [LSK*06]. All of these systems provide abstraction layers that hide the graphics centric GPU management core from the application programmer.

These research projects soon inspired vendor-specific solutions such as ATI/AMD's Close To Metal [AMD] and NVIDIA's CUDA [NVI], which allowed direct access to the GPU as a stream processor for the first time, thus eliminating the need to encapsulate graphics centric GPU access with abstraction layers. Today, all manufacturers of graphics hardware with stream processing capabilities also support the cross-vendor standard OpenCL [Khr].

1.3.2 Related Work

The improvements of graphics hardware described in the previous section made it more and more interesting to use GPUs to solve problems that are not directly related to graphics. This field is now often summarized by the term General Purpose Computations on Graphics Processing Units (GPGPU) [GPG].

Among the first general purpose tasks implemented on the GPU were image processing tasks, where one or more 2D input images are transformed into a 2D output image using the fragment shader of the graphics pipeline [Jar04, War05]. The reason is that this task maps ideally to the graphics pipeline. In the simplest form, one or more textures that contain the input images are rendered into an output framebuffer of the same dimensions using a simple quad, thereby establishing a one-to-one mapping of input to output pixels. The fragment shader that is executed for each output pixel then can gather the various input pixels, perform the desired computations, and write the result to its designated output pixel.

A thorough survey of early work in the GPGPU field was given by Owens et al. in 2007 [OLG*07]. A large part of this survey describes data structures and programming techniques that map common tasks to the graphics pipeline. This was necessary because at that time the graphics hardware was still very much centered around the classic graphics pipeline model, and graphics APIs such as OpenGL and DirectX were of-

ten the only feasible way to program GPUs.

However, the stream processing concepts described in this survey still apply to the more general architecture of todays GPUs. The techniques that are relevant to the sensor data processing pipeline of the framework are summarized below.

Separation of input and output. Graphics APIs require strict separation of input (textures) from output (framebuffer) data. While newer APIs do allow mixed input and output in the same memory region, this is highly inefficient. For best performance, the hardware driver must be able to schedule the per-sample computations on different compute cores, and to coalesce the resulting memory writes into a contiguous memory area [NVI10].

Input gathering and avoidance of scattering. Gathering is the process of reading input data from different memory locations, and scattering is the process of writing output data to different memory locations. Graphics APIs allow gathering by texture sampling, while implementing scattering requires indirections. Newer APIs allow scattering to be implemented directly, but again, this is highly inefficient because the hardware driver must be able to coalesce memory writes into a contiguous memory region for best performance. While there are different techniques to implement scattering in relatively efficient ways using intermediate steps, it is usually better to reformulate a given algorithm to avoid scattering if possible. This is the case for many types of sensor data processing methods [LKNK07].

Branching techniques. Older GPU generations had very limited support for branching in shaders: initially it was only possible for an output operation to choose between two computed results based on a comparison value [War05]. Various techniques were used to avoid in-shader branching by moving the branching to other parts of the graphics pipeline. Newer GPUs have much improved support for branching, but it is still important to avoid branching in compute kernels if possible, even more so than it is to avoid inner-loop branching on CPUs. One reason is performance, and the other reason is to avoid hitting hardware limits such as the maximum number of instructions allowed in a compute kernel or the maximum number of required registers. As a result, it is still beneficial to move branching out of the compute kernels and into other parts of the program flow, and to choose one of multiple highly specialized compute kernels instead of using a single combined one that uses branching.

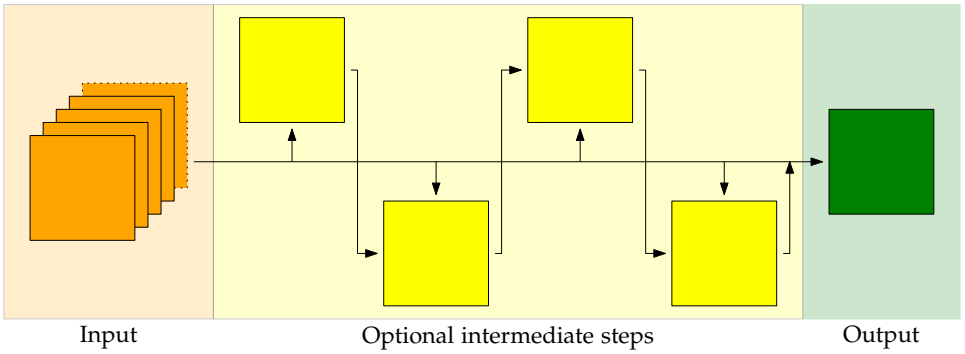

Figure 1.11: Processing pipeline.

1.3.3 Processing Pipeline

The aforementioned basic image processing technique, where input images are mapped to an output image by rendering a quad, can be extended with support for intermediate processing steps. Combined with the extended support for texture and framebuffer data types in current GPUs, in particular floating point types, this results in a general processing chain that allows the implementation of many classes of sensor data processing methods. See Fig. 1.11. This processing chain takes one or more sensor data maps as input and produces one output map. Usually, this output map will contain an elevation map or a texture. Optional intermediate processing steps store their results using ping-pong buffers.

To apply this processing pipeline to the task of transforming sensor data to elevation maps or images, usually only one sensor data quad is used as input. Depending on its complexity, a transformation method might require one or more intermediate steps to produce the output. For example, a separable Gauss filter would usually be implemented using one intermediate step. Many classes of sensor data processing methods can be implemented using this method [LKNK07]. See Chapter 2 for details and examples.

To apply the processing pipeline to the task of fusing multiple elevation maps or textures into a single elevation map or texture, multiple input quads are used. The currently implemented fusion filter uses simple blending of input maps, but supports special `nodata` flags that mark areas for which the input data set provides no data. This simple fusion filter requires only one processing step. More advanced fusion filters, e.g. DEM fusion filters with intelligent `nodata` hole filling techniques, might use multiple intermediate steps.

Note that the original sensor data quads contain an overlap area that allows sensor data processing methods to access local neighborhoods with-

out expensive data fetches from neighboring quads (see Sec. 1.2.3). After sensor data processing and fusion, this overlap area is no longer required, so that the output quad of the fusion step may be smaller than its input quads. However, a minimum border of one sample must be kept in order to avoid interpolation artifacts in the rendering stage. This applies to both elevation maps and textures.

The processing pipeline is used for each quad of the restricted quadtree hierarchy that is necessary to render the current scene, as determined by appropriate level of detail techniques (see Sec. 1.4). Currently, these computations are repeated for each rendered frame. Changes of processing parameters take effect immediately, and the user gets immediate feedback on interactions. It would be possible to cache the processing results and recompute only when relevant processing parameters have changed, but this would require substantial amounts of graphics memory and is thus currently not implemented in the framework.

The current implementation of the processing pipeline in the framework is based on OpenGL, since it maps directly to the graphics pipeline. However, it can of course be replaced by e.g. an OpenCL implementation should the need arise.

1.3.4 Summary

This section described a GPU-based processing pipeline for two central tasks in the visualization framework: the transformation of sensor data to elevation maps and textures, and the fusion of multiple elevation maps or textures into a single elevation map or texture, as shown in Fig. 1.1. The processing pipeline uses the GPU to achieve the necessary processing speed. Processing methods must adhere to a set of requirements to make full use of the GPU's potential.

The implementation of different classes of sensor data processing methods using this pipeline is demonstrated in Chapter 2 based on the example of SAR images.

1.4 Level of Detail and Rendering

This section describes the rendering part of the framework. This part consists of three components:

- A basic level of detail technique that chooses the subset of data from the quadtree hierarchy that is required to render the current scene. This is the CPU-based part shown in Fig. 1.1.

- A technique to build a single, seamless triangle mesh from the chosen quadtree subset. This is included in the GPU-based part shown in Fig. 1.1. Note that only the creation of an initial mesh is discussed in this section. The refinement of this mesh for final level of detail computations is discussed in Chapter 3.

- The rendering technique itself, including support for distributed rendering e.g. in Virtual Reality installations. This, too, is included in the GPU-based part shown in Fig. 1.1.

The main challenges associated with these components are the following:

- Since the terrain data is generated dynamically, only limited supplemental information is available. In particular, the level of detail technique cannot rely on precomputed error estimates.

- The generated mesh spans an arbitrary area on the planet's surface, and not just a limited local patch relative to a plane. Therefore, the underlying earth model has to be taken into consideration, and geometry computations are subject to special precision requirements.

- The rendering technique must allow mesh refinement techniques for final level of detail computations as described in Chapter 3. It must have efficient access to input data hierarchy information, without the need to traverse tree structures on the GPU. Furthermore, the rendering state must be representable in a data structure that allows distributed rendering across multiple hosts and/or graphics cards.

In the following, the components as implemented in the framework are described in detail.

1.4.1 Related Work

Pajarola and Gobetti's survey shows that quadtree hierarchies are commonly used in the context of terrain rendering both because they provide a hierarchical structure for the terrain data (elevation maps and images) and because they can easily be used for level of detail techniques [PG07]. As mentioned in Sec. 1.2, earlier terrain rendering methods used quadtree hierarchies for full level of detail computation and then generated a minimal or near-minimal triangle mesh for the resulting quadtree. Generally, such methods split a quad into four sub quads if it does not fulfill an acceptance criterion, and then ensure that neighboring quads are recursively split, too, if that is necessary to keep the quadtree restricted. When

all quads fulfill the acceptance criterion, the minimal conforming triangle mesh is created and rendered [LKR*96, Paj98, DWS*97]. However, this approach is not efficient anymore because the CPU cannot generate this mesh fast enough to saturate the GPU. Newer methods prefer to choose and combine larger blocks of triangles on the CPU for rendering on the GPU, instead of working on triangle level [PG07].

Traditionally, terrain rendering methods assumed that the input geometry and texture data is static and cannot change. Under this assumption, elaborate offline preprocessing can be applied to the original sensor data to produce highly optimized geometry and texture data representations, including precomputed level of detail information. For example, Duchaineau et al. [DWS*97] and Bösch et al. [BGP09] precompute object-space error metrics. In contrast, a fully interactive visualization system must compute the level of detail based on dynamically generated terrain data, where only limited supplemental information is available.

Creating and rendering triangle meshes for an area on the earth's surface (as opposed to a limited local reference plane) leads to two precision problems. First, the single precision floating point data type that is commonly used in the graphics pipeline does not provide enough accuracy for cartesian coordinates, resulting in jitter and cracks. Thorne noted that due to the non-uniform resolution of this data type, precision problems can be avoided by rendering the scene relative to the viewer, i.e. with viewer coordinates $(0, 0, 0)$ [Tho05]. This makes sure that inaccuracies and loss of precision affect only regions far away from the viewer, and thus do not result in visible screen space errors. As a generalization of this method, all vertex computations (e.g. for the purpose of mesh refinement) can be performed in a local reference frame to reduce numerical inaccuracies. This has been used by the few terrain rendering methods that explicitly address the planetary scale [CGG*03, KLJ*09]. Terrain rendering methods that ignore these precision problems are generally only applicable to scenes of limited size.

The second precision problem that may arise is limited depth buffer precision. In some terrain rendering situations, it is difficult to determine good near and far plane values, and as a result the available depth buffer precision may not be sufficient to accurately render the scene. This is traditionally solved by arbitrarily limiting the far plane, e.g. using fog effects, and/or by dividing the frustum into multiple partitions, each with enough depth precision, and then rendering the scene in multiple passes.

1.4.2 Level Of Detail

In the level of detail technique used in the framework, the quadtree hierarchy is only used to determine a first, conservative estimation of the final level of detail. The purpose of this step is to determine the sensor data quads that are required to render the scene, and to create a single conforming triangle mesh that is suitable for further refinement depending on the elevation map data generated from the sensor data quads (such refinement methods are detailed in Chapter 3). This triangle mesh consists only of right-angled, isosceles triangles, and is free of T-junctions that could result in cracks in the rendered image. These mesh properties also guarantee that elevation maps are sampled in a regular pattern.

This level of detail technique works as follows:

1. Determine the view frustum of the current scene.
2. Start with quadtree level 0, which contains a single quad representing the whole planet surface.
3. Compute a bounding volume for the current quad.
4. Project the bounding volume to screen space.
5. If the quad covers more screen space pixels than it provides map samples, split it into four sub quads.
6. Enforce the restricted quadtree property by recursively splitting neighboring quads as needed.
7. Repeat steps 3 – 6 for all newly created quads, until no quads need to be split anymore.

In the first step, the view frustum must be determined. The view frustum is defined by the viewer position and orientation and by the display setup, except for the near and far plane. Since the final scene geometry is not yet known, finding optimal values for the near and far plane is not possible. By using inner and outer bounding spheres for the earth to model the maximum extent of the final geometry, it is possible to find acceptable values for many common situations. See Fig. 1.12. Such situations include from-space views that show the whole planet, and close-ups that show only a limited part of the surface. However, for one common situation, namely flat views across the surface to the horizon, near and far plane are often too far apart to give enough depth buffer precision. If this is the case, it is necessary to split the view frustum into several parts along the viewing direction, each using near and far plane settings such that the

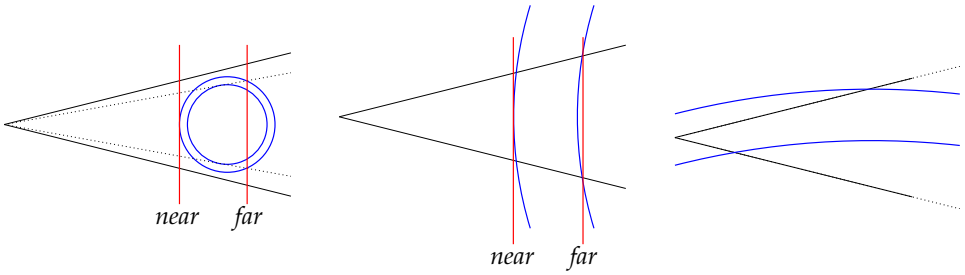

Figure 1.12: Near and far plane settings (red) using inner and outer bounding spheres (blue) for the earth in three common situations: a from-space view of the whole planet (left), a close-up of a limited surface area (middle), and a flat view to the horizon (right). In the latter case, setting near and far plane requires special handling.

resulting depth buffer precision is acceptable, and then render the scene in multiple passes.

After determining the view frustum, bounding volumes have to be computed for each quad and then projected to screen space. To compute a bounding volume for a quad, its minimum and maximum elevation values must be known. To make these values available, three requirements have to be fulfilled:

- Each DEM data set hierarchy must store the minimum and maximum elevation value for each quad in the form of quad-based metadata, as described in Sec. 1.2.4.

- The processing method that generates elevation maps from DEM data quads must provide the minimum and maximum elevation values of the generated maps.

  Note that it is not necessary for this purpose to scan the generated elevation map for its minimum and maximum values. Instead, the processing method can be defined in a way that allows to guarantee bounds for the output values when given bounds for the input values.

- The data fusion method that combines several input elevation maps into a single elevation map must combine all input minimum/maximum pairs into a single minimum/maximum pair. Again, it is not necessary at this point to scan the output quad for minimum and maximum values; it is sufficient to combine the given input ranges.

Once a bounding volume is computed, view frustum culling can be applied at this early stage with no additional cost: if the projected bounding

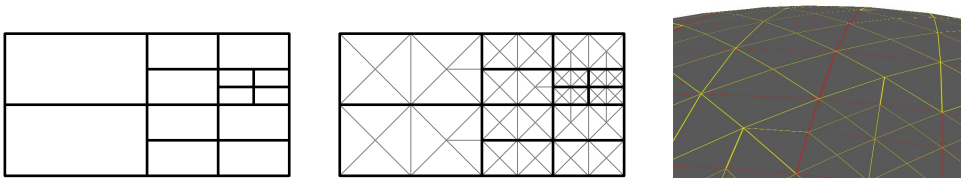

Figure 1.13: Mesh creation and rendering. A restricted quadtree is given by the level of detail mechanism (left). This quadtree is triangulated using a predefined refinement level l (middle). For rendering, the resulting mesh must be transformed from quad coordinates to cartesian coordinates (right).

volume is outside the visible area, then it does not need to be subdivided or rendered, and consequently the sensor data for this quad does not need to be transferred or processed.

1.4.3 Mesh Creation

The mesh creation step creates a triangle mesh for the restricted quadtree that determines the initial level of detail estimate, as described in the previous section. See Fig. 1.13. This mesh consists entirely of right-angled, isosceles triangles, which allows further refinement on the GPU as described in Chapter 3.

At the mesh creation step, it is possible to choose the initial mesh refinement level. As explained in Sec. 1.2.2, refinement level 0 corresponds to the minimal conforming triangulation of the restricted quadtree, and refinement level l corresponds to the minimal conforming triangulation after subdividing each quad l times. Thus, higher refinement levels result in finer meshes with more triangles per quad. See Fig. 1.3.

In the framework, quads provide $2^{k+1} \times 2^k$ samples. See Fig. 1.5. The maximum initial refinement level that makes sense for these quads is $l_{\max} = k$, because at this level all elevation map information is covered, and further refinement would not reveal additional geometric detail. Thus, the mesh at refinement level $l_{\max}$ would result in an accurate rendering of the scene. However, meshes at this level are too large and detailed to be rendered at sufficiently high frame rates. It is thus necessary to choose a lower level l and apply mesh refinement only where necessary. See Chapter 3 for a discussion of the parameter l.

In summary, the rules to generate an initial mesh of refinement level l for the current quadtree hierarchy are the following (see Fig. 1.3 and Fig. 1.5):

- Divide each quad into $2^{l+1} \times 2^l$ square sub quads.
- Generate 4 triangles for each square sub quad, by dividing it using its two diagonals.
- If the hypotenuse of a triangle is shared with a neighboring quad, and this quad is of a higher level than the current quad, split the triangle into two smaller triangles by splitting the hypotenuse at its center.

Note that the core mesh inside each quad is the same for all quads, and only the border triangles may differ. Furthermore, there are only 16 possible border triangle setups, depending on whether the neighboring quad at each of the four sides has a higher level or not. See Fig. 1.5. Thus, for a given level l, the core mesh as well as the 16 border triangle setups can be precomputed and stored in vertex buffer objects on the GPU. To generate the mesh, it is then only necessary to combine these precomputed meshes on the GPU.

In order to make quadtree hierarchy information available to the GPU-based mesh refinement and rendering steps without having to perform expensive tree traversals on the GPU, the quads that need to be rendered are enumerated, and the quad index q is stored as a vertex attribute. The quadtree information can then be stored in a set of arrays that can be accessed using this index. These arrays store the following information:

- Quad level and coordinates (l, x, y), to identify each quad and to know its location on the WGS84 map.
- Indices of neighboring quads. Each quad can have up to two neighbors in each direction, so a total of 8 neighbor indices needs to be stored for each quad. If a neighbor does not exist, -1 is stored in its index field.

The currently required elevation map and texture can be identified using the quad index vertex attribute, and the texture coordinates correspond to the original quad-space vertex coordinates.

The result of the mesh creation step is a single mesh for the restricted quadtree, consisting only of right-angled, isosceles triangles. This mesh initially is in quadtree space, which corresponds to WGS84 coordinates as explained in Sec. 1.2.2. In order to allow fast GPU-based access to quadtree information, a vertex attribute carries the index of the current quad.

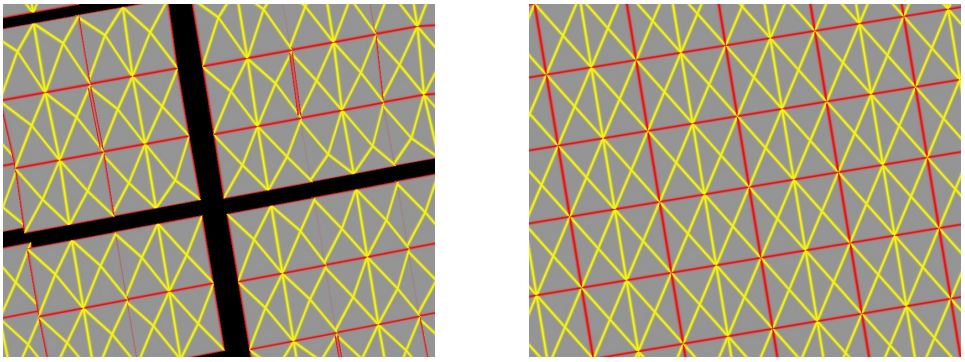

Figure 1.14: Demonstration of precision problems with single precision floating point computations. The left image shows the mesh for a ground area viewed from 36 m altitude, with the viewer-relative world coordinates computed using a single precision floating point data type. The right image shows the same scene, but with viewer-relative world coordinates computed using double precision.

1.4.4 Rendering

To render the mesh, each vertex must first be transformed to cartesian world space coordinates, which then allows the usual graphics pipeline transformations to screen space. See Fig. 1.13.

The transformation of quad-relative coordinates to cartesian world space coordinates works as follows. First, the quad index q is extracted from a vertex attribute. Then the elevation h for the vertex is determined by sampling the elevation map corresponding to quad q at the quad-space vertex coordinates (x_q, y_q). Then, the quad coordinates (l, x, y) and quad-space vertex coordinates (x_q, y_q) are combined and transformed to WGS84 latitude/longitude coordinates (ϕ, λ). The complete three-dimensional WGS84 coordinates (ϕ, λ, h) can then be transformed to cartesian coordinates (x, y, z) using the formulas defined by the WGS84 standard.

To avoid precision problems with the single precision floating point data type used by the graphics pipeline, the cartesian world space coordinates must be transformed to be relative to the viewer by subtracting the viewer position. In order to get an accurate result that can then be passed to the single precision graphics pipeline, the first computation steps, i.e. the transformation to world space and the subtraction of the viewer position, must be done using a double precision data type. See Fig. 1.14. This data type is available on current GPUs, and consequently the mesh transformation can be performed entirely on the GPU. Note that double precision computations are still relatively expensive on the GPU, but the performance gap between single and double precision computa-

tions is narrowing with each new generation of GPUs [NVI09b].

Once the mesh is transformed to single precision viewer-relative world space coordinates, it can be rendered using the standard graphics pipeline. This allows the integration of commonly used effects such as per-pixel lighting, atmospheric effects, ocean reflections, or clouds [Mal06]. The correct texture for each quad is determined by the quad index q. To allow seamless texturing without interpolation artifacts at quad borders, an additional quad border of one pixel is required, as explained in Sec. 1.3.

1.4.5 Distributed and Parallel Rendering

To allow distributed rendering, the visualization framework must encapsulate the render state, i.e. the set of all information that together defines the current scene. The main application process manages the master render state, and may distribute it over a network.

In the framework, the render state consists of the viewer position and orientation, the list of active data sets and their processing parameters, and additional rendering parameters such as the maximum allowed screen space error for the mesh refinement step.

To render a view of the scene, a renderer must know its view frustum in addition to the render state. With this information, the scene can be rendered for different displays, managed by different GPUs (e.g. on visualization workstations), potentially residing on different hosts (e.g. the nodes of a render cluster): each display renders its own view of the scene defined by its own view frustum.

In a common desktop environment, only a single view of the scene is rendered. For high resolution display walls or virtual reality systems, multiple synchronized render states exist on multiple hosts, and the scene may be rendered on multiple GPUs using different view frustums.

Each render process builds its own restricted quadtree for rendering, depending on its view frustum. Thus, each render process decides which sensor data quads are required for its own view, and needs to transfer and process only these quads.

The multilevel cache hierarchy can be distributed for maximum efficiency: all render processes can share a single on-disk cache stored on a fast, shared network filesystem, and multiple GPUs on a single host can share a single main memory cache on that host.

By guaranteeing a low screen space error using the mesh refinement methods described in Chapter 3, it is possible to ensure a consistent view across neighboring displays without a visible transition between them.

The current implementation of distributed and parallel rendering in the framework, in particular the synchronization of render states and the

configuration of multiple views, is based on the Equalizer Parallel Rendering Framework [EMP09, Eil].

1.4.6 Summary

This section describes the rendering part of the framework. The three main aspects of this rendering part are the initial level of detail technique that determines which parts of the sensor data hierarchy are required for accurate rendering of the scene, the mesh creation technique that builds an initial mesh by efficiently combining larger triangle patches, and the rendering technique that handles precise coordinates on a planetary scale and uses flattened quadtree information stored in arrays to access hierarchy information. Furthermore, the rendering part is built in a way that allows distributed rendering, e.g. for virtual reality installations. Mesh refinement methods for final level of detail computations are described in Chapter 3.

1.5 Visual Assistance Tools

The interactive approach to visualization of Remote Sensing data leads to many opportunities for the user to choose and adjust methods and parameters. This flexibility comes at the price of increased complexity. Finding a suitable set of visualization parameters may become time consuming and/or non-intuitive.

Visual assistance tools are integrated into the user interface of visualization systems to help manage the complexity of the parameter adjustment task. In the context of Remote Sensing data visualization, the main challenges associated with visual assistance tools are the following:

- The assistance tools must be designed for application area experts, and not for visualization experts.

- Since the assistance tools must steer parameters of subsequent data processing and fusion methods, they must work on the data representation level that is available before data processing and fusion, i.e. hierarchical sensor data with little or no supplemental data. See Fig. 1.1.

This section describes visual assistance tool concepts and their application to Remote Sensing data visualization.

1.5.1 Related Work

As computer hardware in general and graphics hardware in particular become more powerful, visualization systems move towards more interactive control over the visualization process. This leads to more powerful and flexible applications, but at the cost of increasingly complex user interfaces.

Designing intuitive and manageable user interfaces is a common challenge in all application areas of interactive visualization systems. An important consideration in this regard is that the targeted user group of visualization systems are application area experts, but not necessarily visualization experts. As a consequence, user interfaces for visualization systems are highly specialized for their application area. For example, for the field of interactive visualization of medical volume data, Rezk-Salama et al. propose a system that introduces application area semantics to the visualization user interface [RSKK06].

This section focusses on two user interface concepts that are applicable to the field of interactive visualization of Remote Sensing data: *lenses* and *detectors*.

A *lens* allows two sets of visualization parameters to be active at the same time: one global set, and one local set that only applies to the region of the lens.

In the original work by Bier et al. [BSP*93], a lens defines a rectangular sub-region of the two-dimensional screen space. In this sub-region, visualization methods differ from those used in the rest of the screen space. This two-dimensional screen space lens is mainly applied to the drawing of two-dimensional shapes, but an initial example of a 3D scene is also given.

Viega et al. introduced a three-dimensional volumetric lens that is a sub-region of the three-dimensional object space instead of the two-dimensional screen space [VCWP96]. This volumetric lens is used to inspect 3D objects.

Borst et al. adapted the concept of volumetric lenses to specialized applications in the domain of geospatial information visualization, and discussed aspects of a GPU-accelerated implementation [BBBK07].

All these lens concepts are tailored to their specific application area.

A *detector* allows an interactive visualization system to find interesting features in the sensor data in real time. The system can then display visual hints for each detected feature, thereby guiding and assisting the user in exposing the important detail of the data.

A detector usually specializes in finding well-defined features in a data set with known properties. Consequently, a very wide range of detectors

exists, each tailored for a special application, and often consisting of multiple processing steps. For example, Brekke and Solberg proposed specialized methods to detect oil spills in SAR images [BS05]. Such specialized detectors are not designed to be used in interactive applications, and consequently are far too slow for this purpose.

Diard demonstrated an efficient GPU-based implementation of a feature detector using the geometry shader stage of recent GPUs [Dia08], suitable for interactive use. Nevertheless, even with an optimized GPU-based implementation, detectors for interactive applications need to be restricted in computational costs and complexity.

1.5.2 Lenses

The lens concept allows to compare two different parameter sets directly, thus giving additional insight into the data and the effects of parameters, and simplifying the choice of a particular parameter set.

The implementation of the lens concept is specific to the application. Previous two-dimensional and three-dimensional lenses all have in common that the geometry of the scene must be known at the time the lens is applied, so that the part of the geometry that is relevant to the lens can be handled in a special way.

In contrast, in the situation of interactive visualization of Remote Sensing data, the geometry is not known before the lens is applied, because the geometry depends on the visualization parameters defined by the lens. For example, a lens might define processing parameters for a DEM data set that differ from the global parameters, resulting in different geometries. Therefore, a lens in the framework must be defined in terms of a geometry that is fixed regardless of visualization parameters.

To this end, the lens used in the framework is defined as a two-dimensional circular area on the planet surface, given by a center and a radius. Since WGS84 coordinates are used in the framework, the WGS84 ellipsoid is used as the reference surface. See Fig. 1.15.

Each quad of the quadtree hierarchy describes an area on the WGS84 ellipsoid (see Fig. 1.4), and for each quad one of the following three cases applies:

1. The quad and lens areas do not intersect. Only the global processing parameters apply to the quad.
2. The quad area is a subset of the lens area. Only the lens processing parameters apply to the quad.
3. The quad area is partly inside and partly outside the lens area. Both the global and lens parameter sets apply to the quad.

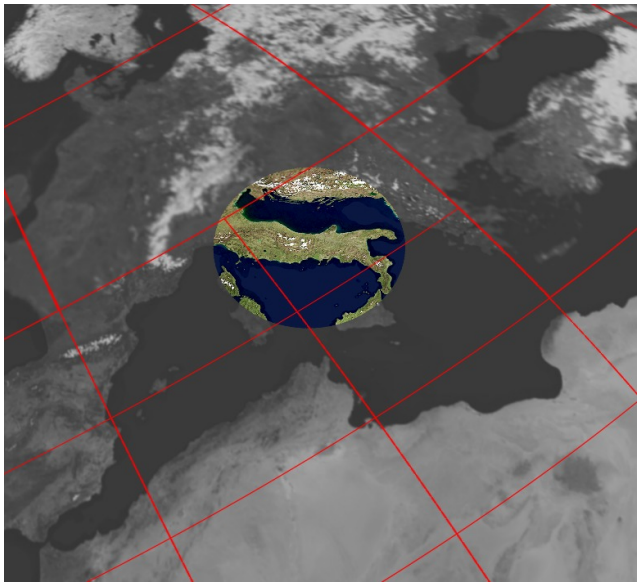

Figure 1.15: A circular lens, defined by center and radius, on the WGS84 ellipsoid surface. Different processing parameters are applied for regions inside and outside the lens.

In the latter case, the quad is processed with both parameter sets, and an additional postprocessing step is applied that combines the two processing results into a single quad by determining for each sample whether it lies inside or outside the lens. This double processing for a small number of quads is the only computational overhead caused by the usage of lenses.

Since this lens concept is applied at the sensor data processing stage, it is completely transparent to the rendering stage. In particular, it is guaranteed to produce a consistent and crack-free geometry even at the border of the lens. This is in contrast to previous two-dimensional and volumetric lenses, where different visualization methods lead to inconsistencies at the lens borders.

1.5.3 Detectors

The purpose of a detector in the context of interactive visualization is to draw the user's attention to important aspects of the data, and to give the opportunity to quickly examine these aspects for a more detailed analysis.

For this purpose, a detector must consist of three components:

- A feature detection component that scans the current set of hierarchical sensor data for relevant features.

- A display component that provides a visual hint about location and basic attributes to the user.
- An analysis component that presents the user a detailed view and examination of a detected feature on demand.

All three components need to be highly specialized for one kind of sensor data and one kind of feature to be effective. In Sec. 2.4, a detector for near-ideal reflectors in SAR amplitude images is presented in detail.

The feature detection component works on sensor data quads managed by a quadtree hierarchy, and thus must take the different resolution levels into account. Detectors that look for features with similar sizes in world space must adapt to the resolution level. For example, if a feature spans n samples in quadtree level l, then it spans $\frac{n}{2}$ samples in level $l-1$ and $2n$ samples in level $l+1$. A detector for this feature will only work in the limited range of levels that represent the feature with adequate detail. On the other hand, detectors that look for features with similar sizes in screen space (and thus varying scales in world space) might choose to ignore the resolution level. For example, certain terrain features occur at multiple scales (this self-similarity allows to use fractal methods for the generation of artificial terrain [MKM89]), and a detector for such features might choose a fixed mask size regardless of the quadtree level, so that such features will be detected at the scale given by the current region of interest.

A detected feature initially only has an associated two-dimensional location on the WGS84 ellipsoid, because the full geometry of the scene is not known at this stage of the framework. The display component usually requires the full three-dimensional location. This information can be acquired after the processing and fusion stage, by sampling the fully processed and fused elevation map corresponding to the quad that contains the feature.

To be interactively usable, a detector must focus on speed rather than accuracy, and thus can only give a rough hint about features in the scene. A detailed examination of a feature by the user, possibly with the help of the analysis component of the detector, is always necessary.

1.5.4 Summary

In this section, two user interface concepts are described that help the user in managing the potentially complex task of parameter adjustment. Lenses allow focused comparisons of different parameter sets, thus helping to understand the effects of parameters. Detectors point the user to important features in the data, thus allowing directed parameter adjustments.

Both concepts are applied at an early stage of the visualization framework, and thus must work on hierarchical sensor data representation. While lenses are a general concept applicable independent of the type of sensor data, detectors must be highly specialized to be effective. A detailed description of such a specialized detector is given in Chapter 2.

1.6 Results

This chapter described a framework for GPU-based interactive visualization of remote sensing data. Its main components are

- A quadtree-based data hierarchy and management module.
- A GPU-based processing chain for sensor data.
- A level of detail technique to choose the suitable subset of data from the hierarchy.
- A rendering component to render the processed and fused elevation maps and textures.
- Visual assistance tools to help the user manage the various interactively adjustable parameters.

The hierarchical data structure is a restricted quadtree based directly on the WGS84 map. It is currently valid for a limited latitude range, spanning more than 92 % of the planet surface, but can be extended to be valid for the complete planet by subdividing the planet surface into six equal areas with proper neighboring relations; the techniques presented in this chapter still apply.

For each sensor data type that is visualized using this framework, specialized methods must be implemented for GPU-based data processing and fusion, and for visual assistance tools such as detectors. Chapter 2 will give a detailed description of such specialized methods for SAR images.

The level of detail technique uses a conservative approach to determine the restricted quadtree subset that is necessary to render the scene. An initial mesh can be constructed from this subset by combining larger triangle patches directly on the GPU. This initial mesh consists only of right-angled, isosceles triangles and is free of T-junctions.

The final level of detail for accurate rendering lies in the responsibility of a GPU based mesh refinement technique, as described in Chapter 3. This technique must work on dynamically generated terrain data and guarantee screen space error bounds. The output of this technique is

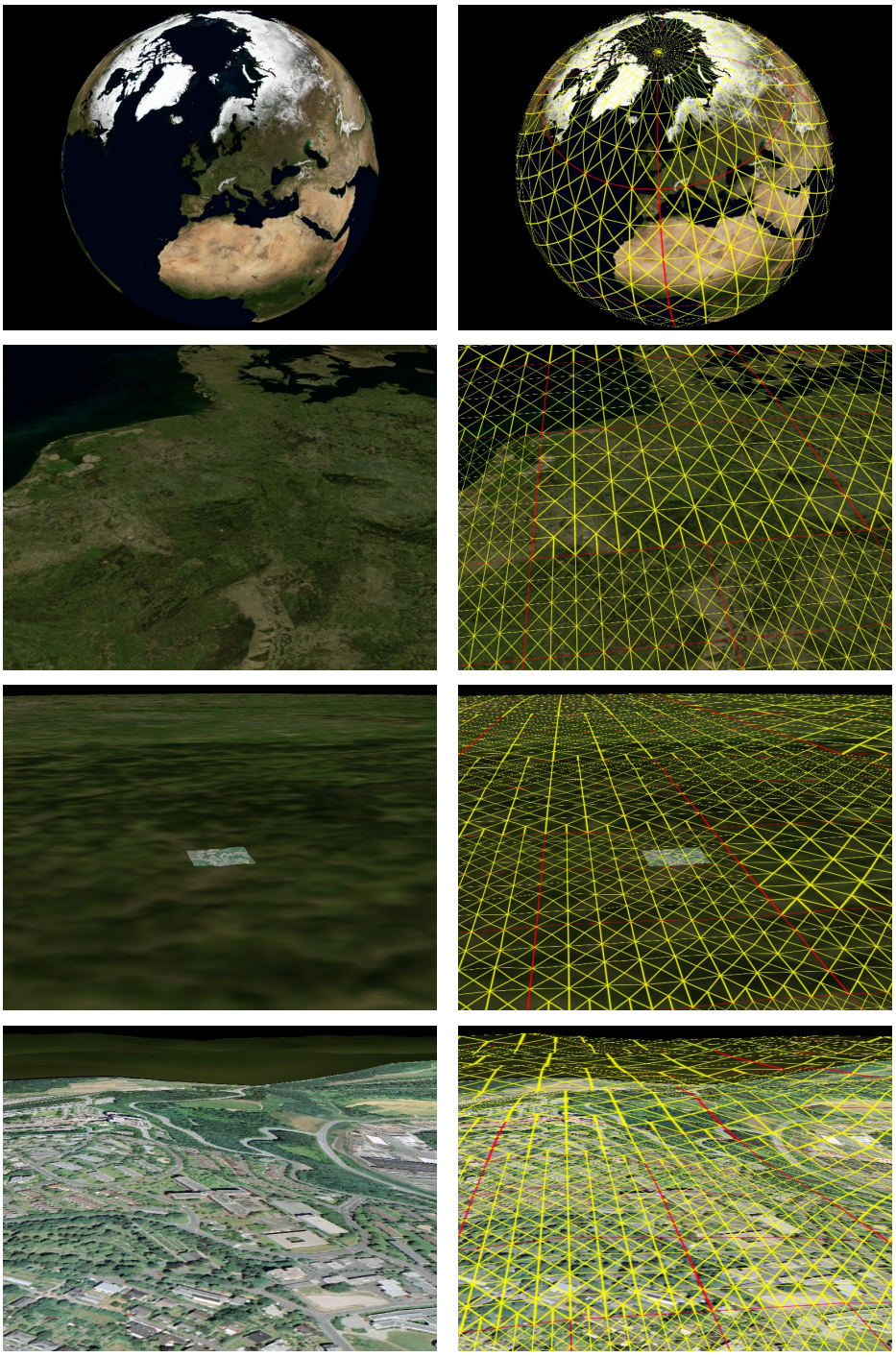

Figure 1.16: An image sequence zooming in on the University of Siegen, Germany. The images on the right show the restricted quadtree generated by the level of detail technique (red), and an initial conforming triangle mesh of level $l = 2$ (yellow).

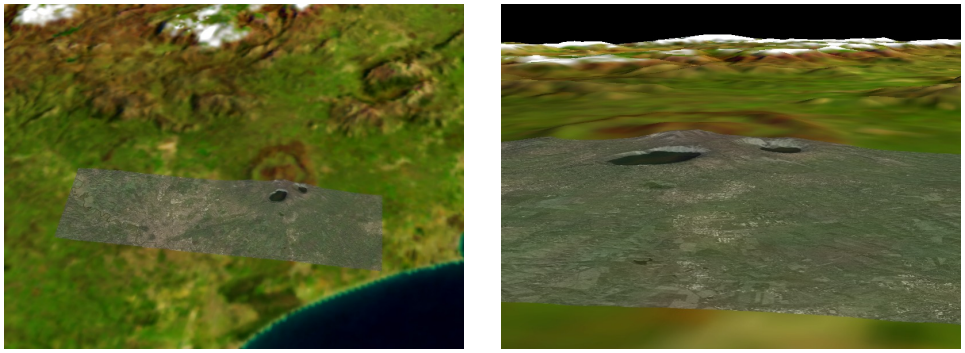

Figure 1.17: The area around Rome, Italy, in a combined view of two DEM data sets, a false color image, and a SAR amplitude image.

a triangle mesh with the same properties as the input mesh (right-angled, isosceles triangles, no T-junctions). This output mesh can then be rendered using the techniques described in this chapter.

In the following, example results are shown that demonstrate the integration of the different presented framework components. These example results were produced on a PC with an Intel Core 2 Duo 3 GHz CPU, 8 GB main memory, and an NVIDIA GTX 285 graphics card with 2 GB of graphics memory. The software environment consisted of Debian GNU/Linux 5.0 64bit, NVIDIA CUDA 2.3 and the NVIDIA graphics driver version 190.53.

As described in Sec. 1.2.2, the size of a quad is $2^{k+1} \times 2^k$ samples, plus an additional border to allow local neighborhood access during data processing. All following examples use a quad size of 512×256 samples ($k = 8$) and a border of 9 samples. This quad size was found to be a good compromise between the initial level of detail estimate granularity, GPU data chunk handling performance, and data storage overhead. The border size was chosen so that all SAR processing methods have access to local neighborhoods of up to 19×19 samples (see Chapter 2).

In the examples shown here, triangle meshes with different initial refinement levels are used (as described in Sec. 1.4.3), but these meshes are not refined yet. A discussion of suitable refinement methods for final level of detail computation is given in Chapter 3. Since rendering performance largely depends on these mesh refinement methods, results with performance measurements are given in that section, and are omitted here.

Fig. 1.16 shows a sequence of images zooming in on the University of Siegen, Germany. Data sets in this scene are the DEM from the Nasa Blue Marble Next Generation (BMNG), the BMNG image for April 2004, and a higher resolution aerial photograph of the city area around the university.

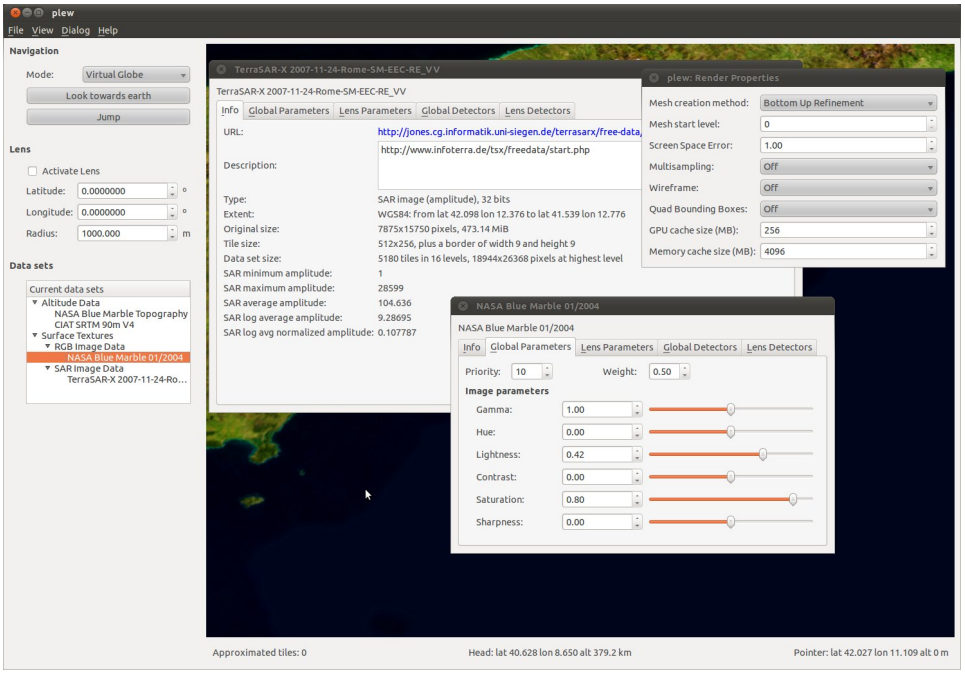

Figure 1.18: A screenshot of the framework, showing a selection of user interfaces.

The wireframe version of the images shows the restricted quadtree structure built by the level of detail technique, and the initial triangle mesh generated from it.

Fig. 1.17 shows an example scene of the area around Rome, Italy. Active data sets in this scene are:

1. The DEM from the NASA BMNG data set. This DEM provides world-wide coverage at a relatively low resolution: 86400×43200 data samples, each 16 bits wide, for the complete WGS84 map. This results in an original raw data size of 6.95 GiB.

2. A DEM based on data from the Shuttle Radar Topography Mission (SRTM), but post-processed using additional data sources to fill holes and eliminate inconsistencies [JRNG08]. This data set has a significantly higher resolution, but covers only part of the earth surface. It provides 432000×144000 data samples, each 16 bits wide, for the latitude range from $+60°$ to $-60°$ on the WGS84 map. This results in 115.87 GiB of raw data.

3. The NASA BMNG image for January 2004. This data set provides 86400×43200 color image pixels (RGB, combined 24 bits) for the complete WGS84 map, resulting in 10.43 GiB of raw data.

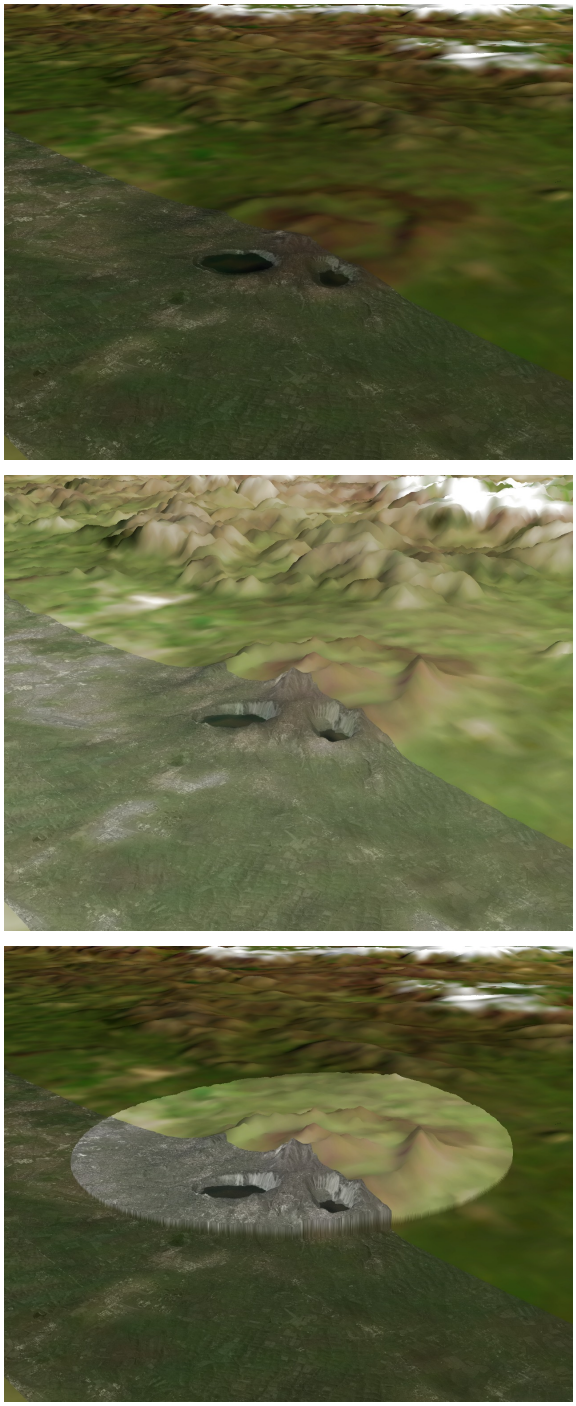

Figure 1.19: Demonstration of a lens, applied to the example scene shown in Fig. 1.17. The top and middle views show the scene with different processing and fusion parameters. The bottom view shows both parameter sets combined using a lens.

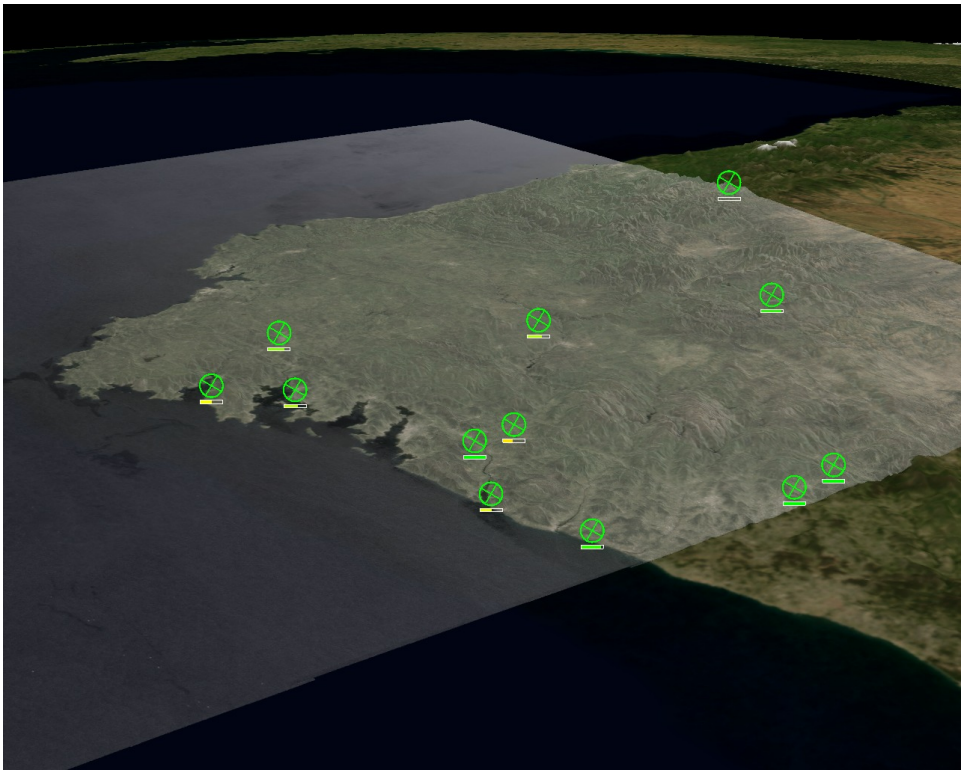

Figure 1.20: A detector for near-ideal reflectors in SAR images, displaying visual hints for each detected reflector.

4. A SAR amplitude image taken by the TerraSAR-X satellite and provided by Infoterra GmbH. This image covers only a limited area around Rome, and provides 7875 $\times$ 15750 data samples, each 16 bits wide, resulting in 236.57 MiB of raw data.

In this example view, the two DEMs are scaled with different factors and then combined into a single elevation map set using a simple weighted fusion filter. The BMNG image and the SAR image are processed and then combined into a single texture set using a weighted blending filter. All processing and fusion parameters are interactively adjustable, as shown in Fig. 1.18.

The compressed quadtree hierarchies for the four data sets use a combined 123.5 GiB of storage space. This includes overhead for the multiresolution representation, the 9 sample border around each data quad, and global and per-quad metadata. Building these hierarchical representations of the original sensor data requires substantial amounts of time, but only needs to be done once per data set. Previous static terrain rendering methods that build hierarchical representations of fixed texture and elevation

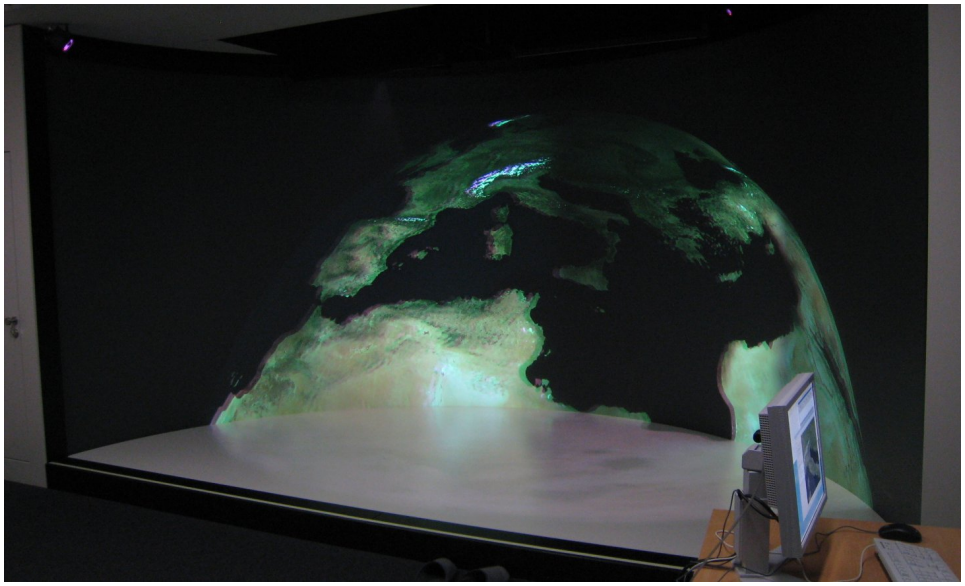

Figure 1.21: The framework running in a Virtual Reality laboratory, with distributed rendering across six render nodes with a total of 12 graphics cards.

map sets have more opportunities for optimization in the preprocessing step, e.g. by applying lossy compression techniques to lower the demand for storage space.

Fig. 1.19 demonstrates the use of a lens. The data sets and scene are identical to Fig. 1.17, but the processing and fusion parameters differ. The lens allows direct comparison of two parameter sets, thus allowing a better understanding of the effects of parameters and of data set properties.

Fig. 1.20 shows an example of a detector for near-ideal reflectors in SAR amplitude images. This detector is described in detail in Sec. 2.4.

Fig. 1.21 demonstrates distributed rendering. It shows the framework running in the Virtual Reality laboratory of the University of Siegen. This setup uses six cluster nodes for rendering, each equipped with two graphics cards for passive stereo (left and right view), and an additional master node. The framework also includes support for dynamic warping via a post-processing fragment shader pass, which is required for user tracking in this installation due to the curved screen [KLT*09].

Chapter 2

Synthetic Aperture Radar Image Visualization

2.1 Overview

Synthetic Aperture Radar (SAR) is an active imaging sensor technology. A target scene is illuminated using electromagnetic waves, typically in one of the microwave frequency bands X (3 cm), C (6 cm), or L (24 cm). A receiver collects the signals reflected from the target area. In Remote Sensing applications, both transmitter and receiver are typically spaceborne or airborne. Monostatic SAR systems use the same platform for transmitter and receiver. Recently, experiments with bistatic SAR systems were conducted, where different platforms are used for transmitter and receiver, e.g. a satellite and an airplane [WEB*10].

The two dimensions of a SAR image are given by the azimuth and range directions. The azimuth (or along-track) direction is defined by the motion of the receiver relative to the target. The range (or cross-track) direction is usually perpendicular to the azimuth direction.

The image resolution in range direction is determined by the bandwidth of the transmitted signal. To achieve a high resolution in azimuth direction, SAR systems use a synthetic aperture by combining multiple recordings of a physical antenna that moves across the target area. This requires fully phase-coherent signal collection and processing, and exact knowledge of the transmitter and receiver positions relative to the target for the full duration of the recording [Mor00].

Unlike passive optical imaging systems, SAR systems are independent of daylight conditions, since they actively illuminate the target area, and they are also independent of weather conditions, since microwaves can penetrate clouds. At the same time, current SAR systems achieve high spa-

Figure 2.1: A SAR amplitude image of an area in Peru, taken by the TerraSAR-X satellite on 2008-03-12 and provided by Infoterra GmbH.

tial resolutions (less than 1 m resolution cell diameter for spaceborne systems, and less than 10 cm for airborne systems). Additionally, microwaves can partially penetrate canopy, soil, and snow, and the characteristics of SAR images allow analysis of features that cannot be observed through other sensor systems. These properties make SAR images indispensable for many Remote Sensing tasks.

A SAR image is produced from the collected reflected signals in the SAR image formation process. A variety of image formation methods exist, with focus on different optimization strategies and different transmitter/receiver configurations [Bam92, LNPK04]. Due to various constraints regarding the available processing power, storage space, and bandwidth, most SAR systems store the collected raw signals onboard and transmit them asynchronously to a ground station where the image formation process takes place.

Originally, one sample of a SAR image contains both amplitude and phase information. The phase information can be used in interferometry applications. Additionally, depending on the SAR system, polarization information can be exploited. The focus of this dissertation is the amplitude information only; interferometry and other applications are not

considered. Therefore, the term SAR image in this dissertation refers to a SAR amplitude image, where each sample contains an amplitude value. Furthermore, it is assumed that radiometric correction and georeferencing were already applied.

A SAR image typically uses a 16 bit integer or single precision floating point data type to store amplitude values (see Sec. 2.2). SAR amplitude images are usually displayed by mapping these amplitude values to gray levels. See Fig. 2.1 for an example.

The interpretation of SAR images is a complex task that depends on the characteristics of both the SAR system and the features under investigation [OQ04]. Nevertheless, two common challenges can be identified:

- the high dynamic range of SAR amplitude data, and
- the speckled nature of SAR images.

In this chapter, techniques are presented that address both aspects in an interactively adjustable way, based on the data processing stage of the framework as described in Sec. 1.3. Additionally, a detector for near-ideal reflectors in SAR images is presented that is usable as a visual assistance tool for SAR image analysis as discussed in Sec. 1.5.

2.2 Dynamic Range

2.2.1 Introduction

The dynamic range of a SAR image is described by the ratio of its largest amplitude value and its smallest non-zero amplitude value. This dynamic range depends on the characteristics of the SAR system and the target scene. For example, newer SAR systems with higher ground resolutions tend to produce images with increased dynamic range due to pulse compression effects [Mor00]. Furthermore, corner reflectors in the target scene, commonly used for calibration and georeferencing purposes, generate especially high amplitude values in their characteristic response patterns. See Sec. 2.4. Data types commonly used for accurate representations of SAR images include 16 bit integer types and the single precision floating point type.

The dynamic range of a SAR image is typically much higher than the dynamic range of common display devices such as LCD monitors or printed media. Common gray level images used for display and printing use the 8 bit integer range $\{0,\ldots,255\}$ to represent the gray levels. Currently, only special purpose display systems can use a wider 10-bit integer range [NVI09a]. In either case, mapping SAR amplitude values

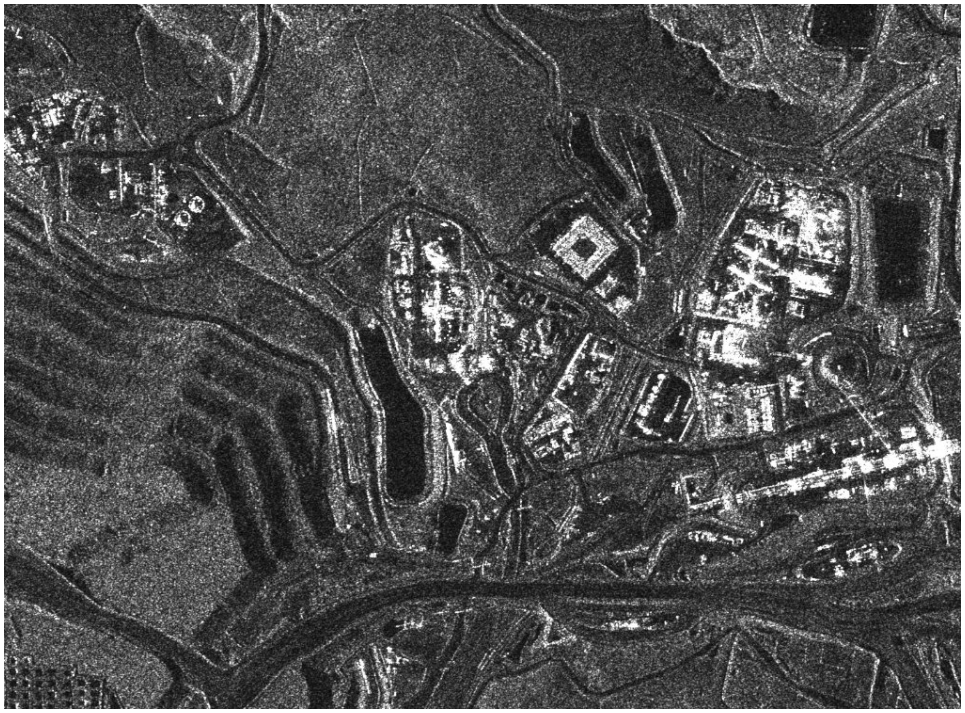

Figure 2.2: The SAR amplitude image from Fig. 2.1. To produce this image, the lowest 10 % of the amplitude range was linearly mapped to gray levels. The highest 90 % of the amplitude range was clipped to white, and all details in this range are lost.

to gray levels for display results in information loss: low amplitude values are clipped to black, high amplitude values are clipped to white, and many intermediate steps are lost because they cannot be represented. For example, a simple linear mapping of the amplitude values to gray levels delivers unusable results, mostly because a large range of amplitude values is clipped. See Fig. 2.2.

A dynamic range reduction method for SAR images maps the amplitude values a to gray levels $g \in [0,1]$ using a mapping function f. The gray level range $[0,1]$ is then quantized to a range of integers suitable for the output device (usually the 8 bit range $\{0,\ldots,255\}$).

The mapping function of *global* dynamic range reduction methods depends only on the current sample $a(x,y)$:

$$g(x,y) = \mathrm{f}(a(x,y)) \tag{2.1}$$

For *local* methods, f depends on a all samples in a neighborhood $\mathcal{N}(x,y)$ of the current sample $a(x,y)$:

$$g(x,y) = \mathrm{f}(\{a(x',y') | (x',y') \in \mathcal{N}(x,y)\}) \tag{2.2}$$

In the remainder of this section, it is assumed that the amplitude values have been normalized to $[0, 1]$ using the maximum amplitude value $a_{\max}$ stored in the image.

The goal of the mapping function f is to minimize the information loss that occurs in the quantization step, and at the same time maximize contrast in interesting amplitude ranges and image areas. Since the interesting amplitude ranges and image areas will vary between different data sets and different visualization tasks, the dynamic range reduction methods must be adjustable. To enable quick exploration of image features in different amplitude ranges and optimization of the mapping for the features that are relevant to the current visualization task, it must be possible to perform this adjustment interactively.

2.2.2 Related Work

There is very little published work on dynamic range reduction specifically for SAR images. The mapping of SAR amplitude values to gray levels in different software packages is largely undocumented in the literature. Often, ad-hoc methods are used that give visually pleasing results for most data sets. For example, the RAT radar tools [RH04] first apply gamma mapping to the amplitude values, then clip values that are larger than a threshold, and then map the results linearly to $[0, 1]$. There are many similar possibilities, e.g. mapping an interval $[a, b]$ to $[0, 1]$ using a logarithmic function after estimating the bounds a and b from the amplitude data.

All of the known ad-hoc methods are global methods; local dynamic range reduction methods for SAR images have not been described before. Local methods can enhance local contrast and thereby improve the visibility of details, but resulting gray levels in different image areas are not directly comparable anymore.

While there is very little previous work on dynamic range reduction for SAR images, much research has been carried out in a related field from the computer graphics domain: tone mapping. Tone mapping is used to reproduce high dynamic range optical images on low dynamic range display devices.

For this purpose, tone mapping operators (TMOs) reduce the dynamic range of the luminance in optical images. Global TMOs use an identical mapping function for all image pixels. In contrast, local TMOs map pixels depending on their neighborhood: bright pixels in dark areas are treated differently than bright pixels in bright areas. This gives local TMOs more flexibility both for reduction of the dynamic ranges and for preservation of local image details. Global techniques are the natural choice when processing speed is critical, but GPUs also allow fast implementations of local

methods, e.g. for video applications [KMS05].

A number of global and local TMOs have been proposed in recent years. An introduction and extensive overview was given by Reinhard et al. in 2005 [RWPD05]. The focus in this section is on TMOs that are relevant for the purpose of interactive visualization of SAR amplitude images, i.e. TMOs that

- are applicable to SAR amplitude values, and
- are suitable for a GPU-based, interactively adjustable implementation.

This excludes TMOs that are based on sophisticated models of color perception, such as the iCAM model proposed by Fairchild and Johnson [FJ04] and the multiscale model proposed by Pattanaik et al. [PFFG98]. Additionally, some TMOs aim to reproduce certain perceptual effects of the human visual system, e.g. night vision and glare effects [KMS05]. Such methods are not applicable to SAR amplitude values, and are therefore not considered here.

The majority of the remaining TMOs work on single channel data: they first extract the luminance information from the image, then reduce its dynamic range, and re-add the color information afterwards. By omitting the color handling, such TMOs are applicable to SAR amplitude values (and other types of data) in a straightforward way. Of course, the quality of the results varies depending on the models on which the individual method is based.

Although the purpose is similar, there are several important differences between tone mapping for optical images and dynamic range reduction for SAR images:

- Tone mapping approximates the original appearance of high dynamic range optical images on low dynamic range displays. Therefore, quality measurement techniques for TMOs can be based on comparisons of the reproduced result with the original scene, or with the original scene as reproduced by a special high dynamic range display device [YMMS06, LCTS05]. Such quality measurements can also use a model of the human visual system to account for different dynamic ranges in the reproductions [AMMS08]. For SAR images, no original appearance exists, and similar quality measurements for dynamic range reduction methods are not possible.
- The distribution of amplitude values in SAR data sets differs from the distribution of luminance in optical images [OQ04]. For example, corner reflectors and other strong scatterers cause small peaks of

high amplitude in areas of significantly lower amplitude in the SAR image. This is very uncommon in optical images and may cause problems for TMOs. Additionally, local average values in optical images are often estimated based on assumptions that do not hold for the speckled nature of SAR images.

- The goal of TMOs is often to produce results that look natural to the human observer. The reproduction of details is not of high importance for this goal [ČWNA06], but it is essential for dynamic range reduction for SAR images.

A selection of global and local TMOs and their application to SAR images is described in Sec. 2.2.4.

2.2.3 Commonly Used Methods

The methods described in this section are those that are reported to be commonly used in the context of SAR image display. All of them are variants of linear, logarithmic, or gamma mapping.

Note that all commonly used methods are global dynamic range reduction methods. No local dynamic range reduction methods for SAR images are documented.

Linear Mapping

Linear mapping from amplitude values to gray levels is only usable if a relatively small range of the amplitude values is selected and all values outside of this range are clipped to black or white respectively. Otherwise, most details are lost in very dark regions of the resulting image. A possible mapping function is the following, where a_{avg} is the arithmetic mean of the amplitude values and a_σ is the standard deviation:

$$g(x,y) = \frac{a(x,y)}{t}, \quad t = a_{\text{avg}} + 3a_\sigma \tag{2.3}$$

Due to clipping of the result to $[0,1]$, all information in amplitude values greater than t is lost when using this method.

Logarithmic Mapping

Logarithmic mapping is often used with SAR data, since most of the image detail is concentrated in low amplitude ranges (see Fig. 2.2). Logarithmic

mapping methods are usually adjustable with one parameter c that controls the overall brightness of the result. The following mapping function can be used:

$$g(x,y) = \frac{\log(1 + c \cdot a(x,y))}{\log(1+c)}, \quad c \geq 1 \tag{2.4}$$

To make better use of the available gray level range, often only a part of the amplitude range, from $t_{\min}$ to $t_{\max}$, is mapped to gray levels in this way. Values outside of this range are clipped to black or white respectively. The values $t_{\min}$ and $t_{\max}$ can be estimated from the SAR images. Interactive systems allow the user to change these values on the fly.

Gamma Mapping

This is an ad-hoc dynamic range reduction method based on gamma mapping, with properties similar to the logarithmic mapping described above:

$$g(x,y) = a(x,y)^{\gamma}, \quad \gamma \in (0,1) \tag{2.5}$$

The RAT radar tools [RH04] use the following variant: First, gamma mapping with $\gamma = 0.7$ is applied to the amplitude values. Then the mean value m of the results is computed. After that, the interval $[0, 2.5 \cdot m]$ is linearly mapped to $[0,1]$. Values larger than $2.5 \cdot m$ are clipped to white.

2.2.4 Tone Mapping Operators

This section describes a selection of global and local tone mapping operators and their application to SAR images.

In the tone mapping situation, the input samples for the dynamic range reduction method usually provide absolute luminance values measured in cd/m^2. In some situations, absolute luminance values are not known and the input image only provides relative values from $[0,1]$ without an explicit scale. Furthermore, some TMOs do not require absolute luminance values as input samples, and instead just assume a certain range of input values.

As stated above, it is assumed in this section that SAR amplitude values are normalized to $[0,1]$. Where necessary, this interval can be transformed for input into a dynamic range reduction method using a prescaling factor p, which becomes an additional parameter if the method does not have prescaling built in.

Drago Logarithmic Mapping

Drago et al. propose a global logarithmic mapping method that adapts the base of the logarithm depending on the current value [DMAC03].

$$g(x,y) = \frac{m}{\log(1+c)} \cdot \frac{\log(1+c \cdot a(x,y))}{\log(2+8 \cdot a(x,y)^b)}, \quad c \geq 1, m \geq 1, b \in [0,1] \quad (2.6)$$

The parameter c is analogous to the simple logarithmic mapping. The parameter m determines the maximum brightness of the result, and the parameter b steers the amount of contrast.

Tumblin Brightness Preserving Operator

Tumblin and Turk describe a global TMO that is based on psychophysical experiments measuring the interrelation of luminance and perceived brightness [TT99]. The aim is to preserve the perceived brightness in the resulting image. The method uses two adaptation parameters, a_d and a_a, for display luminances and amplitude values respectively. Additionally, a prescaling parameter c is used.

$$g(x,y) = a_d \left(\frac{c \cdot a(x,y)}{a_a} \right)^{\frac{\gamma(a_a)}{\gamma(a_d)}}, \quad c \geq 1, a_d \geq 1, a_a \geq 1 \quad (2.7)$$

The function γ is a logarithmic function that models the human contrast sensitivity, derived from measured data.

Schlick Uniform Rational Quantization

Schlick proposes a general global quantization method that is applicable to single channel data such as normalized SAR amplitude values in a straightforward way [Sch95]. The mapping function is not based on assumptions about the human visual system:

$$g(x,y) = \frac{b \cdot a(x,y)}{(b-1) \cdot a(x,y) + 1}, \quad b \in [1,\infty) \quad (2.8)$$

It uses a single parameter b which controls the overall brightness of the result.

Reinhard/Devlin Photoreceptor Model

Reinhard and Devlin describe a TMO that is motivated by photoreceptor behaviour and is designed to work on the RGB channels of optical

images [RD05]. Using sophisticated models for the computation of a photoreceptor adaptation level, effects such as light adaptation and chromatic adaptation can be simulated. If the method is applied to single channel data, only the light adaptation term l remains.

$$l(x,y) = (1-c)\cdot a(x,y) + c\cdot a_{\text{avg}}, \quad c \in [0,1] \tag{2.9}$$

$$m = 0.3 + 0.7\cdot\left(\frac{1-a_{\text{avg}}}{1-a_{\text{min}}}\right)^{1.4} \tag{2.10}$$

$$g(x,y) = \frac{a(x,y)}{a(x,y) + (\exp(-b)\cdot l(x,y))^m}, \quad b \in [-8,8] \tag{2.11}$$

This method requires knowledge of the minimum and average amplitude values in the image, a_{min} and a_{avg}.

The parameter c in the light adaptation term controls the contrast in the resulting image. The parameter b determines the brightness.

Chiu Spatially Variant Operator

Chiu et al. proposed one of the first local methods to be used for tone mapping [CHS*93]. The mapping function is as follows:

$$g(x,y) = \frac{a(x,y)}{c\cdot a_g(x,y)}, \quad c > 0 \tag{2.12}$$

Here, a_g is a Gauss-filtered version of a. Therefore, $g(x,y)$ depends on the neighborhood $\mathcal{N}$ as described by Eq. 2.2. The parameter c is a constant of proportionality between the two values.

Rahman Retinex

Like the Chiu method, the method proposed by Rahman, Jobson and Woodell is a local method that uses Gauss filtering to reduce the dynamic range [RJW96]. Multiple filtered versions of the input data set are produced using different Gauss filter sizes. A set of weights determines the influence of each filtered data set on the result. The weights can be computed from a single user parameter w. Additionally, the method uses a prescaling parameter.

Ashikhmin Spatially Variant Operator

Ashikhmin describes a TMO that is designed to preserve contrasts [Ash02]. It is a local TMO that uses multiple Gauss filtered versions of the input data set to determine a local adaptation value A for each pixel. The largest

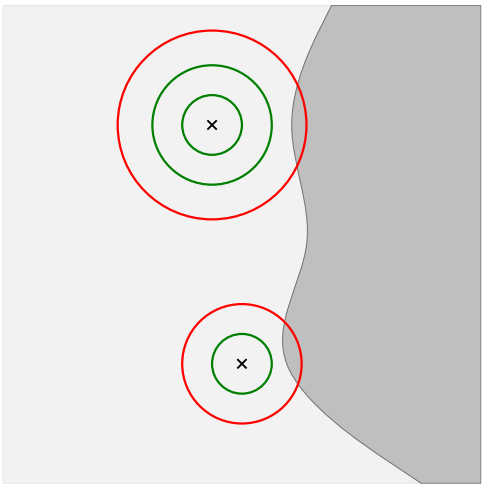

Figure 2.3: The transition between the bright image area on the left and the dark image area on the right causes high gradient values at the border between both areas. For the positions marked with a cross, the largest Gauss filter that does not cross this border is chosen (green). A Gauss filter that crosses the border would cause blurring of image features and is rejected (red).

Gauss filter that does not cross high gradients is used, as shown in Fig. 2.3. This avoids blurring of image features. The mapping function is then defined as

$$g(x,y) = F(c \cdot A(x,y)) \frac{c \cdot a(x,y)}{c \cdot A(x,y)}, \quad c > 0 \tag{2.13}$$

The parameter c is a prescaling factor. An additional parameter $t > 0$ is used to determine the minimum strength of gradients that the Gauss filters are not allowed to cross. The function F is designed using perceptual motivations.

Reinhard Photographic Tone Reproduction

Reinhard et al. propose a tone reproduction operator that uses ideas from photography, namely scene keys and dodging-and-burning [RSSF02]. It is a local TMO and uses the following mapping function:

$$g(x,y) = \frac{c \cdot a(x,y)}{1 + c \cdot A(x,y)}, \quad c > 0 \tag{2.14}$$

Analogous to the Ashikhmin TMO, A is a Gauss filtered version of the amplitude values a, using the biggest filter size that does not cross strong gradients. Reinhard et al. propose some refinements to this equation to

make better use of the available gray level range. Parameters of the resulting mapping function are the prescaling factor c and a parameter w that influences the smallest amplitude that is mapped to white. Additionally, the computation of A requires a sharpness-influencing parameter s and a gradient threshold t.

Durand Bilateral Filtering

The TMO proposed by Durand and Dorsey splits the input image into a base layer and a detail layer. Dynamic range compression is applied only to the base layer, and the details are re-added afterwards [DD02]. The base layer is produced by applying a bilateral filter to the original data set. The bilateral filter is a smoothing filter that avoids to smooth across boundaries, i.e. high amplitude differences. The detail layer is the difference between the original and the filtered data set.

2.2.5 Local Methods for SAR Images

The mapping function of global dynamic range reduction methods is usually monotonically increasing. Brighter pixels in the result displayed are known to be caused by larger amplitude values. This is not necessarily true for local methods, because local methods can treat the same amplitude value differently depending on its neighborhood. The comparability of resulting gray levels is lost, but local contrast can be enhanced to improve the visibility of details. Interactive visualization systems allow to use global and local methods simultaneously, thus combining the benefits of both. A lens is ideally suited for this purpose (see Sec. 1.5.2).

The local TMOs described in the previous section all compute a local adaptation value from a neighborhood and use this value to determine how to treat the current sample. Such local adaptation values are variants of local averages. Most local TMOs use Gauss filtering to determine the local adaptation value.

Simple Gauss filtering as used by early methods such as the Chiu and Rahman methods leads to large halo artefacts (contrast reversals) at the boundary of bright image features [RWPD05]. Therefore, more recent methods use various strategies to avoid computing averages across borders between image regions of different brightness, such as bilateral filtering (Durand method) or locally varying Gauss filter sizes (Ashikhmin method, Reinhard photographic method). This reduces undesired halos.

However, small halos in largely homogeneous regions can emphasize details that might otherwise be lost. This property of local methods is desirable in the context of dynamic range reduction for SAR images.

The main problem of adapting local methods to dynamic range reduction of SAR images is to find a suitable method to compute local adaptation values. Because of the different properties of SAR images and optical images, methods developed for optical images give unsatisfactory results (see also Sec. 2.2.7).

In this section, new adaptive methods for dynamic range reduction of SAR images are proposed. The methods are based on existing tone mapping techniques, but the computation of local adaptation values is tailored to the properties of SAR images.

Base Methods

Dynamic range reduction methods intended for interactive use should not require tweaking of too many parameters, and the parameters that are required should have an intuitive and predictable effect on the image.

Of the global TMOs described in Sec. 2.2.4, the Schlick quantization method and the Reinhard/Devlin photoreceptor model both fulfill this requirement. The Schlick method has a single parameter b that steers the brightness of the result (see Eq. 2.8), and the Reinhard/Devlin method uses two parameters b and c that control brightness and contrast of the result (see Eq. 2.11). The local methods, on the other hand, usually require more parameters and/or parameters whose impact on the result is difficult to understand.

Both Schlick and Reinhard/Devlin describe extensions to their global methods to make them adaptive to local neighborhoods. Since these extensions do not significantly improve the tone mapping results for optical images, both methods are mostly used in their global variant [RWPD05]. However, the extended methods deliver good results for SAR images when used with a suitable method to compute local adaptation values.

The Schlick quantization method is extended by replacing the current amplitude value $a(x,y)$ in the denominator of Eq. 2.8 with a replacement value $a'(x,y)$ [Sch95]:

$$g(x,y) = \frac{b \cdot a(x,y)}{(b-1) \cdot a'(x,y) + 1}, \quad b \in [1,\infty) \tag{2.15}$$

Similarly, the Reinhard/Devlin method is extended by replacing the term $a(x,y)$ in the computation of the light adaptation value in Eq. 2.9 with $a'(x,y)$ [RD05]:

$$l(x,y) = (1-c) \cdot a'(x,y) + c \cdot a_{\text{avg}}, \quad c \in [0,1] \tag{2.16}$$

The replacement value $a'(x,y)$ can be computed using a linear interpolation between the current amplitude value $a(x,y)$ and the local adaptation

value $A(x,y)$:

$$a'(x,y) = (1-d)\cdot a(x,y) + d\cdot A(x,y), \quad d \in [0,1] \tag{2.17}$$

This allows to steer the amount of adaptivity of each method using the parameter d. For $d = 0$, the extended method is equivalent to the global method, and for $d = 1$ the adaptivity is set to its maximum.

The remaining problem is to find a suitable method to compute the local adaptation values $A(x,y)$ for SAR images.

Adaptation Values for SAR Images

The choice of a method to compute adaptation values depends on the image features one wants to emphasize. In regions where no features should be emphasized, the adaptivity value $A(x,y)$ should not differ much from the current amplitude value $a(x,y)$, so that the method behaves similar to its global counterpart. Around image features that should be emphasized, $A(x,y)$ should be the average of a local region, and therefore usually different from $a(x,y)$. The method can then adapt to the properties of local neighborhoods, thereby emphasizing the feature of interest.

In SAR images, major features of interest are peaks in the data that have a considerably greater amplitude than the surrounding region. The following method to compute a local adaptation value $A(x,y)$ accounts for this:

- Compute local averages G_r around the current pixel (x,y) using Gauss filters of increasing radius r, from 1 sample to 10 samples. Choosing the standard deviation $\sigma = r/2.5$ ensures that the filter mask covers almost all of the filter mass.
- Compute quotients $v_{r+1} = \frac{|G_r - G_{r+1}|}{G_r}, \quad 1 \le r \le 9$.
- Choose the largest G_r such that $v_r > t$ for a user specified threshold t. If no v_r is greater than t, use $G_1(x,y) = a(x,y)$.

In largely homogeneous regions, this method will choose $A(x,y) = G_r(x,y)$ for small values of r or even $r = 1$, which means that the dynamic range reduction will behave like its global counterpart. Around peaks however, the local adaptation value will be the Gauss weighted average of a larger region. This results in an emphasized peak in the gray level image.

Note that this is the opposite of what the Ashikhmin Spatially Variant TMO [Ash02] does: in Ashikhmin's method, the largest G_r such that $v_r < t$ is chosen. This results in using large radii in homogeneous regions and smaller ones when boundaries between bright and dark image

regions would be crossed otherwise. Image details in largely homogeneous regions are emphasized, and halos around borders between image regions of different brightness are avoided. These properties are suitable for optical images, but not for SAR images.

2.2.6 Implementation

The dynamic range reduction methods described above were implemented using the GPU-based processing chain of the visualization framework, described in Sec. 1.3.

The global methods do not require intermediate steps: the input image can be mapped directly to the output image in a single step, and the mapping function can be implemented in a GLSL shader or CUDA kernel in a straightforward way.

Most local methods compute local adaptation values, and often this computation can be done by a separable filter. In this case, a multi-step implementation can be used. For example, to apply multiple Gauss filters at the same time for the Ashikhmin operator or the local SAR methods, a first shader can compute the horizontal filter components and store them in different array components of an intermediate result. If OpenGL textures are used, up to four components can be stored in this way; with OpenCL or CUDA, this limitation does not exist. A second shader step can then compute the vertical filter component and store the result in the output buffer.

2.2.7 Results

The implementation as part of the visualization framework allows to interactively explore the properties of each method when applied to different SAR images. Having a variety of dynamic range reduction methods is only useful if the user can interactively adjust method and parameters to both the current SAR image and the analysis aims. In particular, the user must be able to interactively switch between global methods that allow a direct comparison of different image regions, and local methods that show more details in specific regions. The lens concept of the visualization framework allows to use global and local methods simultaneously, as shown in Fig. 2.4. This allows users to benefit from the more detailed results of local methods while still allowing the direct comparison of different image regions permitted by global methods.

Fig. 2.5 shows results of selected methods for an example image produced with the AER-II sender and PAMIR receiver [BE06]. The raw data was processed at ZESS, University of Siegen, to produce the SAR image.

Figure 2.4: Global and local dynamic range reduction methods applied simultaneously to a SAR image using a lens.

The methods were also applied to different SAR images from the ERS-1/2, Envisat, and TerraSAR-X spaceborne platforms.

Global Methods

The result of linear mapping (top left in Fig. 2.5) clearly shows that high amplitude ranges are clipped to white, and therefore details are lost. The logarithmic mapping (top right in Fig. 2.5) and the similar gamma mapping avoid this problem to some extent. The result of gamma mapping lacks contrast when compared to other results. The Drago logarithmic mapping method can produce better results than the simple logarithmic mapping due to its adaptability and additional parameters, but this requires some fine-tuning.

The Tumblin-Rushmeier method is adjustable to a wide range of input data via its parameters, but it tends to clip higher amplitude ranges to white. The psychophysical model that motivates the method is likely not suitable for SAR data.

The Reinhard-Devlin method (middle left in Fig. 2.5) and the Schlick

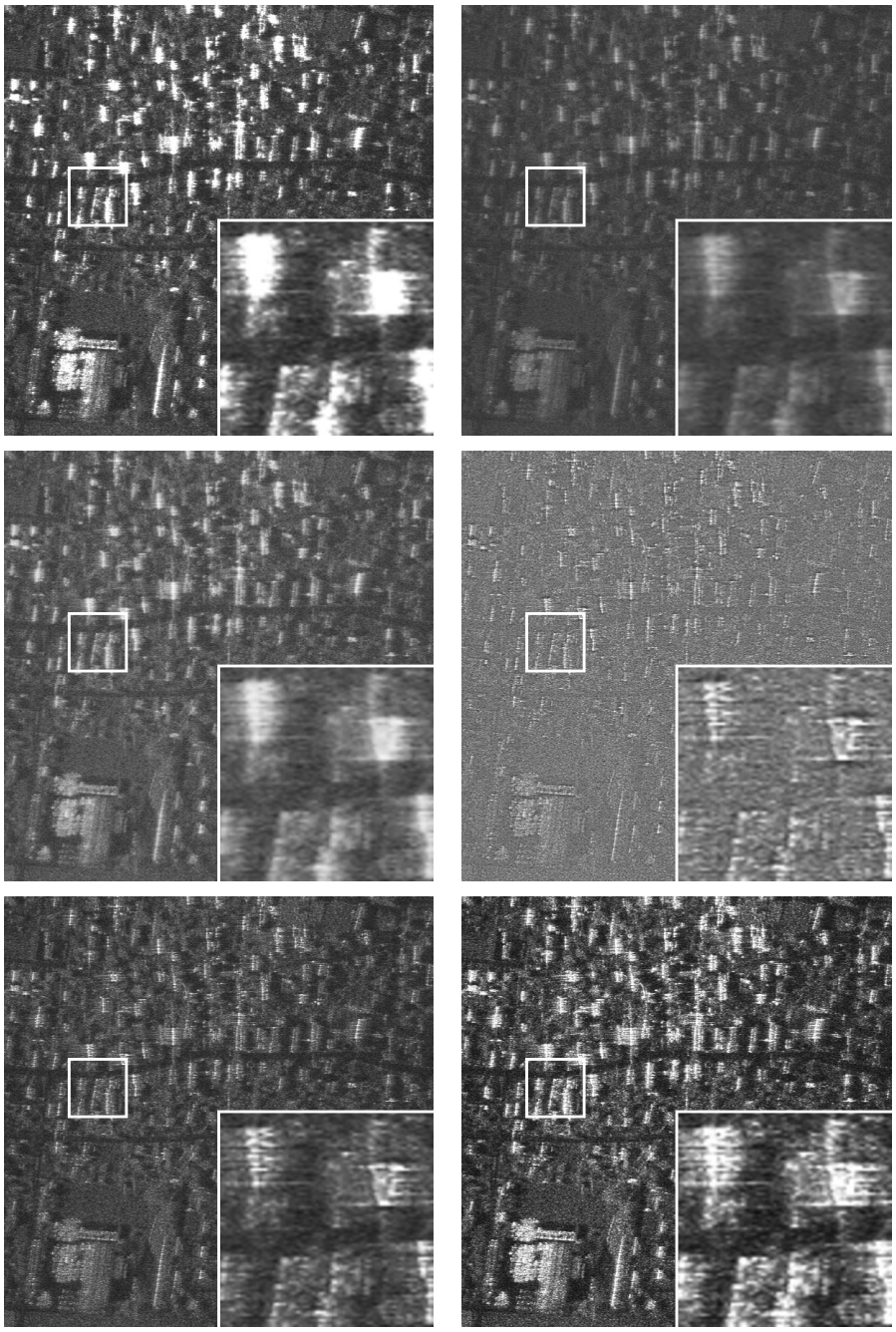

Figure 2.5: Results from a selection of global and local dynamic range reduction methods. Top row: linear and logarithmic methods, middle row: Reinhard/Devlin and Rahman methods, bottom row: Ashikhmin and Reinhard Photographic method. The bottom right part of each image shows a closeup of the area framed by the white rectangle.

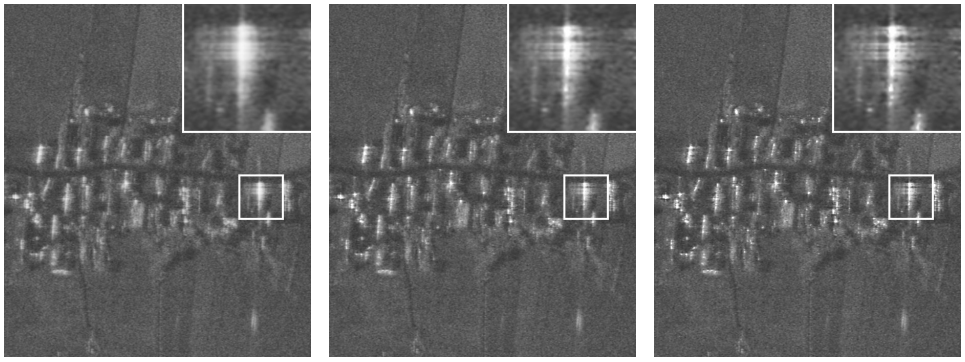

Figure 2.6: Detail of a SAR amplitude image, visualized using the extended Schlick method with parameter $d = 0.0, 0.5, 1.0$.

method both produce high contrast results without clipping high amplitude values. The brightness parameter of the Schlick method and the brightness/contrast parameter pair of the Reinhard-Devlin method are easily adjustable. Both methods are valuable alternatives to the commonly used simple logarithmic mapping method.

Local Tone Mapping Operators

The Rahman method (middle right in Fig. 2.5) emphasizes high frequencies corresponding to very fine local structures even when used with relatively strong Gauss filtering. The similar Chiu method exhibits the same problem to an even greater extent. With these methods, it is difficult to obtain usable results for SAR images because of their speckled nature, although for the Rahman method the influence of speckle can be reduced somewhat by tweaking the parameters.

The Ashikhmin method (bottom left in Fig. 2.5) works well for SAR data even though it is based on perceptual motivations. The parameter t influences the amount of detail that is emphasized. By setting t appropriately, the results are much less affected by speckle than the results of the Chiu and Rahman methods.

The Reinhard Photographic method (bottom right in Fig. 2.5), like the Tumblin-Rushmeier method, tends to clip higher amplitude ranges to white when applied to SAR data. Additionally, for the test images, it is hard to adjust the four parameters to produce reasonable results. The Durand method is slightly easier to adjust, but like the Tumblin-Rushmeier and Reinhard Photographic methods it tends to clip higher amplitude ranges to white. For all three methods, this effect may be caused by the photo-centric nature of the underlying assumptions.

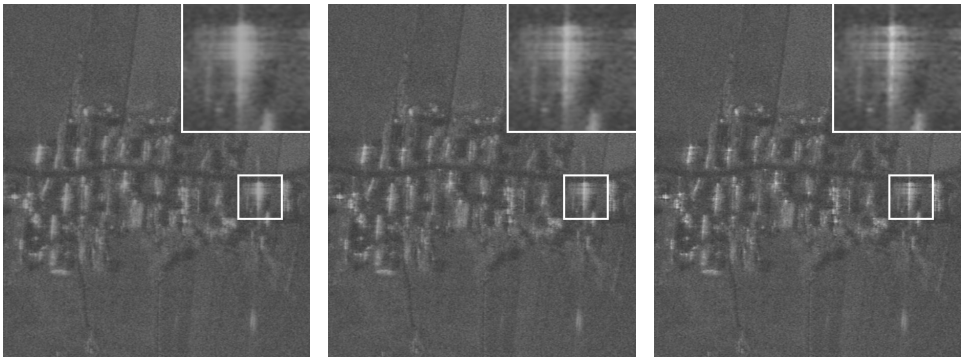

Figure 2.7: Detail of a SAR amplitude image, visualized using the extended Reinhard/Devlin method with parameter $d = 0.0, 0.5, 1.0$.

Fig. 2.5 shows that local methods can produce results with increased local contrast, and significantly increase the preservation of local details. This helps to analyze image details. Nevertheless, the results should still be improved by using local adaptation values suitable for SAR images.

Proposed Local Methods

The local extensions of the global Schlick and Reinhard-Devlin methods combine the advantages of easy-to-adjust, intuitive parameters of the global methods with the increased local contrast offered by adaptive methods. The local variants use adaptation value computations suitable for SAR images, and introduce the additional parameters d, to steer the amount of adaptivity, and the threshold value t.

Fig. 2.6 shows results of the local Schlick method. The brightness parameter b was set to 75 in this example. The parameter d was set to 0.0, 0.5, and 1.0 respectively. For $d = 0.0$, the extended method is equivalent to the global method.

Fig. 2.7 shows results for the Reinhard/Devlin method. Similarly to Fig. 2.6, the parameter d was set to 0.0, 0.5, and 1.0 respectively. The brightness parameter b was set to -1.5 and the contrast parameter c to 0.9. Setting the contrast parameter to 1.0 would eliminate the influence of the local adaptivity value (see Eq. 2.16). Therefore, c should not be higher than 0.95.

Setting the threshold t for the computation of the local adaptation values too low will result in emphasized details in all image regions, including emphasized speckle; this is not desirable. Setting the threshold too high will disable all emphasizing effects because the local adaptation value will always be the same as the current amplitude value. Experi-

ments have shown that threshold values between 0.05 and 0.1 work well for different SAR images from different sensors. The images in Fig. 2.6 and Fig. 2.7 were produced using a threshold of $t = 0.05$.

The method to compute local adaptation values for SAR images proposed in Sec. 2.2.5 can also be used with other methods, such as the Durand Bilateral Filter TMO or the Ashikhmin Spatially Variant TMO. However, the Schlick and Reinhard/Devlin methods are preferable because of their intuitive parameters and the quality of their results.

2.3 Speckle

2.3.1 Introduction

A SAR image resolution cell contains the combined reflected signals of the objects located in the target area covered by this cell. Since the backscattered signals of all objects inside the resolution cell are coherently summed, constructive or destructive interference can lead to very high or very low values. This results in the characteristic speckled nature of SAR images [MJ04]. See Fig. 2.1.

It is important to note that this speckle is the recorded signal and not noise. Nevertheless, speckle can hinder both human visual interpretation of SAR images as well as computer-based image analysis, segmentation, and classification techniques. For these reasons, a reduction of the speckle is a central step in many SAR image analysis applications.

One way to reduce speckle in SAR images is using multi-look techniques which combine several measurements for each resolution cell during the image formation process. This reduces the speckle at the cost of reduced spatial resolution. Multi-look techniques are applied at the image formation stage and are therefore part of the SAR system. Such techniques are not discussed in this section; the focus here is on filtering techniques for standard single-look SAR images.

In the context of the interactive visualization framework presented in Chapter 1, the goal of this section is not to propose new speckle filter methods, but rather to show the feasibility of interactively adjustable, GPU-based implementations of the various existing filtering methods.

2.3.2 Related Work

A wide variety of filters for speckle reduction in SAR images exist. These filters have different strengths and weaknesses for different types of input images. Therefore, having a variety of methods to choose and adapt to the current task is important for interactive visualization of SAR images.

Various comparative studies give details about the existing filter methods and their properties [LJD*94, GJ97, Tou02]. Touzi presents a classification of filter methods that is based on the different speckle models used by the filters [Tou02]. In contrast, this section uses a classification that is based on the processing techniques used, as this is the main property of interest with regard to GPU-based implementations of the filters. Typical representatives for each class are mentioned in parentheses:

- Convolution filters (Mean filter, Gauss filter). These filters apply different convolution kernels to the SAR image data. Most of the common convolution kernels are separable, and allow to split the filtering process into two steps to reduce the total number of operations.

- Filters based on local statistics (Lee filter, Kuan filter, Frost filter, Gamma maximum a posteriori probability). These filters compute statistics from a local neighborhood, and produce an output value based on a statistical model of SAR speckle.

- Rank Operators (Median filter). These filters apply some form of sorting to the input values given by the mask size.

- Wavelet based filters (Soft thresholding). These filters perform filtering on wavelet coefficients.

The following section discusses the details of GPU-based implementations of filters from each of these classes, using the GPU-based processing chain presented in Sec. 1.3.

2.3.3 Interactive Speckle Reduction

The interactive visualization framework presented in Chapter 1 uses a hierarchical, quad-based representation of the SAR image, to allow interactive use of processing methods, including speckle reduction for SAR images. Therefore, all speckle reduction methods must work on this hierarchical, quad-based representation. This has two consequences:

1. Speckle reduction filters can directly access only local information. Global information requires special handling.

2. Filter mask sizes must be adapted to the hierarchy level, so that the filters have a constant mask size in world space.

Convolution filters. Simple convolution filters like the Mean and Gauss filters are easily implementable in the framework. A 3×3 Mean filter, for example, can be implemented using one processing step. The GPU microprogram gathers the information from the 9 samples of the local neighborhood and computes the filtered output sample. Mean and Gauss filters with larger masks can be implemented in a separable manner to increase performance by reducing the total number of operations. In this case, each filter needs two processing steps: one for the horizontal mask, and one for the vertical mask. The masks can usually be stored in uniform variables (OpenGL) or shared memory (CUDA), but for special filters it may be beneficial to use 1D textures or arrays, which allows fast interpolated access to mask elements [Nov05].

Filters based on local statistics. Such methods interpret a local neighborhood $\mathcal{N}$ (e.g. 5×5 or 7×7) as a sampling distribution, and compute sample mean m and sample variance s^2 of the amplitude values $a_i \in \mathcal{N}$:

$$m = \frac{1}{|\mathcal{N}|} \sum_{a_i \in \mathcal{N}} a_i \tag{2.18}$$

$$s^2 = \frac{1}{|\mathcal{N}| - 1} \sum_{a_i \in \mathcal{N}} (a_i - m)^2 \tag{2.19}$$

$$= \frac{1}{|\mathcal{N}| - 1} \left(\sum_{a_i \in \mathcal{N}} a_i^2 - \frac{1}{|\mathcal{N}|} \left(\sum_{a_i \in \mathcal{N}} a_i \right)^2 \right) \tag{2.20}$$

The value of the output pixel is then computed based on the underlying assumptions about statistical properties of SAR images. For rectangular neighborhoods, the sums $\sum_{a_i \in \mathcal{N}} a_i$ and $\sum_{a_i \in \mathcal{N}} a_i^2$ used in Eq. 2.18 and Eq. 2.20 can be computed simultaneously in a separable manner by storing the results of the first processing step in an intermediate texture or array with two components. In the second processing step, after computing m and s^2 from these sums, the filter result is computed and stored in the output texture or array.

Some methods based on local statistics need additional values that are computed globally from the SAR image. For example, the method proposed by Xiao, Li, and Moody [XLM03] needs global minimum and maximum deviation values. These cannot be computed efficiently on the GPU due to the quad-based data management: only local information is available. Generally, there are three methods to solve this kind of problem:

1. Compute the values in the preprocessing step, while building the data hierarchy.

2. Estimate the values based on a local neighborhood. This can be done on the GPU.

3. Treat the values as additional parameters to the method and let the user adjust them.

The first method is the method of choice in order to achieve results that are comparable to the original algorithm. See Sec. 1.2.3. The third method also works reasonably well, but at the cost of additional parameter adjustment required by the user, which is disadvantageous in interactive visualization systems. The second method will lead to unintended differences in filter behaviour in different regions of the image.

Ranking operators. Ranking operators like the Median filter are based on sorting the values in the local neighborhood, and computing their output based on the order of the sorted values. Usually, sorting on the GPU is performed with algorithms that fit the GPU programming model, such as merge sort variants [KW05] or, more recently, algorithms based on parallel scan techniques [SHG09]. However, for the relatively small local neighborhoods used in typical filters, the overhead of applying such techniques outweighs their benefits.

Instead, a small number of values can be sorted directly within an OpenGL fragment shader or CUDA kernel using simple algorithms such as bubble sort. If the number of values (i.e. the mask size) is known at compile time, the GPU compiler can produce relatively efficient code using complete loop unrolling, thus reducing branching.

The maximum size of neighborhoods is limited with this approach, because the maximum number of GPU instructions allowed in one unit can easily be exceeded, depending on the hardware limits. A solution to this problem is to use a separable approximation of the median filter [Nar81]. That way, the number of values in each of the two processing passes is kept small, and reasonably large filter masks can be used.

Wavelet based filters. Filters based on wavelet transforms allow frequency-related manipulations of the SAR image. A suitable wavelet transform is applied to the image, and a filtering method, e.g. a variant of soft thresholding [Don95], is applied to the wavelet coefficients. The reverse wavelet transformation then yields the filtered SAR image. Due to the limited local support of wavelets, it is possible to perform the wavelet transform on the quad-based image representation used in the framework. The discrete wavelet transform can be computed efficiently on the GPU [WLHW07].

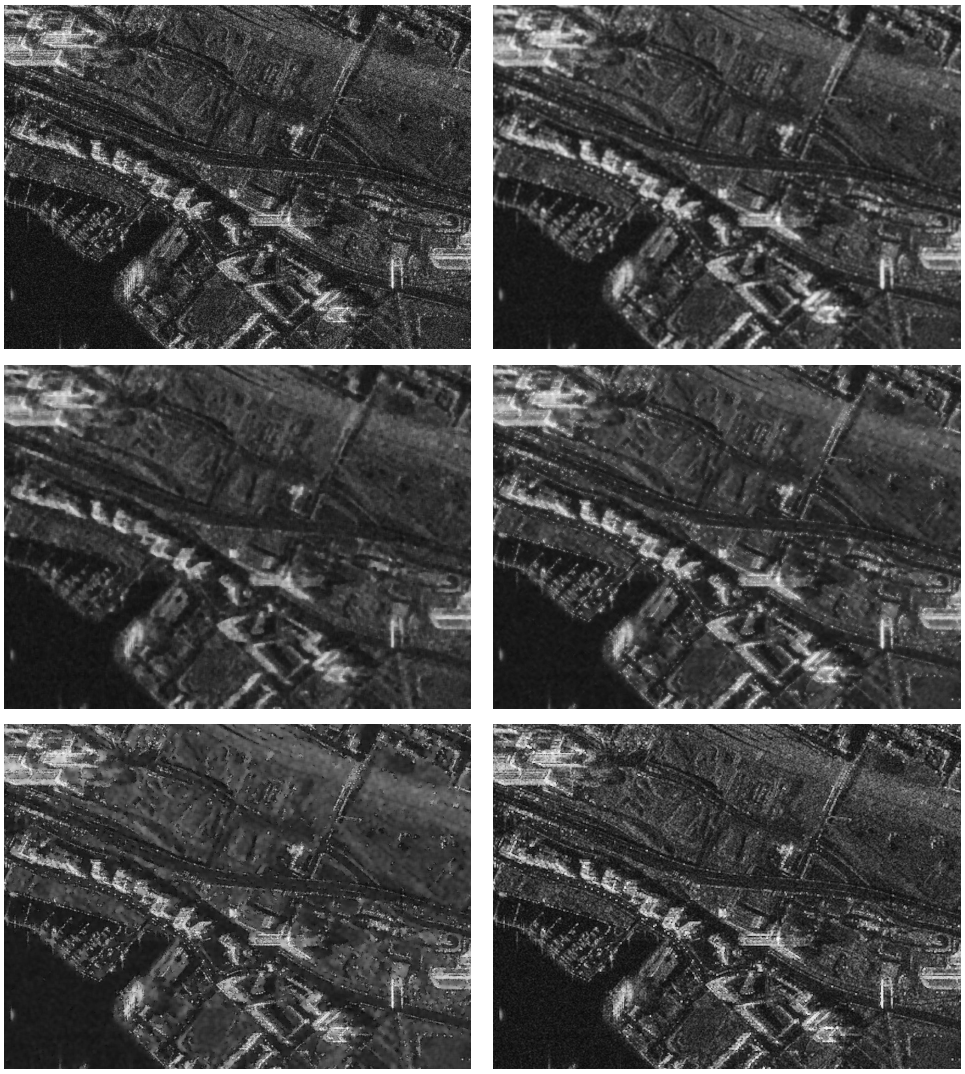

Figure 2.8: Results from a selection of despeckling filters, applied on a TerraSAR-X SAR image of a rural area provided by Infoterra GmbH. In reading order: original image, Gauss filter, approximative Median filter, Lee filter, Gamma MAP filter, and Wavelet Soft Thresholding.

2.3.4 Results

Fig. 2.8 shows example despeckling results for a TerraSAR-X SAR image. The despeckling filters can be interactively adjusted for the SAR image and the visualization task. Typically, filter masks are not larger than 9×9 or 11×11 for most filter methods. Since efficient separable implementations of the filter methods are possible, processing time is not a problem for interactive use.

2.4 Point Target Analysis

2.4.1 Introduction

A single perfect reflector for the signals sent by the SAR transmitter would result in a characteristic pattern, known as an ideal point target response, in the SAR image [Mor00]. See Fig. 2.9. The central lobe is the main lobe. Side lobes are arranged along two axes that cross at the main lobe.

This property of SAR images is commonly used for quality assessment purposes [SBA*99]. A corner reflector placed into the target scene reflects the radar signal back in the direction of the SAR receiver and serves as an approximation of the theoretical perfect reflector. By comparing the SAR image pattern caused by the corner reflector with the ideal point target response pattern, the quality of the SAR image can be estimated.

Another important use of the point target response is that it can be used to mark fixed locations in a scene. A corner reflector placed in a precisely known, fixed location is known as a persistent scatterer. A persistent scatterer will cause the typical point target response pattern in each SAR image taken of the target scene. The point target response pattern can therefore be used for precise registration of SAR images [FPR01]. This is a prerequisite for important applications in interferometric SAR as well as for time-dependent analysis of SAR image series.

In the absence of corner reflectors in the target scene, e.g. when the terrain is inaccessible or a costly preparation of the target scene is unfeasible, strong natural scatterers available in the scene can be used as a substitute.

Due to the importance of SAR point target analysis, an interactive visualization framework should provide tools to assist the user in this task. In this section, a detector for point target response patterns in SAR images is presented, based on the detector concept described in Sec. 1.5.3. The detector automatically gathers information about each pattern in the currently viewed scene, allowing the application to provide the user with a visual overview of potentially interesting patterns as well as their key properties. Furthermore, the user can select single patterns for a more

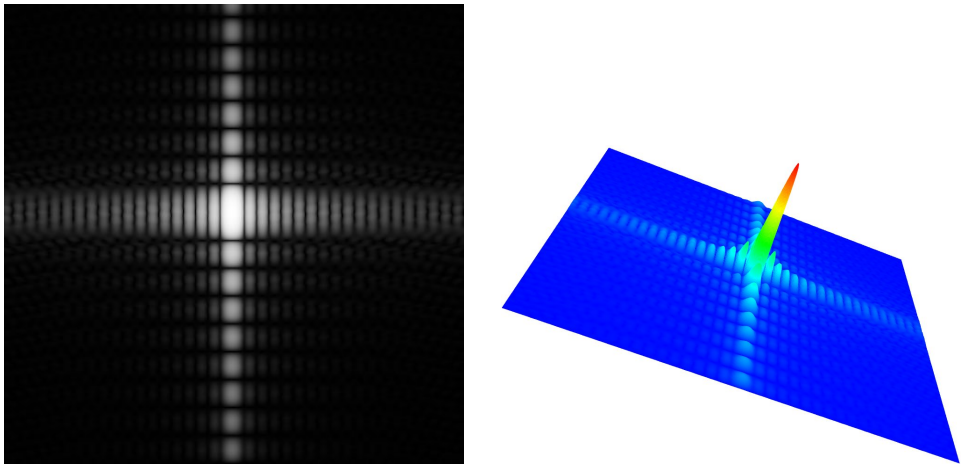

Figure 2.9: A simulated SAR point target response pattern

detailed analysis.

2.4.2 Related Work

A detector for point target response patterns typically searches for the characteristic crossing axes in the amplitude values. Even if an application is only interested in point target response patterns from known corner reflectors, automatic detection of such patterns is still beneficial to cope with uncertainties and inaccuracies [GZY08].

In a georeferenced SAR image, the angle between the two axes of a point target pattern depends on the setup of the SAR system that took the image, and is usually identical for all such patterns in the image.

Quality measurements associated with a point target response pattern include the estimated width of the main lobe, as a measurement for ground resolution, and the peak side lobe ratio (PSLR), which measures the ratio of the peak side lobe amplitude to the peak main lobe amplitude [SBA*99]. As a visual analysis aide, point target analysis tools usually include cross-sections through the two side lobe axes of the pattern [RH04].

2.4.3 Interactive Point Target Analysis

As outlined in Sec. 1.5.3, the detector works on a hierarchical quad-based SAR image representation. The SAR image quads that are used to form the current view of the scene are searched for typical patterns on the GPU. For interactive use, detection speed is more important than accuracy; an

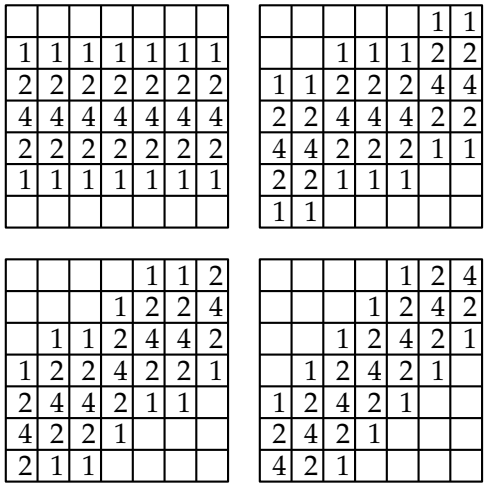

Figure 2.10: The detection masks for 0°, 15°, 30°, and 45°. The masks for the directions 60°, . . . , 165° are analogous.

interactive detector cannot provide more than a rough overview of the interesting features in a scene.

For these reasons, a very basic detector setup is used as follows. Every SAR image sample in an input quad is tested to determine whether it is the center of a point target response pattern. If it is, rough estimates of basic properties of the pattern are computed, and the results are written to an output map of the same dimensions as the input quad. Stream reduction techniques [OLG*07] are then used to transfer this map to a list of detected point target patterns and their basic properties. In an OpenGL implementation, this task can be performed using a geometry shader [Dia08]. The list of detected patterns is then used to provide visual hints about the detected patterns.

In detail, the actions performed for each SAR image sample are the following:

1. If the amplitude value is lower than 15% of the maximum amplitude value in the SAR image, then it is considered too low. A zero is written to the output map, and no further tests are performed.

2. If the amplitude value is not the maximum value in its 7×7 neighborhood, then it cannot be the center of a point target. A zero is written to the output map, and no further tests are performed.

3. Mean values are computed using 12 masks for the directions 0°, . . . , 165° as shown in Fig. 2.10. The two highest mean values whose directions differ by at least 45° are chosen. If one of these two values

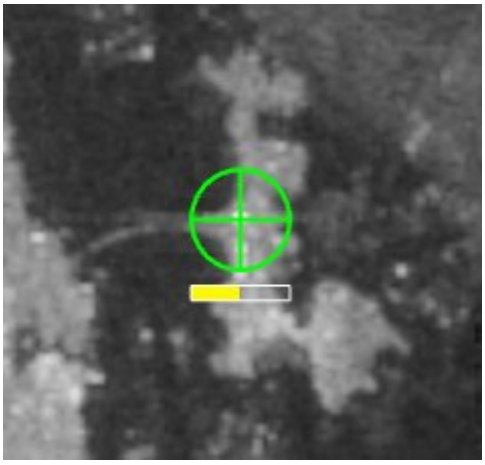

Figure 2.11: A visual hint for a detected point target response pattern.

is lower than a threshold t multiplied with the arithmetic mean of all amplitude values in the 7×7 neighborhood, then the pixel is not considered to be a point target center, and a zero is written to the output map. Otherwise, the two directions and a rough estimate of the PSLR are encoded in a non-zero value that is written to the output map.

The masks shown in Fig. 2.10 are generic enough to work for point target patterns of different sizes, which is necessary because the pattern size depends on the strength of the reflected signal: stronger reflections will result in larger patterns. As a consequence, the masks also work for patterns of a given size across different hierarchy levels.

If the side lobe direction of response patterns is known for the SAR image under examination, the computational costs of the test can be reduced significantly by only using the relevant masks. Detection accuracy will improve with better SAR image resolution. At hierarchy levels that provide only a very coarse resolution of the SAR image, point target response patterns become undetectable.

The threshold parameter t steers the sensitivity of the detection process. Higher values result in less detected point targets.

In the detector implemented in the framework, a visual hint for a detected pattern consists of a circle enclosing a cross that gives a hint about the detected side lobe axes orientation. A color coded bar below the circle represents a quality estimate derived from the coarse PSLR estimation: a full green bar indicates a high PSLR, and a nearly empty red bar indicates a low PSLR. See Fig. 2.11. An overview of detected patterns in a scene can contain multiple of these hints. See Fig. 2.12.

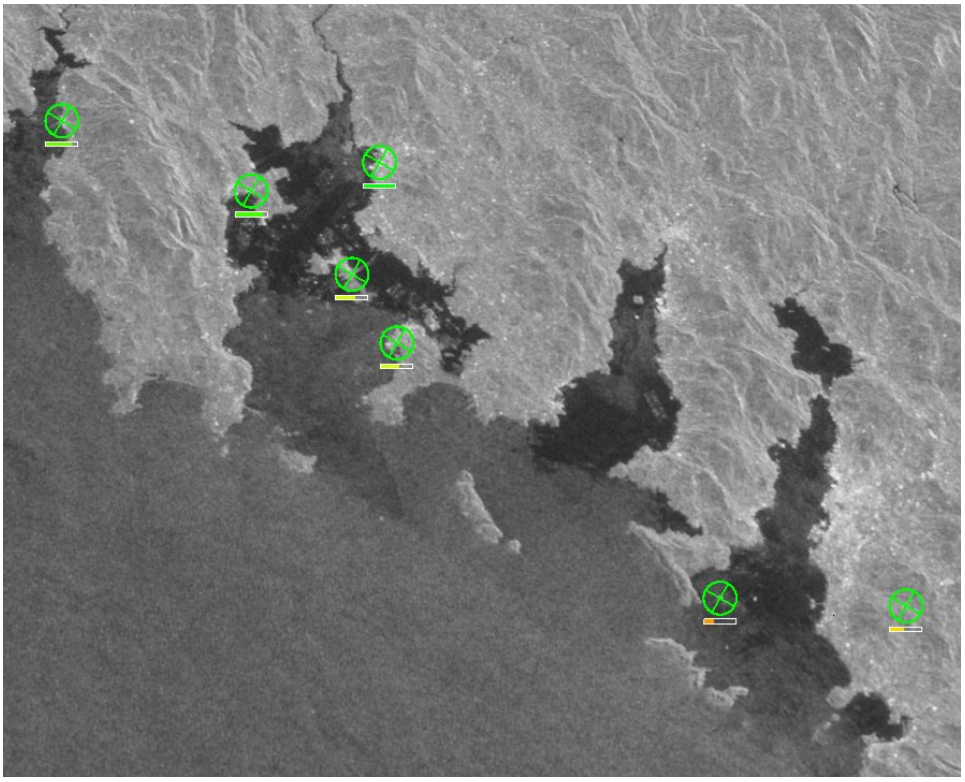

Figure 2.12: An overview of detected patterns in an Envisat ASAR image.

When a user requests a more detailed analysis, much more accurate and detailed examinations of the pattern are possible. An example for an analysis view is given in Fig. 2.13. This view presents cross sections through the two axes of the response patterns and lists several key properties of the pattern.

2.4.4 Results

The presented point target response pattern detector is capable of interactively detecting patterns in the currently visualized scene. Because of the generic masks used in the detector, this works across several hierarchy levels, as long as point target responses are represented with sufficient detail. A switch between hierarchy levels, e.g. when zooming in or out, will occasionally cause previously detected patterns to be undetected, or previously undetected patterns to be detected.

As predicted in Sec. 1.5.3, the detector is computationally very expensive, even if it is simplified to provide only very coarse and approximate detection. Therefore, the detector can only be used sparingly; detecting

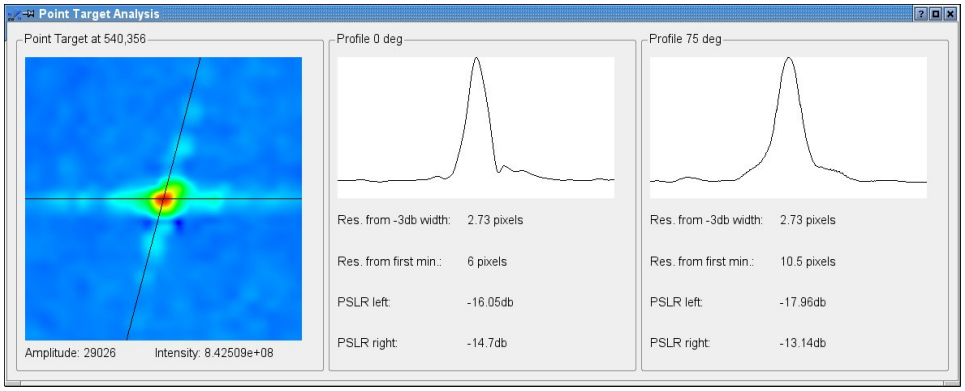

Figure 2.13: Example of a point target response analysis window, displayed on user demand.

patterns in multiple stacked SAR images at once or in very high resolution views is currently not possible.

Despite this limitation, the presented detector can give a useful overview of a scene and intuitive access to advanced analysis tools, which is useful in the context of interactive visualization.

2.5 Summary

In this chapter, interactive visualization methods tailored to SAR images were presented. SAR images are very important for many remote sensing applications, and at the same time pose special challenges to a visualization system, mainly due to their high dynamic range and speckled nature.

To address the dynamic range problem, existing dynamic range reduction operators were implemented in the GPU-based framework to allow interactive use, and new operators tailored to SAR images were proposed, inspired by related tone mapping methods.

It was shown that the main classes of known speckle reduction techniques can be implemented in an interactive visualization system based on GPU data processing. This allows to easily adjust methods and parameters to the current SAR image and visualization task.

Additionally, an interactive detector assists with the task of SAR point target response analysis.

Taken together, the presented techniques provide powerful and flexible tools for the interactive visualization of SAR images.

Chapter 3

Dynamic Terrain Rendering

3.1 Overview

Terrain rendering is the process of rendering a part of a surface that is described by distances to a reference geometry and accompanying texture data. Usually, the surface is a part of the earth's surface, and the geometry is given either by height maps (relative to a local reference plane) or by elevation maps (relative to a model of the earth, e.g. a sphere or ellipsoid).

Terrain rendering is an important problem in the field of computer graphics and has been intensively investigated in the last decade. A key challenge is the large amount of data typically contained in a terrain data set. Handling such data sets requires hierarchical data representations and level of detail techniques.

Traditionally, terrain rendering methods have assumed static input data. This allows sophisticated offline preprocessing to be applied before the online rendering takes place. This has been used extensively to improve the performance of the rendering.

In the context of interactive visualization of Remote Sensing data, terrain rendering methods need to address two additional challenges:

1. The terrain data is generated dynamically, depending on various input data sets as well as interactively adjusted data processing and fusion methods. See also Fig. 1.1. This means that existing methods that rely on offline preprocessing cannot be used.

2. The visualization of Remote Sensing data for analysis purposes requires accurate rendering, to avoid misinterpretations. Therefore, the required level of detail techniques must guarantee upper bounds on rendering inaccuracies. This is in contrast to other applications of terrain rendering, e.g. in the field of entertainment, where the ren-

dering result needs to be visually pleasing but not necessarily correct with regard to the input data.

In this chapter, data structures and algorithms are presented that allow accurate, GPU-based refinement of triangle meshes for the purpose of rendering dynamic terrain data.

To this end, a data structure called *edge mark array* is proposed that allows GPU-based bottom up and top down refinement of triangle meshes based on hierarchical elevation map (or height map) structures. The refinement methods require no supplemental information beside the elevation map data, and therefore can be applied to dynamic terrain data sets. They work entirely on the GPU, avoiding any transfer of geometry data between CPU and GPU. The resulting meshes consist only of right-angled and isosceles triangles and are free of cracks and T-junctions. They can be used for continuous level of detail rendering of dynamic terrain data, and guarantee an upper bound on the screen space error.

The methods fit into the framework described in Chapter 1 in the following way (see Fig. 3.1):

- A first, conservative level of detail estimate is computed on the CPU as described in Sec. 1.4.2. According to this level of detail, data from the hierarchical input data sets are transferred to the GPU.

- The data is processed and fused on the GPU, forming an elevation map set and a texture set.

- Based on the restricted quadtree that serves as the initial level of detail estimate, a single conforming triangle mesh is created by the framework. See Sec. 1.4.3.

- This mesh is adaptively refined using the methods described in this chapter. The result is an adaptive mesh based on the dynamically generated elevation map set and the current view. This mesh determines the final level of detail with accuracy guarantees.

- The refined mesh is then rendered using the techniques described in Sec. 1.4.4.

Despite this tight integration into the framework, the methods presented in this chapter are general enough to be applicable to mesh refinement problems in all applications that use some form of quad-based elevation or height map management.

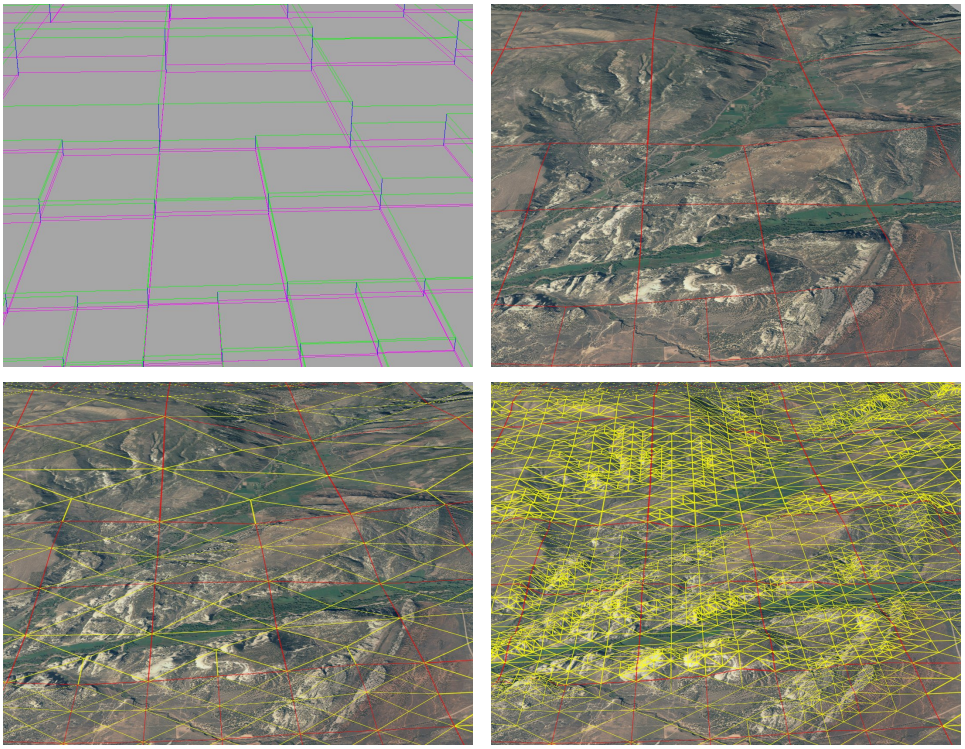

Figure 3.1: Overview of the dynamic terrain rendering process. Top left: coarse level of detail estimation based on bounding boxes. Top right: data processing and fusion. Bottom left: generation of an initial mesh. Bottom right: rendering of the refined mesh in the final level of detail.

3.2 Related Work

In the last decade, many methods to render large-scale terrain data sets have been proposed. This section focusses on the more recent GPU-oriented techniques.

As discussed in Sec. 1.2, the large size of terrain data sets necessitates the use of hierarchical data structures with appropriate level of detail methods. This ensures that the amount of data required to render a given scene depends on the viewport and not on the resolution of the data sets. An overview of such data structures and methods is given by Pajarola and Gobetti [PG07]. In the following, some of the more recent variants are examined in detail.

Many terrain rendering methods compute the triangulation for individual tiles in a separate preprocessing step. For example, Wahl et al. precompute triangulated irregular networks (TINs) with object-space error bounds and use compression techniques to lower the memory and band-

width requirements of the meshes [WMD*04]. An alternative to TINs are regular triangulations, most prominently restricted quadtree triangulations [Paj98]. Regular triangulations generally use more regular data structures than TINs, which can result in more GPU-friendly methods. For example, Schneider and Westermann use progressive transmission of nested, quadtree-based meshes to reduce data transfers between CPU and GPU [SW06]. Livny et al. combine triangular tiles of different, fixed resolution levels [LKES09]. Dick, Schneider, and Westermann precompute adaptive quadtree meshes and use geometry compression schemes that are tailored for GPU-based decoding [DSW09]. Bösch, Goswami, and Pajarola precompute object-space error metrics to choose the level of detail at runtime [BGP09]. Cignoni et al. use patches of triangles that form binary tree hierarchies [CGG*03].

Most methods that use precomputed triangulations guarantee error bounds by choosing the level of detail according to precomputed or implicit object-space error bounds of the tiles. Methods that combine meshes of different tiles must avoid inconsistencies such as cracks, e.g. by adding special stitching geometry [WMD*04, DSW09, LKES09].

All of the methods mentioned above assume static height map data to precompute meshes and thus cannot be used with dynamically generated terrain data.

A few methods explicitly support dynamic terrain data to a certain extent. Losasso and Hoppe use a set of nested regular grids centered around the viewer [LH04]. The grid resolution decreases with increasing distance to the viewer, resulting in an approximately uniform screen-space resolution. Terrain synthesis is possible, but required to be spatially deterministic, and no screen space error bound can be guaranteed for terrain with steep slopes. Dachsbacher and Stamminger use a fixed base mesh that is warped according to an importance map to move geometric detail to the appropriate regions [DS04]. A terrain synthesis process can generate additional detail. This detail is restricted to a given frequency band, depending on the geometry density in the current region, and the authors explicitly state that no guarantees about error bounds can be made. Kooima et al. present a method that allows the combination of different terrain data sets on a planetary scale [KLJ*09] using overlay techniques, but this method is not designed to be precise on small scales ($< 1\,\text{m}$) and does not guarantee error bounds.

While these methods can handle dynamic terrain data to a certain extent, they place severe restrictions on the dynamic terrain synthesis and do not guarantee screen space error bounds.

In addition to the classic terrain rendering approaches summarized

above, a few techniques exist that allow manipulation of terrain for the use in games or simulators. The approaches of He et al. [HCP02] and Bhattacharjee et al. [BPN08] focus on handling changes to a single height map, and do not contain any error guarantees. Atlan and Garland use a wavelet-based multiresolution representation of height maps [AG06], which allows to use a set of interactive editing tools, but does not allow unrestricted height map manipulations.

3.3 Data Structures

3.3.1 Hierarchy

As discussed in Sec. 1.2, the large size of terrain data sets necessitates the use of hierarchical data structures. The presented framework and the methods described in this chapter are based on a restricted quadtree structure, similar to other current terrain rendering methods. This has the following advantages:

1. A coarse, conservative approximation of the level of detail can be computed on the CPU. This allows to handle all data management tasks (hierarchy traversal, initiation of asynchronous data transfers, caching) on the CPU.

2. A restricted quadtree (as given by the initial level of detail method) allows efficient generation of a single consistent triangle mesh on the GPU. See Fig. 1.2 and Fig. 1.3.

3. The quadtree hierarchy and the resulting mesh are regular structures and thus offer parallelization opportunities for efficient, GPU-based processing.

3.3.2 Mesh

A conforming triangulation of a restricted quadtree is a single, consistent triangle mesh that consists only of right-angled, isosceles triangles and contains no cracks and T-junctions, as described in Sec. 1.2.2. A restricted quadtree hierarchy allows simple and efficient creation of such a conforming triangle mesh. See Sec. 1.4.3. Iterative refinement methods such as those presented in the following sections preserve these mesh properties in each refinement step.

To refine a mesh top down, triangles are split into smaller triangles. To keep the properties of the mesh intact (only right-angled, isosceles triangles, no T-junctions), it is necessary to split the hypotenuse of a triangle

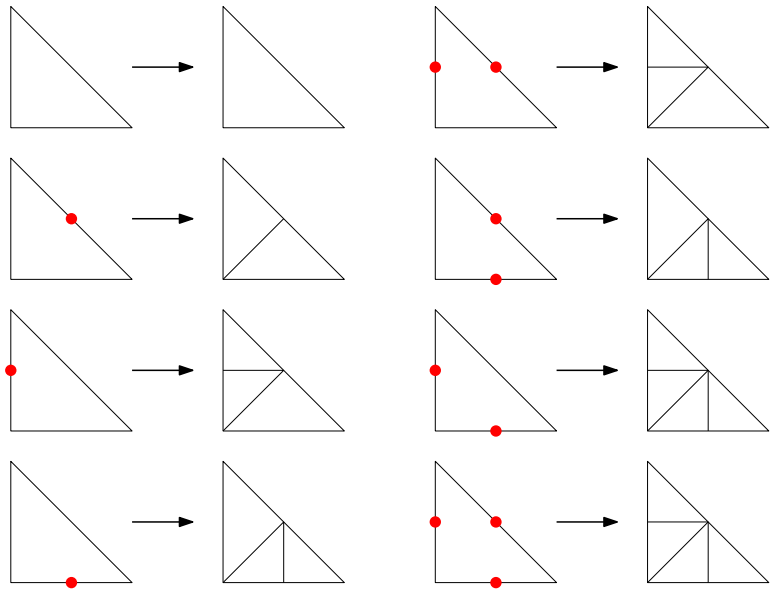

Figure 3.2: Splitting triangles: whenever one edge is split, the hypotenuse is also split, to guarantee right-angled, isosceles triangles.

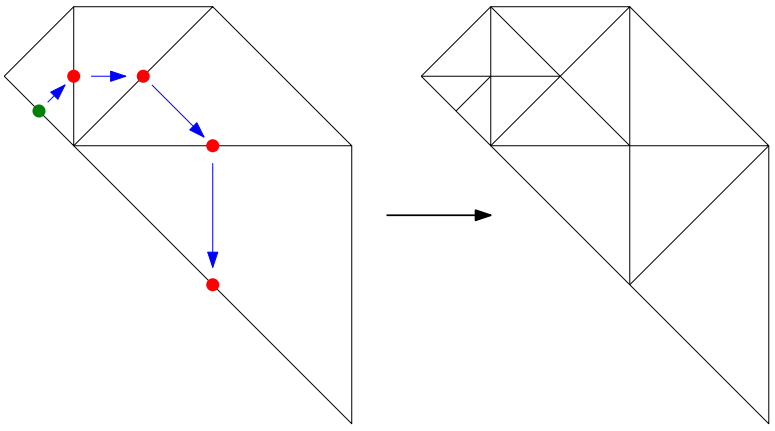

Figure 3.3: Forced splits of the hypotenuse result in propagation of splits to neighboring triangles.

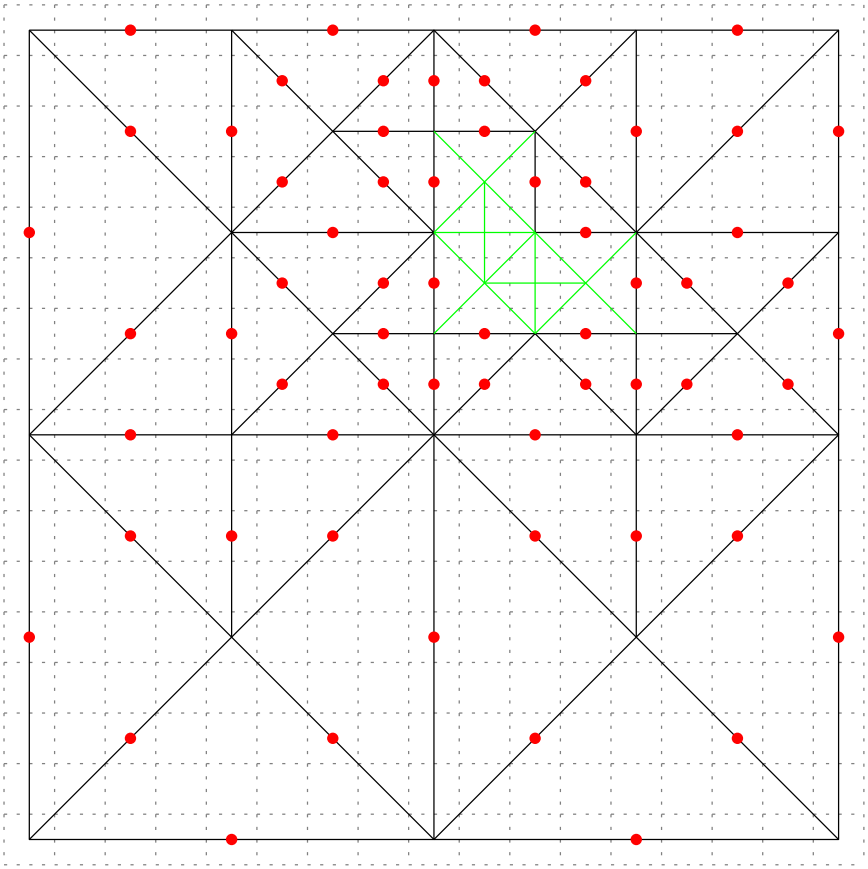

Figure 3.4: An edge mark array of size 17×17 for a quad of size 16×16. Each edge midpoint occupies exactly one edge mark, except for edges of length 1 (marked green), which do not need to be split as they would not reveal further geometric detail.

whenever one of the edges needs splitting [Ulr00]. See Fig. 3.2. This may cause the propagation of edge splits to neighboring triangles, as shown in Fig. 3.3.

Therefore, triangles cannot be refined independently of each other. Furthermore, it is not sufficient to take a restricted neighborhood of each triangle into account, since forced splits in neighbor triangles can themselves force splits in further triangles. This split propagation must also work across neighboring quads of the restricted quadtree. For these reasons, a data structure is required that allows to uniquely identify all edges over all hierarchy levels and the triangles they belong to.

3.3.3 Edge Mark Arrays

An *edge mark array* for one quad with $2^k \times 2^k$ height samples stores $(2^k + 1) \times (2^k + 1)$ edge marks corresponding to all potential edge split points. To cover the whole quadtree hierarchy, one edge mark array for every leaf of the tree is required. Since each edge mark requires only one or two bits to store information (depending on the refinement method; see below), memory consumption is limited, and efficient mesh refinement is possible without explicit geometry data.

An edge mark array uniquely represents all edges with a length greater than 1 of all possible quad triangulations that have the properties described above. Each edge midpoint occupies exactly one edge mark of the array. Edges of length 1 never need to be split and thus do not need to be represented in the edge mark array. See Fig. 3.4.

To find the mark corresponding to a given edge, it is only necessary to compute the midpoint of the edge. Vice versa, a given mark represents the shared hypotenuse of two neighboring triangles in one refinement level. Since each pair of triangles in the refinement hierarchy is defined by its shared hypotenuse, all triangle pairs in the refinement hierarchy can be uniquely identified. See Fig. 3.5.

To find the triangle pair corresponding to a given edge mark $(x, y) \in \{0, \ldots, 2^k\}^2$, it is necessary to first determine the refinement level $l \in \{0, \ldots, 2^{k-1}\}$ in which the mark represents a shared hypotenuse.

With a function lowest_bit(i) that returns the index (starting at 1) of the lowest bit that is set in the integer i, or zero if no bit is set, this can be done as follows:

$$
\begin{aligned}
b_x &= \text{lowest\_bit}(x) \\
b_y &= \text{lowest\_bit}(y) \\
b &= \begin{cases} \max(b_x, b_y) & b_x = 0 \bigvee b_y = 0 \\ \min(b_x, b_y) & \textit{otherwise} \end{cases} \\
l &= \begin{cases} k - 2(b-1) & b_x = b_y \\ 2k - 2b & \textit{otherwise} \end{cases}
\end{aligned}
$$

Note that the first case ($b_x = b_y$) will applies to odd levels, and the second case ($b_x \neq b_y$) applies to even levels.

The L^1 (city block) distance d of edge marks in level l determines the size of the shared hypotenuse:

$$d = 2^{k - \lceil l/2 \rceil}$$

With l and d known, the triangle pair can be computed:

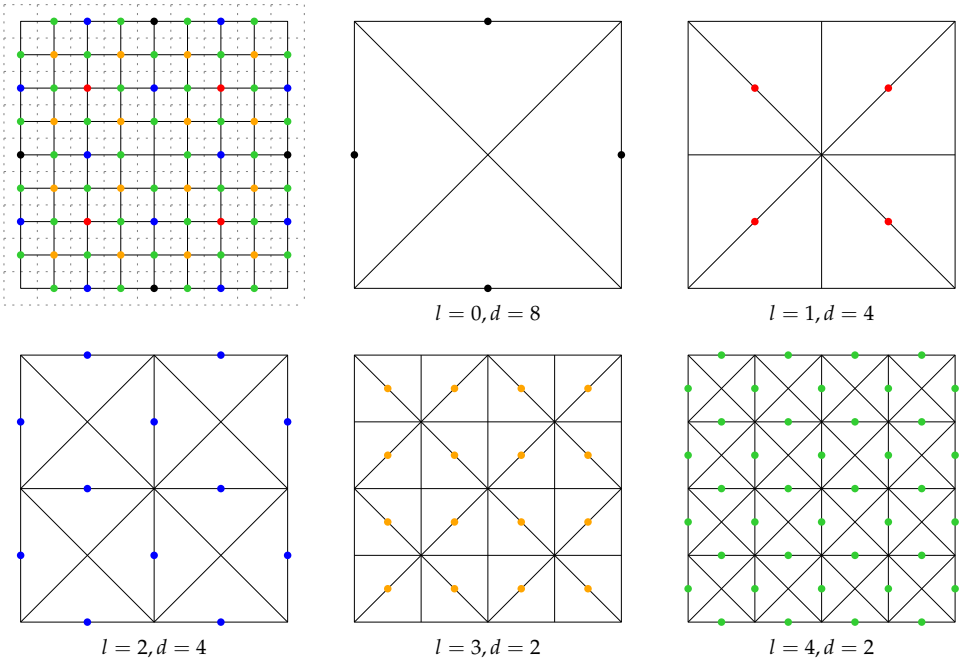

Figure 3.5: An edge mark array and the refinement levels 0 to 4 for a quad of size 8×8, with the associated city block distance d of edge marks. Note that the five edge marks in the corners of the edge mark array and in its center do not correspond to triangle pairs.

- If l is even:
  - If $y \bmod d = 0$, then the triangle pair shares a horizontal hypotenuse that is given by its end points $(x - \frac{d}{2}, y)$ and $(x + \frac{d}{2}, y)$.
  - Otherwise, the triangle pair shares a vertical hypotenuse that is given by its end points $(x, y - \frac{d}{2})$ and $(x, y + \frac{d}{2})$.
- If l is odd:
  - If $\lfloor \frac{x}{d} \rfloor \bmod 2 = \lfloor \frac{y}{d} \rfloor \bmod 2$, then the triangle pair shares a diagonal hypotenuse given by its end points $(x - \frac{d}{2}, y - \frac{d}{2})$ and $(x + \frac{d}{2}, y + \frac{d}{2})$.
  - Otherwise, the triangle pair shares a diagonal hypotenuse given by its end points $(x - \frac{d}{2}, y + \frac{d}{2})$ and $(x + \frac{d}{2}, y - \frac{d}{2})$.

For example, the shared hypotenuse of the triangle pair corresponding to the edge mark $(5, 3)$ in Fig. 3.5 is identified as follows:

$$\begin{aligned} b_x = 1, \quad b_y = 1, \quad b = 1 \\ l = 3 - 2(b - 1) = 3 \quad \text{(odd level)} \end{aligned}$$

$$
\begin{gathered}
d = 2 \\
\lfloor \tfrac{5}{2} \rfloor \bmod 2 = 0, \quad \lfloor \tfrac{3}{2} \rfloor \bmod 2 = 1 \\
\text{End point } 1 : (5-1, 3+1) = (4,4) \\
\text{End point } 2 : (5+1, 3-1) = (6,2)
\end{gathered}
$$

Similarly, for the edge mark $(2,8)$:

$$
\begin{gathered}
b_x = 2, \quad b_y = 4, \quad b = 2 \\
l = 2 \cdot 3 - 2 \cdot 2 = 2 \quad \text{(even level)} \\
d = 4 \\
8 \bmod 4 = 0 \\
\text{End point } 1 : (2-2, 8) = (0,8) \\
\text{End point } 2 : (2+2, 8) = (4,8)
\end{gathered}
$$

Note that an edge mark array can represent any conforming triangulation of its quad, from the minimum triangulation to the finest triangulation and everything in between.

3.4 Refinement

3.4.1 Edge Split Criteria

Both bottom up and top down mesh refinement methods must know whether a given triangle edge needs to be split. A triangle edge is given by its endpoints (x_0, y_0) and (x_1, y_1), which mark positions in the quad that contains the triangle. The decision whether to split a given triangle edge is made by an edge split criterion.

A simple and cheap edge split criterion could compare the elevation values at both ends of the edge and decide to split the edge if their difference is greater than some threshold u:

$$
\text{split}\left((x_0, y_0), (x_1, y_1)\right) = \begin{cases} \text{true} & |h(x_0, y_0) - h(x_1, y_1)| > u, \\ \text{false} & \textit{otherwise} \end{cases}
$$

However, this split criterion does not take the current view into account, and cannot make any guarantees about the screen space error of the resulting mesh.

A split criterion that guarantees an upper bound t on the screen space error does the following: the screen space coordinates s_0 and s_1 of the edge endpoints are computed, as well as the screen space coordinates s_c of the edge midpoint as interpolated from the endpoints. Then, the screen space coordinates s_n of a new vertex at the edge midpoint are computed, using a new sample of the elevation map. If the screen space coordinates of the

interpolated and the new vertex differ by more than the screen space error threshold t, then the edge needs to be split:

$$
\begin{aligned}
h_0 &= h(x_0, y_0) \\
h_1 &= h(x_1, y_1) \\
s_0 &= \text{screen\_space}(x_0, y_0, h_0) \\
s_1 &= \text{screen\_space}(x_1, y_1, h_1) \\
s_c &= \text{screen\_space}(\frac{x_0 + x_1}{2}, \frac{x_1 + y_1}{2}, \frac{h_0 + h_1}{2}) \\
h_n &= h(\frac{x_0 + x_1}{2}, \frac{y_0 + y_1}{2}) \\
s_n &= \text{screen\_space}(\frac{x_0 + x_1}{2}, \frac{x_1 + y_1}{2}, h_n) \\
\text{split}\left((x_0, y_0), (x_1, y_1)\right) &= \begin{cases} \text{true} & |s_c - s_n| > t, \\ \text{false} & \textit{otherwise} \end{cases}
\end{aligned}
$$

This criterion leads to triangle splits only where necessary, depending on the current view. For example, triangles covering vastly different height samples are not split when viewed directly from above, because no screen space error occurs.

Note that this split criterion, when used in the framework presented in Chapter 1, requires the use of double precision computations to compute exact screen space coordinates on a global scale (see Sec. 1.4).

3.4.2 Bottom Up Refinement

Since all conforming quad triangulations can be represented by an edge mark array, it is not necessary to work on actual vertices to refine the quad mesh. Instead, the refinement method can work on the edge mark arrays, and geometry can be generated in a single, last step.

For this purpose, every mark in the edge mark arrays needs to store exactly 1 bit: whether to split the represented edge (1) or not (0). Initially, all edge marks are 0. The decision whether to split an edge can be made for each refinement level l consecutively, starting with $l = 2(k-1)$. An overview of bottom up refinement is given in Fig. 3.6. Details to each step are given in the following paragraphs.

In the initialization step, the five marks of each edge mark array that do not correspond to triangle pairs are initialized to 1, and the four marks of level 0 of each edge mark array are initialized to 1 if the neighboring quad has a higher level than the current quad (see Fig. 3.5).

```
1  for (all leaves L in the quadtree) {
2      initialize_level_0_marks(L);
3      for (l = 2*(k-1); l >= 1; l--) {
4          for (all marks (x,y) in level l)) {
5              tp = triangle_pair(x,y);
6              bool split_hypotenuse = false;
7              if (leg_is_marked_for_splitting(tp)) {
8                  split_hypotenuse = true;
9              }
10             else if (hypotenuse_needs_splitting(tp)) {
11                 split_hypotenuse = true;
12             }
13             set_mark(x,y,split_hypotenuse);
14         }
15     }
16 }
17 generate_mesh();
```

Figure 3.6: Overview of the bottom up refinement method.

In the split decision step, a hypotenuse of a triangle pair is marked for splitting if any one of the four legs of the triangle pair was already marked for splitting in the previous level, or if the edge split criterion decides to split the hypotenuse. This rule ensures that splits are correctly propagated from lower to higher levels.

If a split decision is made for an edge that lies on a quad border, and this quad has a neighbor quad that shares this edge, then the decision must be written to the edge mark array of the neighboring quad, too. See Fig. 3.7. This ensures consistent triangle splits across quads.

To handle hierarchy level differences in the quadtree, one special case has to be accounted for: if a triangle pair consists of two triangles that belong to neighboring quads, and the neighbor quad has a higher level than the current quad, then the two legs of the neighbor triangle were not marked yet. In this case, to be able to make the decision shown in line 7 of Fig. 3.6 for the current triangle pair, the mark of these two legs has to be computed in advance. Since levels of neighboring quads cannot differ by more than one in a restricted quadtree, this special case is enough to handle all hierarchy level differences.

When the edge marks for all levels are set, the edge mark arrays represents a conforming, adaptive triangulation of the quads in the rendering quadtree. This representation must then be transformed to a set of vertices in a last step. For this purpose, all marks in all edge mark arrays are examined. For each mark, the represented triangle pair is computed as described in Sec. 3.3.3. A triangle exists in the represented mesh (and thus

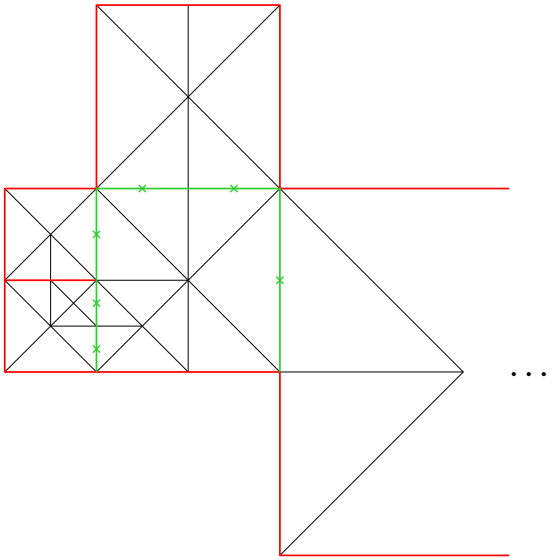

Figure 3.7: Examples for shared edges (green) of neighboring quads (red) in a triangulation based on a restricted quadtree.

needs to be converted to vertices) if and only if the mark for its hypotenuse is not set and the marks for all three triangle endpoints are set.

Bottom up refinement requires the evaluation of the split criterion for every edge mark array element. Since the split criterion is evaluated first for the shortest edges and then successively for all edge lengths up to the longest edges, no elevation map features are missed.

3.4.3 Top Down Refinement

Top down refinement starts with a coarse mesh and splits triangles where necessary.

Unlike bottom up refinement, for top down refinement it is not sufficient to test only the midpoint of an edge to guarantee a screen space error, since this might miss elevation map features that lie between the endpoints and the midpoint of a long edge. Instead, it is necessary to perform a number of tests, depending on the number h of elevation map samples covered by the edge. If the length s of the edge is short in screen space, the number of tests may be reduced accordingly. Specifically, the number n of necessary tests is

$$n = 2^i - 1, \quad i = \max_j \left\{ 2^j \leq \min\{h, s\} \right\}.$$

Additionally, the interior of a triangle must be checked in order to not miss any elevation map features there. The number of interior tests can

```
start with initial mesh M
do {
    do {
        for (all triangles t in M) {
            if (mark(leg0(t)) == 0) {
                if (edge_split_criterion(leg0)) {
                    set_mark(leg0(t), 1);
                }
                else {
                    set_mark(leg0(t), 2);
                }
            }
            if (mark(leg1(t)) == 0) {
                if (edge_split_criterion(leg1)) {
                    set_mark(leg1(t), 1);
                }
                else {
                    set_mark(leg1(t), 2);
                }
            }
            if (mark(hypotenuse(t)) != 1
                && (mark(leg0(t)) == 1 || mark(leg1(t)) == 1)) {
                set_mark(hypotenuse(t), 1);
            }
            else if (mark(hypotenuse(t)) == 0) {
                if (edge_split_criterion(hypotenuse(t))
                    || triangle_split_criterion(t)) {
                    set_mark(hypotenuse(t), 1);
                }
                else {
                    set_mark(hypotenuse(t), 2);
                }
            }
        }
    }
    until (no edges were marked for splitting);
    split marked edges, generating a new mesh M
}
until (no edges were split)
```

Figure 3.8: Overview of the top down refinement method.

be limited by the screen space size of the triangle in the same way as the number of edge tests.

This means that in contrast to bottom up refinement, the number of tests that have to be performed is limited by screen space, resulting in a smaller overall number of tests.

As in bottom up refinement, it would be possible to only work on edge mark arrays and generate actual vertices only in the last step of the refinement. However, since different parts of the quad can be refined at different levels at any given time, and since split propagation propagates from finer to coarser levels (see Fig. 3.2), this would make it necessary to process the complete edge mark array in each refinement step. Since the number of triangles in a quad mesh is usually much smaller than its theoretical maximum, it is cheaper to perform top down refinement on a triangle basis instead, while still keeping the necessary records in the edge mark arrays.

To keep track of edges that were already processed, two bits in each mark of an edge mark array are required to store three different states: edge is unprocessed (0), edge must be split (1), edge does not need to be split (2). The refinement process starts with a coarse conforming quadtree triangulation. This initial mesh is then refined in a two step iterative process. An overview is given in Fig. 3.8.

The first step is the mark step. Its immediate reiteration ensures that forced splits propagate through the mesh. This reiteration is very fast because it mostly works on cached data. The mark step works as follows. If the two legs of the current triangle were not processed yet, they are tested and marked accordingly. If the hypotenuse is not yet marked for splitting but one of the legs is, then the hypotenuse is marked for splitting, to ensure proper split propagation. Otherwise, if the hypotenuse is not yet processed, both the edge split criterion and the triangle split criterion are evaluated, and the hypotenuse is marked accordingly. Marked edges that are shared between two neighboring quads are always marked in the neighboring quad, too. See Fig. 3.7.

The second step is the split step. Every triangle of the input mesh is examined, and the edge mark arrays are used to determine the number of output triangles (between one and four). See Fig. 3.2. In a GPU implementation, this can be done efficiently by writing four output triangles, where some of them may be invalid, and removing the invalid triangles in a subsequent stream compaction step.

Top down refinement performance depends on the initial refinement level of the mesh: a fine initial mesh results in a large number of triangles that have to be processed in each refinement step. On the other hand, a

coarse initial mesh results in more refinement steps until the mesh fulfills the screen space error guarantee.

3.5 Implementation Notes

A GPU-based implementation of the mesh creation and refinement techniques is best done using general purpose programming environments for the GPU, such as CUDA or OpenCL. The presented top down refinement method has been implemented in OpenGL instead for the simplified case of a mesh for a single quad and without screen space error guarantees [Hei09], but writing to edge mark arrays by rendering points into special render targets proved to be cumbersome and too expensive to be used on a whole quadtree hierarchy.

The current implementation is based on CUDA. The edge mark arrays are stored as arrays of unsigned integer values, each storing 32 bits.

The bottom up refinement method uses one GPU thread per edge mark in the currently processed level l, while the top down refinement method uses one GPU thread per triangle of the current mesh. In both cases, the GPU threads can run completely independent of each other. The edge mark arrays are stored in global memory since they do not fit into shared memory on current graphics hardware. For these reasons, there is currently no need for a special arrangement of threads into thread blocks. Instead, the implementation simply runs as many threads in parallel as possible.

Concurrent threads may need to modify the same bits in the edge marker arrays at the same time. To prevent undefined results, special atomic operations provided by CUDA can be used for this purpose. However, in the case of collisions these atomic operations are relatively slow on current graphics hardware because colliding writes halt the accessing threads. To reduce this problem to a certain extent, the marks stored in the 32 bits of an unsigned integer can be organized in a way that makes collisions less likely, for example by representing a 2D subset of the edge mark array instead of consecutive marks. See Fig. 3.5.

Future generations of graphics hardware will allow faster atomic operations. In particular, the next generation of NVIDIA GPUs will introduce a read/write cache architecture that is explicitly designed for this purpose [NVI09b]. Therefore, the negative effects of atomic operation collisions will become less relevant.

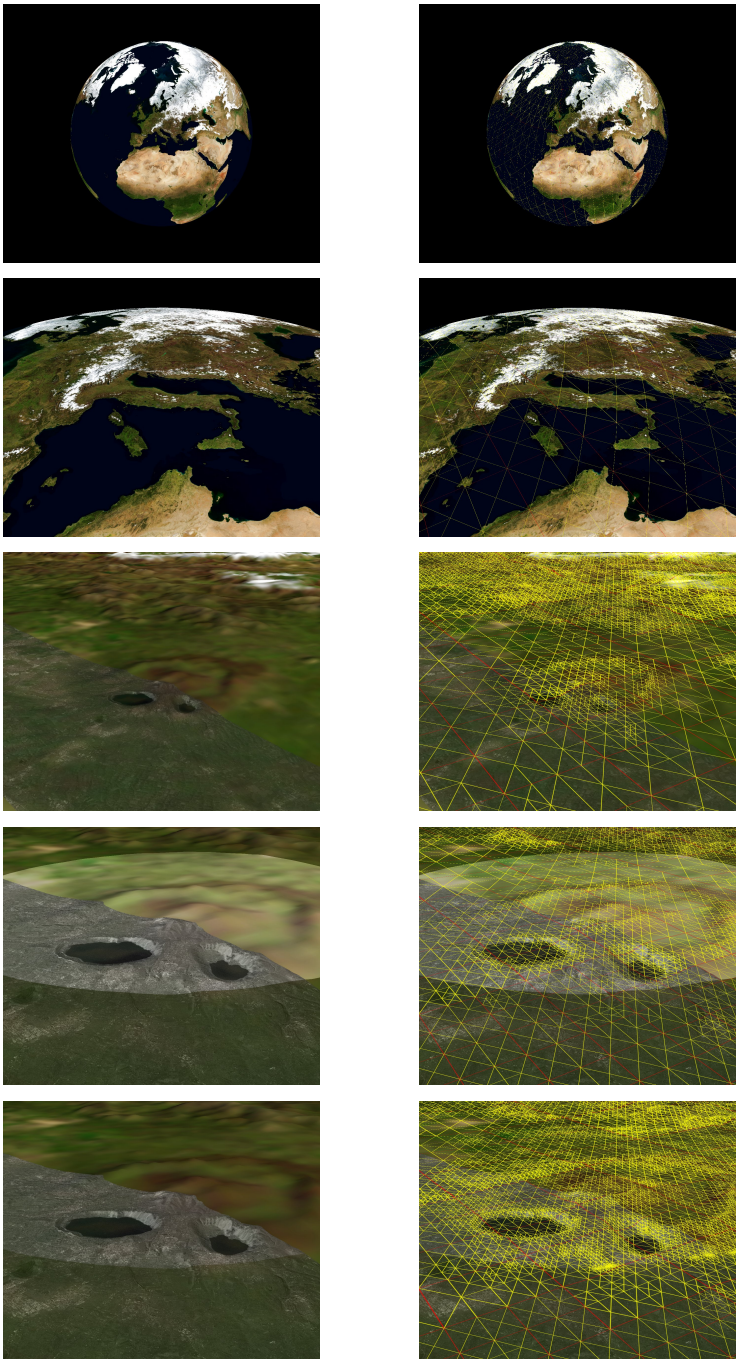

Figure 3.9: Five snapshots of an animated sequence, with and without the adaptive mesh. The sequence shows four different data sets in a combined view. It starts with a zoom on the area around Rome, Italy, and then changes various data processing and fusion parameters, resulting in dynamic changes of the terrain data.

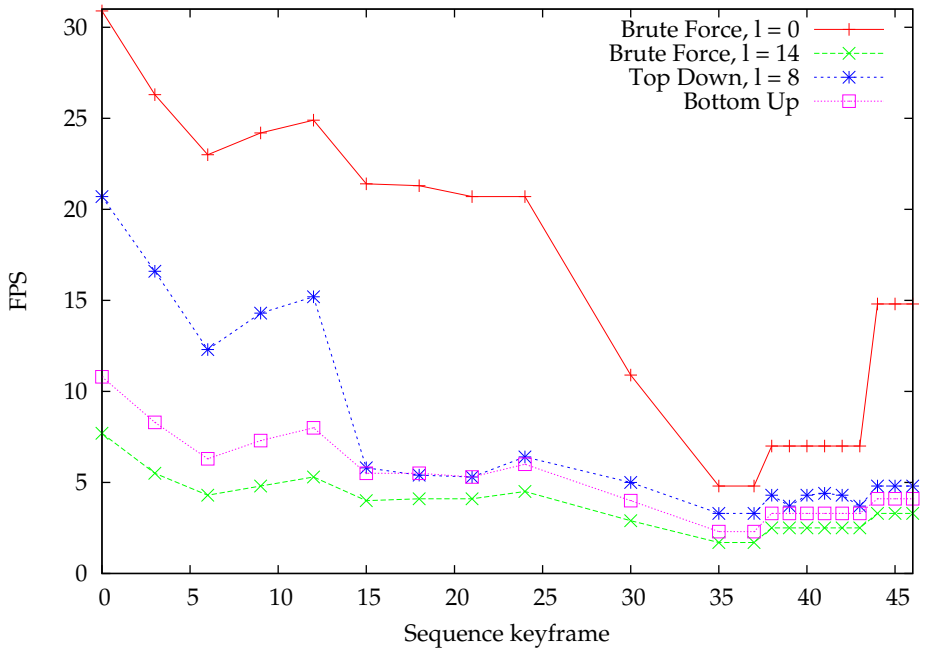

Figure 3.10: Frame rates achieved by different mesh creation and refinement methods for the test sequence shown in Fig. 3.9.

3.6 Results

The methods presented above were tested as part of the framework presented in Chapter 1 on PC hardware with an Intel Core2 Duo 3 GHz CPU, 8 GB main memory, and an NVIDIA GTX 285 graphics card with 2 GB of graphics memory. The software environment consisted of Debian 5.0 64 bit, NVIDIA CUDA 2.3 and the NVIDIA graphics driver version 190.53.

Fig. 3.9 shows an animated sequence that was used in the tests. Both the geometry and the texture are computed on-the-fly according to interactively defined methods. The geometry is a mixture of the DEM data set from the NASA Blue Marble Next Generation (BMNG) project and a DEM data set from the Shuttle Radar Topography Mission (SRTM) [JRNG08]. The texture is a mixture of a BMNG image and a high resolution TerraSAR-X amplitude image provided by Infoterra GmbH. Together, the original data sets amount to 263.4 GiB of data. The sequence zooms in on the area around Rome, Italy, and then changes various data processing and fusion parameters, resulting in dynamic changes of the terrain data.

The quad size used in the tests was 512×256 samples, plus a 9 sample wide border. The sequence was rendered in a 1280×1024 viewport. Four different meshes were used for the tests:

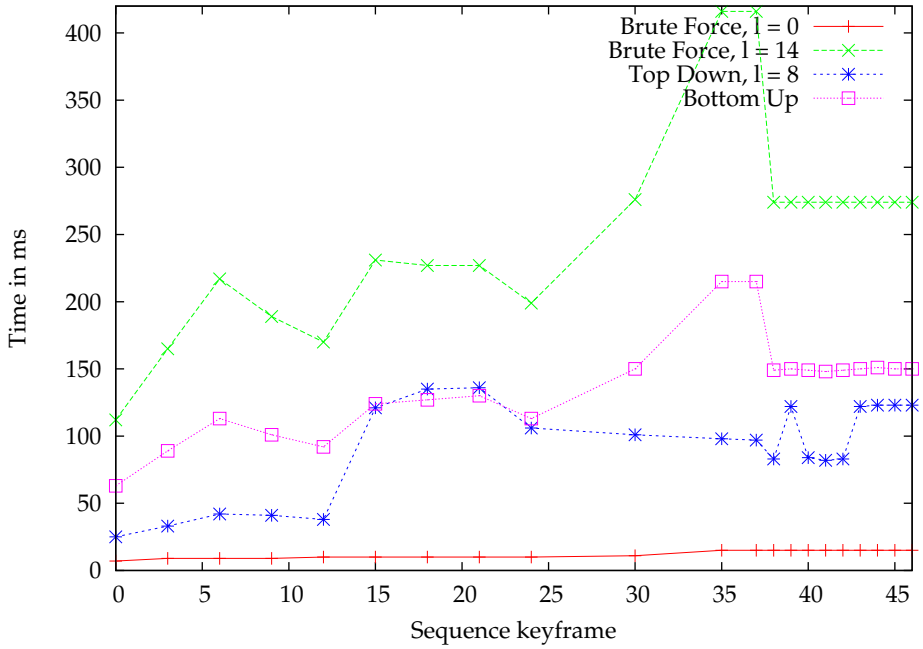

Figure 3.11: Time in milliseconds spent on mesh creation and refinement, without data processing and fusion, for the test sequence shown in Fig. 3.9.

- A brute force mesh with refinement level $l = 0$. This is the minimal conforming mesh.

- A brute force mesh with refinement level $l = 14$, which corresponds to the minimal conforming mesh after subdividing each quad 7 times (see Fig. 1.3; note the different meaning of the term *level*). Due to the quad size of 512×256 samples, this is the finest conforming mesh that can be generated with the method described in Sec. 1.4.3, but it still has edges with length 2 that potentially need splitting to achieve a given screen space error bound.

- A mesh refined using the top down method with the initial refinement level $l = 8$, which corresponds to the minimal conforming mesh after subdividing each quad 4 times. The screen space error bound for the this method was set to 1 pixel. The initial refinement level $l = 8$ was chosen because it consistently resulted in best performance in the tests.

- A mesh refined using the bottom up method. The screen space error bound for this method was set to 1 pixel.

Task	**Time** [ms]
Creation of base mesh	10.9
Refinement pass 1	
Marker pass (initial)	29.5
Marker passes (propagation)	1.2
Triangle splitting	12.0
Refinement pass 2	
Marker pass (initial)	5.8
Marker passes (propagation)	3.0
Triangle splitting	11.5
Refinement pass 3	
Marker pass (initial)	3.3
Marker passes (propagation)	2.7
Triangle splitting	12.6
Refinement pass 4	
Final marker pass (no change)	2.3
Total	94.8

Table 3.1: Time required by the top down refinement method for one example scene.

Fig. 3.10 shows the frame rates achieved with these four methods for the animated sequence. Note that the measured computation time includes all framework overhead, in particular the time that is spent on processing and fusion of dynamic terrain data for each frame. Therefore, frame rates that are comparable to those of methods that render static data cannot be achieved. Instead, the maximum achievable frame rate is that of the coarse brute force method (which produces meshes with very large errors), and the minimum sensible frame rate is that of the fine brute force method (which produces far too many triangles).

Since the time spent on data processing and fusion depends on the open data sets, the number of leaves in the rendering quadtree, and the interactively defined parameters, the maximum achievable frame rate differs significantly across the sequence.

To allow better comparison of the four mesh methods, Fig. 3.11 shows the time in milliseconds spent on mesh creation and refinement, without data processing and fusion. Note that all methods use double precision computations for exact screen space coordinates on a global scale (see Sec. 1.4), therefore even the coarse brute force method requires several milliseconds. Future generations of graphics hardware will reduce

the performance difference between single and double precision computations [NVI09b].

The bottom up refinement method consistently requires around 55% of the time used by the fine brute force method, regardless of the scene. The top down refinement method almost always requires less than 50% of the time used by the fine brute force method, and often less than 30%. This difference in behaviour is due to that fact that the top down method can skip edge split tests based on screen space considerations, while the bottom up method cannot. Consequently, the top down method performs best when it can skip a large percentage of tests. This is the case in scenes that contain a relatively large number of relatively small quadtree leaves. Examples for such scenes include the earth viewed from a distance (seconds 1 – 10 of the sequence), and scenes that have just switched to a finer level of detail in the rendering quadtree (around second 30 – 35 of the sequence).

Tab. 3.1 examines the computation time of top-down refinement for one example scene in more detail. The marker passes are relatively expensive because of the aforementioned use of double precision computations when evaluating the split criterion (see Sec. 3.4.1). The subsequent propagation passes, on the other hand, are very fast because they mostly work on cached data.

3.7 Summary

This chapter presented bottom up and top down refinement methods for conforming triangulations of restricted quadtree hierarchies. Both methods are based on the *edge mark array* data structure, which allows efficient parallel implementations that work entirely on the GPU.

The refinement methods are suitable as level of detail techniques for rendering dynamically generated terrain data. In this case, traditional terrain rendering approaches cannot be used because sophisticated preprocessing of the data is not possible.

The methods can guarantee an upper bound for the screen space error, which is important for visualization of Remote Sensing data.

The refinement methods do not yet achieve interactive frame rates when used in applications that make heavy use of the GPU also for other purposes. However, they will benefit from current trends in graphics hardware development, specifically faster atomic operations and double precision computations.

Conclusion

Summary

Remote Sensing data is of fundamental importance for analysis tasks in many diverse application areas. This is documented by the increasing number of specialized airborne and spaceborne earth observation systems, based on a wide variety of sensor technologies.

The visualization of Remote Sensing data requires the processing of sensor data to produce color and geometry information. The full scope of information contained in multimodal Remote Sensing data sets cannot be preserved in a single, static terrain data set produced in a preprocessing step. Therefore, interactive visualization systems are required to allow interactive generation of dynamic terrain data, tailored to both the properties of the input data, and the goal of the current visualization task.

To handle the challenges associated with interactive visualization of Remote Sensing data, this dissertation proposes

- A framework for GPU-based, interactive visualization of multimodal Remote Sensing data, described in Chapter 1. This framework provides the infrastructure for data management, processing, fusion, and rendering.
- GPU-based visualization techniques tailored to the special requirements of Synthetic Aperture Radar (SAR) data, described in Chapter 2. These techniques serve as examples for the specialized visualization methods that are required for different Remote Sensing modalities.
- Methods for accurate rendering of dynamically generated terrain data, described in Chapter 3. These methods use GPU-based adaptive mesh refinement to allow efficient rendering with guaranteed error bounds.

Combined, the presented methods allow interactive visualization of multimodal Remote Sensing data, including interactive adjustments of

sensor data processing and fusion methods.

Not all problems associated with accurate planetary scale visualization are solved satisfactorily yet. In particular, the data structures show limitations in proximity to the planet poles (see Sec. 1.2.2 and Sec. 1.2.6), and some intermediate steps require double precision computations, which reduces the achievable frame rate (see Sec. 1.4.3 and Sec. 3.4.1). Still, interactive handling of Remote Sensing data on a planetary scale is possible with the presented methods.

Future Work

The size and diversity of Remote Sensing data sets continues to grow rapidly. New and improved sensor systems continue to produce more and more data sets with increasing spatial and spectral resolution. Moreover, time series of Remote Sensing data sets increasingly gain importance. Such time series can contain data sets produced during periodic flyovers by the same sensor system, but also data sets produced by different sensors if these sensors provide sufficient data consistency.

Visualization tools are required to extract the relevant information from this fast growing variety of data sources.

First, specialized tools tailored to specific modalities are required, analogous to the SAR visualization methods proposed in this dissertation. Important modalities that currently pose significant challenges for visualization systems include multi- and hyperspectral sensor data, for example.

Once individual modalities can be handled with adequate interactive visualization tools, it is also important to consider cross-modal data visualization and analysis, i.e. to combine data sets from different sensor systems for joint visual analysis. The basic data fusion infrastructure presented in this dissertation can only be a starting point in this regard.

Furthermore, visual analysis of Remote Sensing data time series has not been investigated yet. New visualization approaches have to be developed for related tasks such as visual change detection and analysis.

To cope with these demands, it will not be sufficient to simply rely on the increasing computational power provided by parallel processing hardware. Instead, new visualization concepts and methods have to be based on new capabilities and increased flexibility offered by new hardware generations.

Bibliography

[AG06] Atlan S., Garland M.: Interactive Multiresolution Editing and Display of Large Terrains. *Computer Graphics Forum 25*, 2 (2006), 211–223.

[AMD] AMD Corporation: AMD "Close to Metal" Press Release. `http://www.amd.com/us/press-releases/Pages/Press_Release_114147.aspx`. Accessed 2010-10-13.

[AMMS08] Aydin T. O., Mantiuk R., Myszkowski K., Seidel H.-P.: Dynamic Range Independent Image Quality Assessment. In *Proc. ACM SIGGRAPH* (2008), pp. 69:1–69:10.

[Ash02] Ashikhmin M.: A Tone Mapping Algorithm for High Contrast Images. In *Proc. Eurographics Workshop on Rendering* (2002), pp. 145–156.

[Bam92] Bamler R.: A Comparison of Range-Doppler and Wavenumber Domain SAR Focusing Algorithms. *IEEE Trans. Geoscience and Remote Sensing 30*, 4 (1992), 706–713.

[BBBK07] Borst C. W., Baiyya V. B., Best C. M., Kinsland G. L.: Volumetric Windows: Application to Interpretation of Scientific Data, Shader-Based Rendering Method, and Performance Evaluation. In *Proc. Int. Conf. Computer Graphics and Virtual Reality* (2007), pp. 72–78.

[BE06] Brenner A. R., Ender J. H. G.: Demonstration of Advanced Reconnaissance Techniques with the Airborne SAR/GMTI Sensor PAMIR. In *IEE Proc. Radar, Sonar and Navigation* (2006), pp. 152–162.

[BFH*04] Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M., Hanrahan P.: Brook for GPUs: Stream Com-

puting on Graphics Hardware. In *Proc. ACM SIGGRAPH* (2004), pp. 777–786.

[BGP09] Boesch J., Goswami P., Pajarola R.: RASTeR: Simple and Efficient Terrain Rendering on the GPU. In *Proc. Eurographics (Areas Papers)* (2009), pp. 35–42.

[BPN08] Bhattacharjee S., Patidar S., Narayanan P. J.: Real-time rendering and manipulation of large terrains. In *Proc. Indian Conf. on Computer Vision, Graphics & Image Processing* (2008), pp. 551–559.

[BS05] Brekke C., Solberg A. H.: Oil Spill Detection by Satellite Remote Sensing. *Remote Sensing of Environment 95*, 1 (2005), 1–13.

[BSP*93] Bier E. A., Stone M. C., Pier K., Buxton W., DeRose T. D.: Toolglass and Magic Lenses: The See-Through Interface. In *Proc. ACM SIGGRAPH* (1993), pp. 73–80.

[Cam08] Campbell J. B.: *Introduction to Remote Sensing*, fourth ed. The Guilford Press, 2008.

[CGG*03] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM). In *Proc. IEEE Visualization* (2003), pp. 147–154.

[CHS*93] Chiu K., Herf M., Shirley P., Swamy S., Wang C., Zimmerman K.: Spatially Nonuniform Scaling Functions for High Contrast Images. In *Proc. Graphics Interface* (1993), pp. 245–253.

[ČWNA06] Čadík M., Wimmer M., Neumann L., Artusi A.: Image Attributes and Quality for Evaluation of Tone Mapping Operators. In *Proc. Pacific Graphics* (2006), pp. 35–44.

[DD02] Durand F., Dorsey J.: Fast Bilateral Filtering for the Display of High-Dynamic-Range Images. In *Proc. ACM SIGGRAPH* (2002), pp. 257–266.

[Dia08] Diard F.: *GPU Gems 3*. Addison-Wesley, 2008, ch. Using the Geometry Shader for Compact and Variable-Length GPU Feedback, pp. 891–907.

[DMAC03] Drago F., Myszkowski K., Annen T., Chiba N.: Adaptive Logarithmic Mapping for Displaying High Contrast Scenes. *Computer Graphics Forum 22*, 3 (2003), 419–426.

[Don95] Donoho D.: De-noising by Soft-Thresholding. *IEEE Trans. Information Theory 41*, 3 (1995), 613–627.

[DS04] Dachsbacher C., Stamminger M.: Rendering Procedural Terrain by Geometry Image Warping. In *Proc. Eurographics Symposium on Rendering* (2004), pp. 103–110.

[DSW09] Dick C., Schneider J., Westermann R.: Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering. *Computer Graphics Forum 28*, 1 (2009), 67–83.

[DWS*97] Duchaineau M., Wolinsky M., Sigeti D. E., Miller M. C., Aldrich C., Mineev-Weinstein M. B.: ROAMing Terrain: Real-time Optimally Adapting Meshes. In *Proc. IEEE Visualization* (1997), pp. 81–88.

[Eil] Eilemann S.: The Equalizer Parallel Rendering Framework. `http://www.equalizergraphics.com/`. Accessed 2010-10-25.

[EMP09] Eilemann S., Makhinya M., Pajarola R.: Equalizer: A Scalable Parallel Rendering Framework. *IEEE Trans. Visualization and Computer Graphics 15*, 3 (2009), 436–452.

[FJ04] Fairchild M. D., Johnson G. M.: The iCAM Framework for Image Appearance, Differences, and Quality. *Journal of Electronic Imaging 13*, 1 (2004), 126–138.

[FPR01] Ferretti A., Prati C., Rocca F.: Permanent Scatterers in SAR Interferometry. *IEEE Trans. Geoscience and Remote Sensing 39*, 1 (2001), 8–20.

[GDA] GDAL – Geospatial Data Abstraction Library. `http://www.gdal.org/`. Accessed 2010-10-13.

[GJ97] Gagnon L., Jouan A.: Speckle Filtering of SAR Images – A Comparative Study Between Complex-Wavelet-Based and Standard Filters. In *Proc. SPIE Vol. 3169* (1997), pp. 80–91.

[GPG] GPGPU – General-Purpose Computation on Graphics Hardware. `http://www.gpgpu.org/`. Accessed 2010-10-13.

[GZY08] Gong L., Zhang J., , Yi L.: Detection of Artificial Corner Reflector on SAR Images. In *Proc. Dragon Symposium* (2008).

[HCP02] He Y., Cremer J., Papelis Y.: Real-Time Extendible-Resolution Display of On-line Dynamic Terrain. In *Proc. Graphics Interface* (2002), pp. 151–160.

[Hei09] Heide F.: *Adaptive Terrain Rendering with Smooth Subdivision Surfaces on the GPU*. Bachelor Thesis, University of Siegen, 2009.

[Jar04] Jargstorff F.: *GPU Gems*. Addison-Wesley, 2004, ch. A Framework for Image Processing, pp. 445–467.

[JRNG08] Jarvis A., Reuter H., Nelson A., Guevara E.: Hole-filled Seamless SRTM Data V4, 2008. International Centre for Tropical Agriculture (CIAT).

[Khr] Khronos Group: OpenCL API Registry. `http://www.khronos.org/registry/cl/`. Accessed 2010-10-13.

[KLJ*09] Kooima R., Leigh J., Johnson A., Roberts D., SubbaRao M., DeFanti T. A.: Planetary-Scale Terrain Composition. *IEEE Trans. Visualization and Computer Graphics 15*, 5 (2009), 719–733.

[KLT*09] Kolb A., Lambers M., Todt S., Cuntz N., Rezk-Salama R.: Immersive Rear Projection on Curved Screens. IEEE VR (Poster-Session), 2009.

[KMS05] Krawczyk G., Myszkowski K., Seidel H.-P.: Perceptual Effects in Real-Time Tone Mapping. In *Spring Conf. Computer Graphics* (2005), ACM, pp. 195–202.

[KW05] Kipfer P., Westermann R.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Improved GPU Sorting, pp. 733–746.

[LCTS05] Ledda P., Chalmers A., Troscianko T., Seetzen H.: Evaluation of Tone Mapping Operators using a High Dynamic Range Display. In *Proc. ACM SIGGRAPH* (2005), pp. 640–648.

[LH04] Losasso F., Hoppe H.: Geometry Clipmaps: Terrain Rendering using Nested Regular Grids. *Proc. ACM SIGGRAPH 23*, 3 (2004), 769–776.

[LJD*94] Lee J. S., Jurkevich L., Dewaele P., Wambacq P., Oosterlinck A.: Speckle Filtering of Synthetic Aperture Radar Images: A Review. *Remote Sensing Reviews 8* (1994), 313–340.

[LK08a] Lambers M., Kolb A.: Adaptive Dynamic Range Reduction for SAR Images. In *Proc. 7th European Conference on Synthetic Aperture Radar (EUSAR)* (2008), vol. 3, pp. 371–374.

[LK08b] Lambers M., Kolb A.: Automatic Point Target Detection For Interactive Visual Analysis of SAR Images. In *Proc. IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)* (2008), vol. 2, pp. II–903–II–906.

[LK09] Lambers M., Kolb A.: GPU-Based Framework for Distributed Interactive 3D Visualization of Multimodal Remote Sensing Data. In *Proc. IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)* (2009), vol. 4, pp. IV–57–IV–60.

[LK10a] Lambers M., Kolb A.: Dynamic Terrain Rendering. *3D Research 1*, 4 (2010), 1–8.

[LK10b] Lambers M., Kolb A.: Visual Assistance Tools for Interactive Visualization of Remote Sensing Data. In *Proc. IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)* (2010), pp. 4745–4748.

[LKES09] Livny Y., Kogan Z., El-Sana J.: Seamless Patches for GPU-Based Terrain Rendering. *Vis. Comput. 25*, 3 (2009), 197–208.

[LKNK07] Lambers M., Kolb A., Nies H., Kalkuhl M.: GPU-based Framework for Interactive Visualization of SAR Data. In *Proc. IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)* (2007), pp. 4076–4079.

[LKR*96] Lindstrom P., Koller D., Ribarsky W., Hodges L. F., Faust N., Turner G. A.: Real-time, Continuous Level of Detail Rendering of Height Fields. In *Proc. ACM SIGGRAPH* (1996), pp. 109–118.

[LNK08] Lambers M., Nies H., Kolb A.: Interactive Dynamic Range Reduction for SAR Images. *Geoscience and Remote Sensing Letters 5*, 3 (2008), 507–511.

[LNPK04] Loffeld O., Nies H., Peters V., Knedlik S.: Models and Useful Relations for Bistatic SAR Processing. *IEEE Trans. Geoscience and Remote Sensing 42*, 10 (2004), 2031–2038.

[LSK*06] Lefohn A. E., Sengupta S., Kniss J., Strzodka R., Owens J. D.: Glift: Generic, Efficient, Random-Access GPU Data Structures. *ACM Trans. Graph. 25* (2006), 60–99.

[Mal06] Malan H.: *OpenGL Shading Language*, 2nd ed. Addison-Wesley, 2006, ch. RealWorldz, pp. 505–541.

[MDTP*04] McCool M., Du Toit S., Popa T., Chan B., Moule K.: Shader Algebra. In *Proc. ACM SIGGRAPH* (2004), pp. 787–795.

[MIA*07] McCormick P., Inman J., Ahrens J., Mohd-Yusof J., Roth G., Cummins S.: Scout: A Data-Parallel Programming Language for Graphics Processors. *Parallel Comput. 33*, 10-11 (2007), 648–662.

[MJ04] McCandless S. W., Jackson C. R.: *Synthetic Aperture Radar Marine User's Manual*. US Dept. of Commerce, 2004, ch. Principles of Synthetic Aperture Radar, pp. 1–23.

[MKM89] Musgrave F. K., Kolb C. E., Mace R. S.: The Synthesis and Rendering of Eroded Fractal Terrains. In *Proc. ACM SIGGRAPH* (1989), pp. 41–50.

[Mor00] Moreira A.: *Radar mit Synthetischer Apertur: Grundlagen und Signalverarbeitung*. Habilitationsschrift, Karlsruhe University, 2000.

[Nar81] Narendra P. M.: A Separable Median Filter for Image Noise Smoothing. *IEEE Trans. Pattern Analysis and Machine Intelligence 3*, 1 (1981), 20–29.

[NGA] NGA (US National Geospatial-Intelligence Agency): World Geodetic System 1984 (WGS84). `http://www1.nga.mil/ProductsServices/GeodesyGeophysics/WorldGeodeticSystem/Pages/default.aspx`. Accessed 2011-05-12.

[Nov05] Novosad J.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Advanced Hight Quality Filtering, pp. 417–435.

[NVI] NVIDIA Corporation: CUDA Zone. `http://www.nvidia.com/cuda`. Accessed 2010-10-13.

[NVI09a] NVIDIA Corporation: 30-Bit Color Technology for NVIDIA Quadro. `http://www.nvidia.com/docs/IO/40049/TB-04701-001_v02_new.pdf`, 2009. Accessed 2010-12-02.

[NVI09b] NVIDIA Corporation: NVIDIA's Next Generation CUDA Compute Architecture: Fermi. `http://www.nvidia.`

com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009. Accessed 2010-10-25.

[NVI10] NVIDIA Corporation: *NVIDIA CUDA C Programming Guide 3.1.1*. Sept. 2010.

[OLG*07] Owens J. D., Luebke D., Govindaraju N., Harris M., Krüger J., Lefohn A. E., Purcell T.: A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum 26*, 1 (2007), 80–113.

[OQ04] Oliver C., Quegan S.: *Understanding Synthetic Aperture Radar Images*. SciTech, 2004.

[Owe05] Owens J.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Streaming Architectures and Technology Trends, pp. 457–470.

[Paj98] Pajarola R.: Large Scale Terrain Visualization using the Restricted Quadtree Triangulation. In *Proc. IEEE Visualization* (1998), pp. 19–26.

[PFFG98] Pattanaik S. N., Ferwerda J. A., Fairchild M. D., Greenberg D. P.: A Multiscale Model of Adaptation and Spatial Vision for Realistic Image Display. In *Proc. ACM SIGGRAPH* (1998), pp. 287–298.

[PG07] Pajarola R., Gobbetti E.: Survey of Semi-Regular Multiresolution Models for Interactive Terrain Rendering. *Vis. Comput. 23*, 8 (2007), 583–605.

[RD05] Reinhard E., Devlin K.: Dynamic Range Reduction Inspired by Photoreceptor Physiology. *IEEE Trans. Visualization and Computer Graphics 11*, 1 (2005), 13–24.

[RH04] Reigber A., Hellwich O.: RAT (Radar Tools): A Free SAR Image Analysis Software Package. In *Proc. EUSAR* (2004), pp. 997–1000.

[RJW96] Rahman Z., Jobson D. J., Woodell G. A.: A Multiscale Retinex for Color Rendition and Dynamic Range Compression. In *SPIE Proc. Applications of Digital Image Processing XIX* (1996), vol. 2847.

[RSKK06] Rezk-Salama C., Keller M., Kohlmann P.: High-Level User Interfaces for Transfer Function Design with Semantics. *IEEE*

Trans. Visualization and Computer Graphics 12, 5 (2006), 1021–1028.

[RSSF02] Reinhard E., Stark M., Shirley P., Ferwerda J.: Photographic Tone Reproduction for Digital Images. *Proc. ACM SIGGRAPH* (2002), 267–276.

[RWPD05] Reinhard E., Ward G., Pattanaik S., Debevec P.: *High Dynamic Range Imaging: Acquisition, Display and Image-based Lighting*. Morgan Kaufmann, 2005.

[SBA*99] Srivastava S. K., Banik B. T., Adamovic M., Gray R., Hawkins R. K., Lukowski T. I., Murnaghan K. P., Jefferies W. C.: RADARSAT-1 Image Quality – Update. In *CEOS SAR Workshop* (1999).

[Sch95] Schlick C.: *Photorealistic Rendering Techniques*. Springer, 1995, ch. Quantization Techniques for the Visualization of High Dynamic Range Pictures.

[SDK05] Strzodka R., Doggett M., Kolb A.: Scientific Computation for Simulations on Programmable Graphics Hardware. *Simulation Practice & Theory 13*, 8 (2005), 667–680.

[SHG09] Satish N., Harris M., Garland M.: Designing Efficient Sorting Algorithms for Manycore GPUs. In *Proc. IEEE Int. Symposium on Parallel & Distributed Processing* (2009), pp. 1–10.

[Sho10] Short N. M.: The Remote Sensing Tutorial. `http://rst.gsfc.nasa.gov/`, 2010. Accessed 2010-10-13.

[SW06] Schneider J., Westermann R.: GPU-Friendly High-Quality Terrain Rendering. *Journal of WSCG 14*, 1–3 (2006), 49–56.

[Tho05] Thorne C.: Using a Floating Origin to Improve Fidelity and Performance of Large, Distributed Virtual Worlds. In *Proc. Int. Conf. on Cyberworlds* (2005), pp. 263–270.

[Tou02] Touzi R.: A Review of Speckle Filtering in the Context of Estimation Theory. *IEEE Trans. Geoscience and Remote Sensing 40*, 11 (2002), 2392–2404.

[TT99] Tumblin J., Turk G.: LCIS: A Boundary Hierarchy for Detail-Preserving Contrast Reduction. In *Proc. ACM SIGGRAPH* (1999), pp. 83–90.

[Ulr00] Ulrich T.: Continuous LOD Terrain Meshing Using Adaptive Quadtrees. http://www.gamasutra.com/features/20000228/ulrich_01.htm, Feb. 2000. Accessed 2011-02-01.

[VCWP96] Viega J., Conway M. J., Williams G., Pausch R.: 3D Magic Lenses. In *Proc. ACM Symp. User Interface Software and Technology* (1996), pp. 51–58.

[War05] Warden P.: *GPU Gems 2*. Addison-Wesley, 2005, ch. GPU Image Processing in Apple's Motion, pp. 393–408.

[WEB*10] Walterscheid I., Espeter T., Brenner A., Klare J., Ender J., Nies H., Wang R., Loffeld O.: Bistatic SAR Experiments With PAMIR and TerraSAR-X – Setup, Processing, and Image Results. *IEEE Trans. Geoscience and Remote Sensing 48*, 8 (2010), 3268–3279.

[WLHW07] Wong T.-T., Leung C.-S., Heng P.-A., Wang J.: Discrete Wavelet Transform on Consumer-Level Graphics Hardware. *IEEE Trans. Multimedia 9*, 3 (2007), 668–673.

[WMD*04] Wahl R., Massing M., Degener P., Guthe M., Klein R.: Scalable Compression and Rendering of Textured Terrain Data. *Journal of WSCG 12*, 3 (2004), 521–528.

[WS92] White R., Stemwedel S.: The Quadrilateralized Spherical Cube and Quad-Tree For All Sky Data. In *Astronomical Data Analysis Software and Systems I* (1992), Worrall D., Biemesderfer C., Barnes J., (Eds.), vol. 25 of *Astronomical Society of the Pacific Conference Series*, pp. 379–381.

[XLM03] Xiao J., Li J., Moody A.: A Detail-Preserving and Flexible Adaptive Filter for Speckle Suppression in SAR Imagery. *Int. J. Remote Sensing 24*, 12 (2003), 2451–2465.

[YMMS06] Yoshida A., Mantiuk R., Myszkowski K., Seidel H.-P.: Analysis of Reproducing Real-World Appearance on Displays of Varying Dynamic Range. In *Proc. Eurographics* (2006), pp. 415–426.