Decoupling Mesh and Data Representations For Geo–spatial Data Visualization

by

Sherin Al Shbat

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

Doctor of Philosophy
in
Computer Science

Approved Dissertation Committee

Prof. Dr. Ing. Lars Linsen (Supervisor)

Prof. Dr. Vikram Unnithan

Jun.-Prof. Dr. Paul Rosenthal (External)

Date of Defense: May 11, 2012

School of Engineering and Science

To the Mas- senger of ALAH MOHAMMED (peace be upon him) , To my mother DAHOK SHEHAB, *To the soul of my father* MOHAMMED AL SHBAT, *To the Bleeding* SYRIA *and all its* MARTYRS, *To my sister Dr.* SHAHINAZ, *To my sister* BASSMA, *To my brother* BASSAM, *To my brothers the twins* READA *&* FARHAN, *To my sister* NOUR *and her kids* FATEMA *&* WALED, *To my German spiritual mother* HELMA MAYER *& her daughter Dr.* DAGMER.

♡

Table of Contents

Acknowledgements

Foremost, I would like to thank Professor Dr. Lars Linsen, my supervisor, for his many suggestions, discussions, comments and constant support during this research at Jacobs University Bremen, Germany. I am also thankful to Professor Dr. Vikram Unnithan for his guidance about geo–science data.

My sincere thanks goes to the dean of School of Engineering & Science Prof. Dr. Bernhard Kramer for his encouragement, and insightful leadership that supports every student for hard issues. Alongside, a special gratitude goes to Dr. Svenja Frischholz who was enlightening my way in every difficult time.

I thank my fellow (VCGL) labmates especially Dr. Tran Van Long who I witnessed the birth of his Dr. epithet and I wish him a fruitful carrer.

Last but not the least, I would like to thank my family: my parents Dahok Shehab and Mohammed AL Shbat, for giving birth to me at the first place and supporting me spiritually throughout my life and I ask the mercy from God for the soul of my father who passed away during the time of doing my research and may give him a supreme residence of the paradise.

Sherin, AL Shbat
Bremen, May 2012

Abstract

Nowadays, data visualization plays an important role in geo-spatial data investigations, specifically in the context of visual exploration at interactive frame rates. Geo-spatial data typically include geographic information in form of a set of gridded or un-gridded points, where each point identifies a location on the earth's surface. A variety of projection methods are in use to re-produce the location of a specific data point. For example, in contrast to the longitude/latitude projection system, the UTM (Universal Transverse Mercator) system provides a conformal projected location of the earth based on a non-linear scaling in both easting and northing. In this work, geo-spatial data such as terrain or ocean floor data are represented as 2D coordinates, e.g., longitude/latitude, and a number of associated attributes of measured information such as depth or height, backscattering, sub-bottom profile information, etc.

Both main and graphical memory capacities are steadily increasing, but so are data sizes. Hence, current and future storage capacities still lack the ability to handle the targeted enormous data sizes with proper efficiency. To allow for interactive exploration, the number of rendered objects in most dynamic approaches is obligated to be below a certain complexity at run time. Fulfilling such goals requires several techniques to be integrated in our data exploration approach. View-dependency, LOD (level of detail) representations, and multi-resolution methods form the basis to tackle the objects redundancy, dynamic data updates, and simplification.

Existing approaches can be classified into two main groups, the ones that operate on irregular triangular meshes and the ones that operate on regular grids. While irregular triangular meshes can provide better adaptivity, regular grids can be handled more efficiently. Our approach is to decouple mesh and data representations such that data management is performed efficiently on regular grids while mesh rendering is executed using a fixed adaptive triangular mesh in parameter plane. The triangular mesh is optimized with respect to the projected triangle sizes and shapes. During interaction we update the mesh by merely adjusting the vertices' heights, which are queried from an underlying multi-resolution grid structure. The triangular mesh and the multi-resolution grid structure are precomputed and cached on the GPU. A tiling strategy is employed for the grid structure. We are able to generate very high frame rates using triangles with optimized shape in a

high-quality rendering.

Our approach is applied to different data types as inputs such as terrain data and bathymetry data. Terrain data include heightfield and color information that is acquired using aerial instruments. From the given data, hierarchies are generated in a pre-processing step. Bathymetry data are acquired using multibeam sonar. The measured data include information on the structure and nature of the ocean floor and gas maps underneath the ocean surface. Data exploration, ocean floor rendering, and further data visualization methods are employed in an interactive system. Our work provides highperformance algorithms and evaluates them utilizing synthetic as well as real data from both terrain and bathymetry measurements. Our contribution in the terrain or ocean floor rendering field is providing a new and fast method for heightfield exploration and rendering with frame rates exceeding those of most state-of-the-art approaches while providing optimal triangle shapes. In particular, using a fixed mesh, some interactive operations do not require data updates. Moreover, the interactive visual system comprises visualization of backscatter and water column data incorporated in the final rendering scene, which is a novel visualization task that has not yet been tackled by the existing 3D visualization systems for bathymetry data such as Fledermaus.

Chapter 1

Introduction

> *The intellectual takes as a starting point his self and relates the world to his own sensibilities; the scientist accepts an existing field of knowledge and seeks to map out the unexplored terrain.*
>
> **Daniel Bell**

Geo–spatial information has become ubiquitous and is being used by a large population, e.g., via on-line services and geographical information systems. In here, visualization provides important tools for geo–spatial information analysis and emphasizes the productive communication between user's interfaces and the format of the provided information that can be driven from distinct data sources. Thus, visualization tools allow the users to access and manipulate data for the purpose of exploration and investigation. In geosciences the visualization tasks include providing methods to solve some issues such as:

- Identifying the ocean floor characteristics including rocks, minerals, oil, and gas.
- Terrain data rendering with understanding the remote sensing images.
- Representing the uncertainty and the fuzziness of geo–spatial data in which the challenges could be focused and tackled.
- Introducing tools for interactive analysis purposes.

However, geo–spatial information belongs to a wide field of visualization for different applications.

1.1 Geo–spatial Data

The prominent sources of geo–spatial data are embedded in a variety of institutions' depository libraries. The majority of those organizations have critical purposes concerning both government and academic aspects as follows:

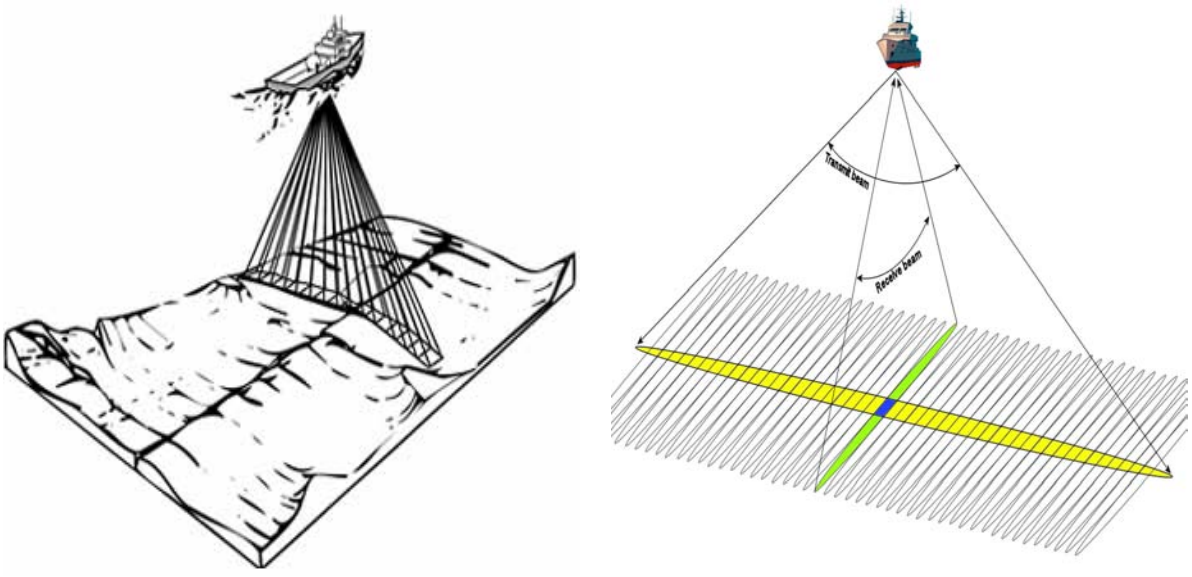
- The observation of predestined geographical locations. In here, the essential focus is on the elevation information implying:
  - Landscape planning (plant modeling, city building, and urban).
- The prediction of the natural resources in which the focus is mostly on the depth information. Thus, ocean floor is the interest of associations as:
  - Fishery agencies and organizations for fishing habitat investigation and improvement where the ocean floor data support the targeted data such as water–column data.
  - Offshore drilling companies who are deemed to possess the information of ocean floor surfaces and sub–surfaces. Those information are the base of any extraction approach of natural oil, gas, and suchlike minerals.

1.2 Data Gathering

There are a lot of data acquisition methods to gather geo–spatial data including remote sensing and on–site measurements. In this context, we focus on two applications: terrain data from remote sensing and ocean data from on–site measurements using multibeams. Therefore, a set of instruments have been used. The necessity of mapping large swaths of seabed for sediment features and living organisms as a fundamental step in every scientific benthic management system introduced the advent of *Multibeam Sonar Survey (MSS)* systems that are reckoned to be one of the recent acoustic tools. In principle, *MMS* technologies stem from a *SOund Navigation And Ranging (SONAR)* technique that measures the distance from an object in seabed via sending an acoustic signal, known as ping, see Figure 1.1 (b), through the water utilizing a transducer that is attached to the bottom of the vessel. *Single Beam Echo Sounders (SBES)* measure the amplitude in beams within a narrow swath. *Multibeam Echo Sounders (MBES)* systems are used to measure and analyze the

energy of the returned signals from the ocean floor or any other objects. *MBES* systems transmit acoustic waves using transducers that are attached to the hull of the vessel. The transmitted signals are formed in a fan shape, i.e, a swath. This fan covers an area of the seabed with a range which is determined depending on the water depth underneath the vessel. Usually, the coverage range is taken as two to four times the water depth. A set of receivers are also prepared to detect the returned signals. Mainly, *MBES* systems measure and record the traveling time of the transmitted acoustic signals to reach the receivers.

In comparison with *SBES*, *MBES* provide an array of sound signals to cover a wide swath, see Figure 1.1 (a). Thus more accurate sampling at higher resolution is gained.



(a) (b)

Figure 1.1: A wide swath is the result of an MBES technique [14] (a), and for each swath an array of pings are detected [59] (b).

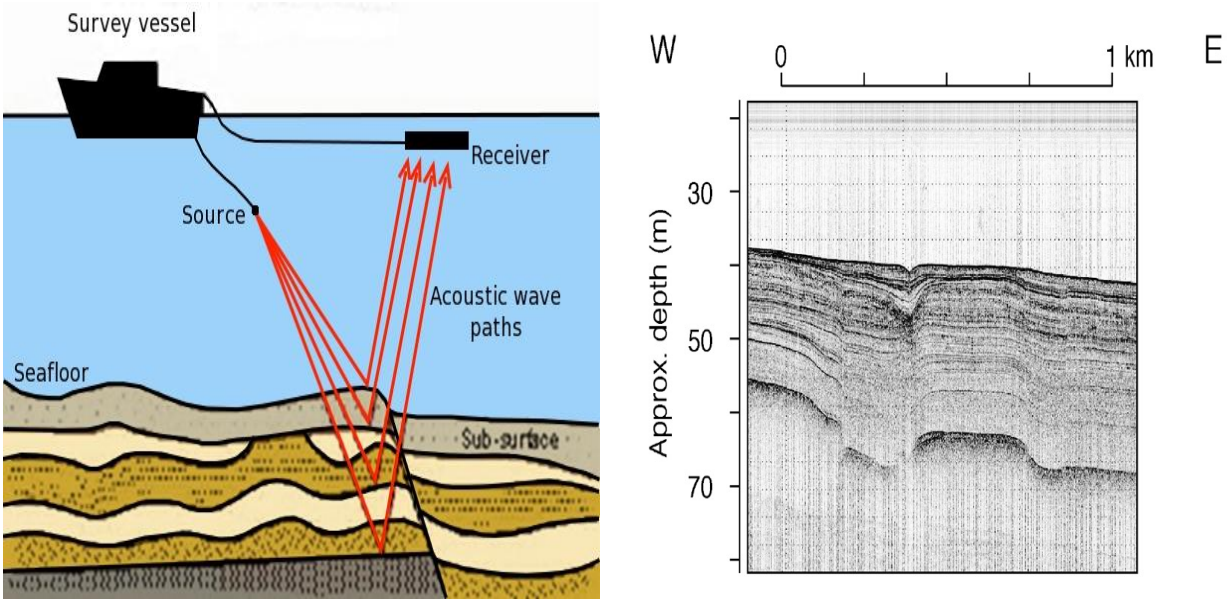
Data acquisition using SBES relies on the incident of transducers' pulse emissions on the seabed and the measurements of the reflected sounds. The reflections provide a variety of data attributes by recording the amplitude of returning energy.

For example **Backscatter (BS)** reflections are used to collect the acoustic energy of the received signals. In this way, one can obtain data by measuring the BS reflection intensity which describes the ocean floor properties as a thin layer of the ocean bed is penetrated by acoustic signals.

The resulting data draw the seafloor roughness, i.e., discovering if the bottom has either a rocky structure or gravels.

In addition to the seabed structure the strength of the reflected energy can recognize the bottom nature and it distinguishes between soft (such as muddy or sandy) and hard (such as rocky) flat areas if they have the same roughness.

For analyzing sediments beneath the seafloor surface low–frequency acoustic pulses are sent. Hence, transmitted signals penetrate seafloor strata. Thus, the reflections of sub–surfaces are received by a **Sub–Bottom Profiling (SBP)** system that produces data sets concerning sediment structure, or the existence of solid minerals beneath the seafloor surface, see Figure 1.2.



(a) (b)

Figure 1.2: In sub–bottom profiling, an acoustic signal is directed into the seafloor bottom. The receiver records the energy that is reflected by different layers beneath the surface. Illustration of seismic reflection profiling [57] (a). Chirp sub–bottom profile of sedimentary layers in Bear Lake, Idaho-Utah, from data collected September 2002 (USGS Open-File Report 03-150) [66] (b).

SBP systems represent the mapping of the geographical strata of the seabed. This mapping is meaningful for several applications such as the investigation for the solidity of the seabed to construct oil platforms.

Water–column data represent another type of reflections which are used for generating global

information nets based on pre–fetched seafloor characteristics. For instance, if gas seeps are detected a net for all seep holes is registered and a water column for every location is generated to record the reflections of gas particles that flow from the seafloor surface upwards, see Figure 1.3.

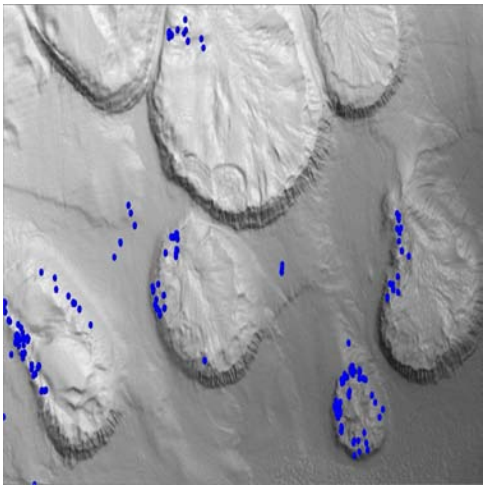


Figure 1.3: A perspective of the seafloor showing preliminary results of gas seeps detected by NOAA Ship Okeanos Explorer multibeam sonar in vicinity of Biloxi Dome in Northern Gulf of Mexico. Gas seep locations are shown as blue dots and are overlaid on the seafloor bathymetry that was collected [56].

Regarding terrain data, the typical acquisition is presented using a remote sensing technique which provides us with information without any contact requirements. In this context, one can refer to the aerial sensors technologies as a new trend for earth sciences including height fields and agricultural fields. Several instruments are available for remote sensing data acquisition using satellite, aircraft, spacecraft, and ships.

Such instruments are dedicated to *geographic information systems (GIS)* that are concerned with geographical information at an earth's surface location which is attached to the relative shapes of features and characteristics. *GIS* tackles such information as a set of layers including the required data to fulfill the spatial analyst questionnaire. These layers must contain at least one geo–layer for determining the geographical location, i.e., longitude/latitude, and another attribute–layer to incorporate some features of that indicated location. Hence, a *Digital Elevation Model (DEM)* arises as one of the most commonly used models to register the height feature of either the terrain surface alone or considering objects on the terrain as in a *Digital Surface Model (DSM)*. Hence,

satellite and aerial imagery provide natural color aerial photography for a variety of phenomena of an area.

1.3 Data Representation

Geo–spatial data are given in multiple formats of information that are acquired by different instruments. Basically, such data can be organized in a set of layers that are composed to represent the final geo–spatial locations. Every layer captures a specific geographical information with a different given resolution which means that the layers could not be mapped directly to one map. Thus, using such layers as input, our work is based on a framework of information layers where those layers are caching different information but are sampled over one parameter plane.

Our framework presents a definition for the parameter plane as $S \in \mathbb{R}^2$ where S is an abstract set of samples. Every sample is defined as $\{x_i;\ i = 0, ..., n-1\}$ where n is the number of samples's information and let $L \in \mathbb{R}^2$ where L represents a set of longitude/latitude coordinates, a given mapping projection function $\Pi : \mathbb{R}^2 \mapsto \mathbb{R}^2$ can be defined as: $\Pi(long, lat) = (x_0, x_1)$ where $long, lat \in L$, i.e., the first two tuples identify the geometry coordinates layer in $\mathbb{R}^2$ space. We consider the rest of the tuples $\{x_i;\ i = 2, ..., n\}$ attributes of the geometry location (x_0, x_1).

In this work, our framework accepts three layers over the parameter plane. The first layer is for heightfield including both height values in terrain data and depth values in bathymetry data. The second layer is for color information from satellite images in terrain data or for backscatter intensity in bathymetry data. In here it is worth mentioning that the original dataset is given in a gridded format or an ungridded format. For ungridded data resampling and interpolation processes are requires to make the input of this layer. As the scatter data interpolation technique is part of our system the original dataset is used directly as input. Another data type for gas seeps, i.e., water–column data, is considered in which our goal is visualizing the density distribution of the gas.

1.4 Data Visualization Technologies

Once the data are gathered, a bunch of issues arise in two main aspects: On–demand data loading at optimal frame rates and detecting the required storage capacity.

The task of heightfield visualization (or terrain rendering) includes the visual challenges of heightfields representation that concern mainly producing high–quality of renderings where it is important to obtain interaction at high frame rates. If the heightfield is rendered using triangulation the geometries that need to be rendered should be reduced without losing the high–quality rendering. In here the reduction can be restricted by using a tolerance of a screen–space error. In addition an optimization for the shapes of the triangles is required to eliminate the possible artifacts. However, the main challenge for most heightfield navigation systems is the huge amount of data in which each sample could have additional attributes.

Existing systems can be classified into two categories: grid–based and mesh–based approaches. Grid–based approaches typically operate on regular quadrilateral grids, which allow for an efficient data management, as both the parameterization and the neighborhood information are given implicitly. All information is stored in form of 2D array–like data structure. Such structures map well to textures, which makes the grid–based approaches both time– and memory–efficient. As the amount of data is too large to meet the memory constraints, adaptive grids have been employed that are generated by applying view–dependency to level–of–detail hierarchies. Handling transitions between different levels becomes the main issue of these so–called semi–regular approaches. Common schemes use quadtrees (e.g. [60]) or bintrees (e.g. [20]). A comprehensive survey of these methods and recent advances is given by Pajarola and Gobbetti [61]. However, all these visualizations are restricted to samples at regular positions.

Mesh–based approaches, typically, allow for sample points at any location and for any connectivity between the samples. Thus, they are highly flexible and amend to any feature. Irregular meshes can produce a minimum complexity model when applying view–dependency. Common schemes are based on progressive meshes (e.g. [37]), triangular irregular nets (e.g. [13]), and multi–tessellation (e.g. [27]). However, one needs to explicitly store the locations of the points in the parameter plane and the connectivity (or neighborhoods) and the information should be captured efficiently in which it can directly be mapped to graphics hardware. Also, the generation of level–of–detail hierarchies and handling of adaptive refinement requires more effort.

Our contribution in the field of geo–spatial data visualization is achieved by exploiting several techniques to introduce interactive visualization and exploration system. The main contribution is

to visualize heightfields with high quality at highly interactive frame rates. The view–dependency is integrated in our system to minimize the number of rendered triangles efficiently. The heightfield is also cached in pyramids for LOD representation. As we propose a hybrid approach such that both mesh and grid are used our main contribution is decoupling the mesh and the heightfield representations. One of the major features of our approach is that the generated triangular mesh has fixed structure that is computed depending on an efficient view–dependent layout. Such computations concern optimal triangles in size and shape when the triangles are mapped to the screen space. Thus, the mesh is sent to the GPU once and it does not change its structure at rendering time. Furthermore, our approach has another main feature which is the performance such that our algorithm is fast if it is compared to other approaches with similar quality. In comparison to the approaches with similar frame rates we are producing lower screen-space error and the shapes of the triangles are improved.

Our work is also presented to explore and visualize terrain data and bathymetry data interactively. Thereby this system allows:

- The visualization of an additional attribute such that the color mapping of the surface is integrated with respect to a given texture at almost no cost. Regarding the multibeam data, we are the first to integrate a volume visualization of water columns in the heightfield visualization framework.
- The interactive exploration of the data.
- The uncertainty visualization of the height values.

1.5 Outline

In this work, we propose a hybrid approach that uses both meshes and grids by decoupling the mesh and the heightfield representation. A fixed mesh structure is computed such that the triangles when mapped to screen space have optimal size and shape. This mesh is kept during rendering and never changes its structure. The only modification that the mesh undergoes is the change of the height values at its vertices. The height value updates are governed by the underlying heightfield. The heightfield, however, is not stored in the mesh structure, but in a multiresolution hierarchy of

regular grids (stored as textures). During rendering, the proper level of detail is determined by the size of the triangle.

The proposed approach involves a pre–processing step that generates the optimized mesh structure and the grid hierarchy. Those preparation steps are described in Chapter 3 and Chapter 4. As the mesh generation only depends on the ratio of the triangle size to the pixel size of the display screen, it can be created once, stored in a file, and generally used for all data sets. The grid hierarchy needs to be created once for each data set. During runtime, the prepared structures can be used to efficiently and accurately visualize the heightfield data. The runtime steps and the description of the implementation on the graphics processing unit (GPU) are explained in Chapter 5. Our system also supports a tiling concept that splits the data into tiles of manageable size and loads the proper tiles when requested. Our results and experiments are presented and discussed in Chapter 6.

Furthermore, we present our techniques for the visualization of terrain data in Chapter 7 and for interactive visual analysis of bathymetric data in Chapter 8.

Several visualization techniques are presented and show the ability of our algorithm to exploit new graphics card features. The implementation results of the presented visualization techniques show the desired performance. Furthermore, our work can visualize interactively different data of the same seafloor region in one rendering pass in contrast to existing systems such as the Fledermaus software. The system design of this work is shown in Figure 1.4.

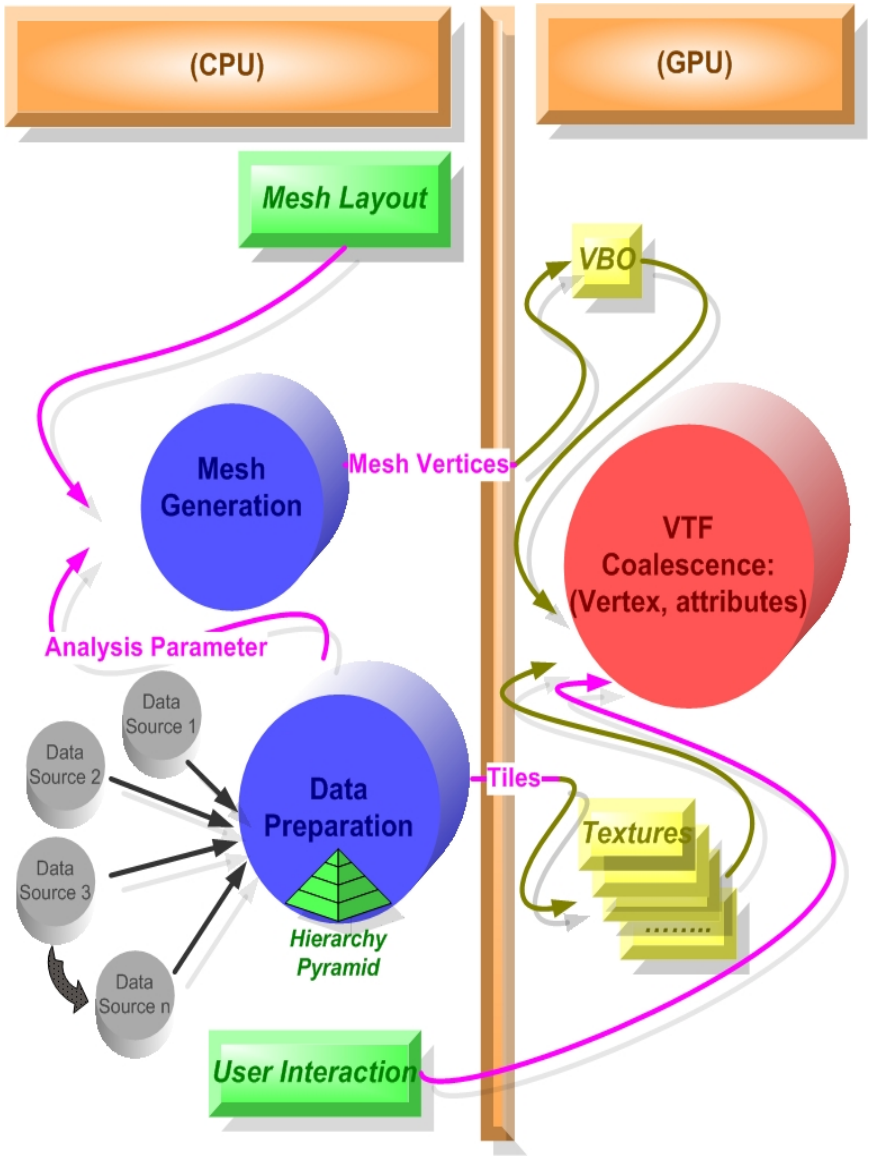


Figure 1.4: System design represented in two major steps exploiting both CPU and GPU features. Mesh generation and data processes in blue color are done in preprocessing step. Vertex texture fetching in red color is achieved on GPU at run–time step. Mesh vertices are loaded in Vertex Buffer Object (VBO) and textures are cached in texture memory, both in yellow color, are transmitted to GPU memory.

Chapter 2

Related Work

This section focuses on the related approaches for heightfield visualization in both surface approximation and data management fields. Heightfield visualization deals with the 3D rendering of data over a 2D parameter plane using surface models. Sometimes this constellation is also referred to as a 2.5D rendering. In literature, two main streams of work can be reported, namely the ones operating with irregular surface meshes and the ones operating over a regular grid structure.

In grid–based approaches a grid is considered in a given space and the heightfield is sampled equidistant. The sample points identify positions that are called nodes. These nodes are the guide of determining the location of the vertices that are used to generate connected polygonal faces for surface adaptation. To allow the flexible sampling at any location of the targeted surface mesh–based approaches presented solutions for generating polygonal meshes such that the sample points are not building a regular pattern, i.e., the mesh is irregular. Irregular means that the mesh has faces of varying number of edges and/or vertices with changing valence. Both grid–based approaches and mesh–based approaches use the concept of view–dependency such that the amount of rendered geometry is adapted to the size of the geometric primitives (typically triangles) when being projected onto the screen.

The surface renderings are typically based on rendering polygons, i.e., using an explicit surface representation. In the ray-casting approaches, the surface is rendered by intersecting viewing rays with the implicitly defined surface. The implicit representation is given in form of the result of an interpolation method applied to the heightfield. For heightfields over regular grids, bilinear interpolation is being used for a piecewise bilinear surface representation.

2.1 Surface Approximation

Most terrain visualization algorithms are based on triangular meshes for approximating a surface optimally as it had been examined by Garland et al. [29] and Rossignac et al. [69]. The optimization of triangular meshes, e.g. [36, 75], is distinguished amongst a variety of methods. Regular, irregular, and semi–regular structures are the primary methods in this context.

2.1.1 Regular Methods

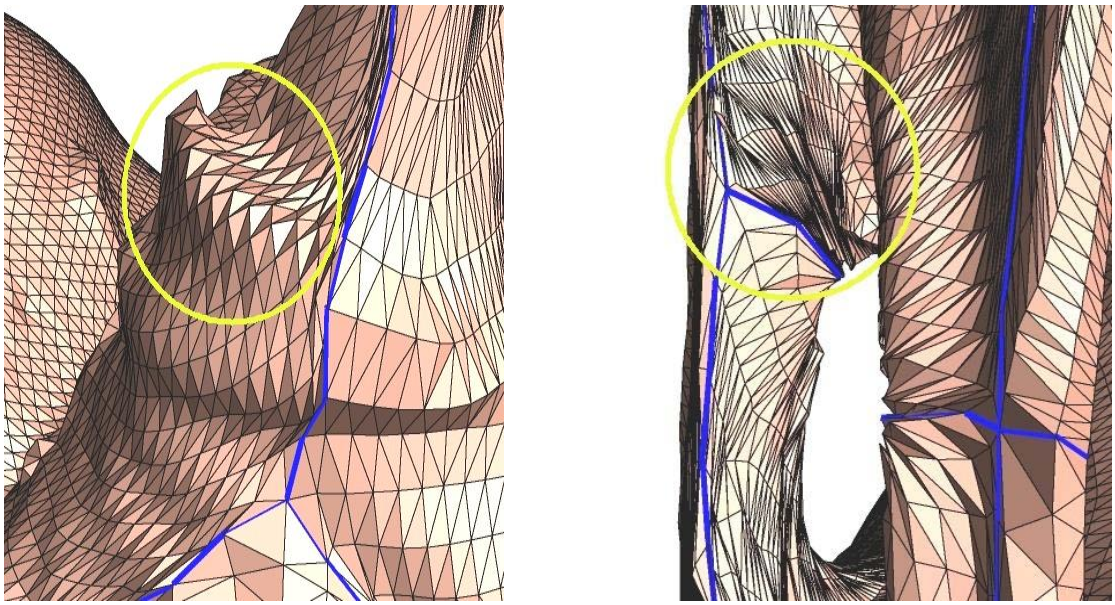
The foremost structure of regular meshes was named a quadtree by Finkel and Bentley [25]. Essentially, the advantages of quadtrees are exploited in an endeavor embedding the hierarchical representation. Pajarola [60] attempted to find a solution for displaying the entire targeted scale of heightfield by adding some extra triangles. This produces a difficult way to convert simple 2D quadtree to the 3D restricted pattern as it was presented in [76].

Lindstrom [48] presented a restriction using a dependency graph that is built on a regular grid. Within this graph a set of vertices are created where each vertex depends on two vertices of the same or the next higher level in the hierarchical quadtree. The main advantage of this dependency is avoiding a common issue of triangulation subdivision which is cracks. Cracks arise when one of two triangles that share a common edge is subdivided by inserting a vertex on the edge and if the newly inserted vertex does not happen to lie on the edge. The newly inserted vertex is, then, referred to as a hanging node or a T-junction, see Figure 2.3. Thus, the used dependency [48] prevents cracks by consistently restricting the selected points to attain a matching triangulation. Pajarola [60] used the dependency graph to construct the restricted quadtree. Here, the points selection is determined according to an error approximation within a threshold. The approximation error is define by the vertical distance between the vertices of merged qaudtree nodes to the average elevation of the vertices of the related parent such that the vertices of each node are its corners.

Another view on the advantages of regular meshes are presented by Gu et al. [31] first, the sampling where no vertex indices issue rise; second, for more efficient reduction of the geometry rendering. The cumbersome process at run–time is alleviated via a contribution of parameterized maps that have suitable structures for involving the graphical parallelism virtue. In this approach the surface is cut into a set of topological disks. For every disk a 2D image, that is called a 2D map with

an arbitrary shape, is defined to capture the geometry. The smooth matching of the reconstructed surface necessitates using the parametrization on the boundary of every image. Hence, cracks are avoided.

However, Gu et al.'s algorithm met a critical situation of sharp features approximation, see Figure 2.1, and hence re–meshing approaches are required for tackling the disturbing visual effects. Such visual effects are often difficult to be integrated in the main algorithm without producing extra objects.



(a) (b)

Figure 2.1: Sharp features issue presents one of the obstacles of regular mesh–based approaches as described in Gu et al. [31] (a). Re–meshing approach is one of the possible solutions for sharp features issue (b).

Lee et al. [46] used parameterized maps (e.g. height fields of terrain) in association with regular control meshes for achieving efficient simplifications of displaced surfaces, see Figure 2.2.

The undesired visual effect is avoided in our work by optimizing the size of each triangle in preprocessing stage. This optimization is left to a view–dependent layout that uses a screen resolution as an estimation factor of sufficiently small triangle size. Moreover, efforts spent on restrictions regarding cracks and T–junctions issues, see Figure 2.3, are not necessary in the pre–created static

(a) (b) (c)

Figure 2.2: Control meshes (a) are involved as a tool over a given smooth surface (b) to produce the more detailed surface (c) that is able to be stored in a compacted parameterized map.

mesh.

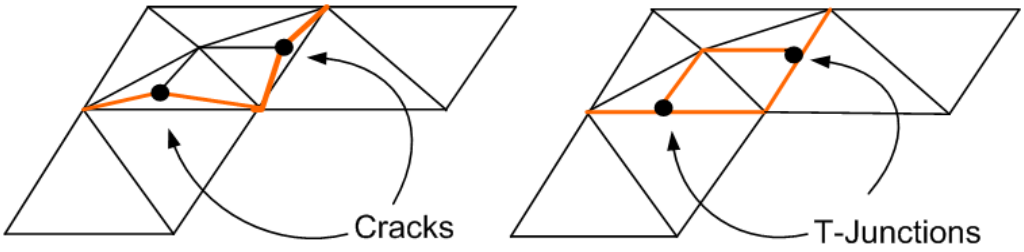


Figure 2.3: The common issues of regular meshes are represented as cracks (left), and T–Junction (right).

Real–time Optimally Adapting Meshes (ROAM) have been presented by Mark Duchaineau et al. [20] and uses two priority queues to drive split and merge operations; thus, holding the continuous triangulation was accomplished. The split/merge operations start with a regular mesh and are based on a binary tree with diodes of right triangles (indicated as A and B in Figure 2.4) as a core where the fine level of the triangular mesh L_i results from the next–coarser level L_{i-1} using a trivial base bisection process, see Figure 2.4.

Therefore, two priority queues were utilized to optimize the triangular mesh and to do this task an error metric is computed for each triangle's vertex within a bounding volume wedge, as shown in Figure 2.5. The wedge depends on the comparison between the height value of the screen–space

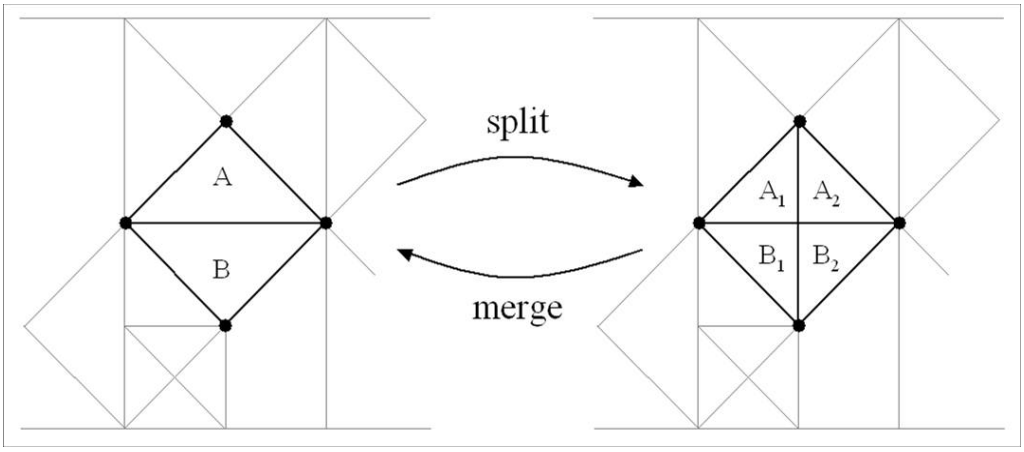


Figure 2.4: Split and merge operations over a binary tree, where A and B are the basic diamonds in all levels [20].

projection and the approximated value of non–optimized triangular mesh.

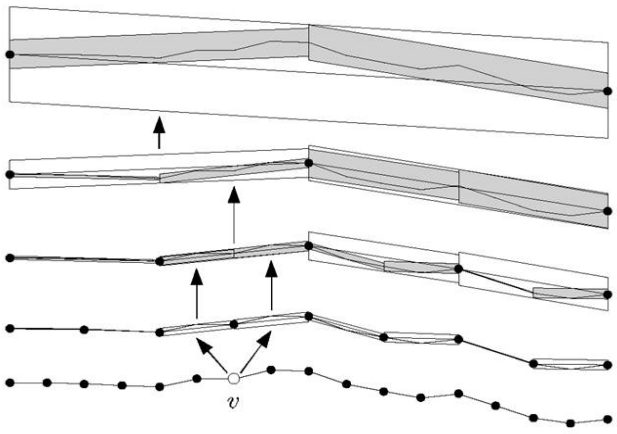


Figure 2.5: ROAM wedges principle [20]: each vertex v has a position correction geometrically through a comparison of triangular mesh approximation and screen–space projection.

This process grants the ROAM algorithm the smooth transitions between adjacent triangles of different levels. Thus, ROAM involves advantages such as frame coherence, that was introduced earlier in [73], and least number of split/merge operations against optimal mesh albeit the per–vertex memory requirement is still expensive in the comparison with alike approaches [70]. Moreover, the aggregate number of triangles is still beyond the planned optimized mesh due to the splitting mission.

Such a mesh structure is a simple enough to present fast rendering but it is worth mentioning

that in this approach the triangles are sent to the graphics card one by one instead of using more efficient structure in which the triangles can be sent once to the graphics card. Our approach in contrast has optimized the number of the well–shaped triangles and send them only once to the graphics card.

2.1.2 Irregular and Semi–Regular Methods

Irregular triangulation showed advantages of mesh adaptation to avert extra triangles. Irregular meshes were first used in Triangulated Irregular Nets (TIN) that were introduced by Peucker et al. [64]. Isenburg et al. [41] implemented TIN using a Delaunay triangulation [13, 34], which required managing the memory storage efficiently. In semi–regular surface representations a variety of arbitrary regular polygons are utilized where the polygons vary in shapes to produce semi–regular meshes with a collection of regular and irregular, i.e., close to regular polygons. Constructing semi–regular meshes are typically evaluated based on a parameterization map that simplifies the remeshing stages and facilitates implementing smooth level of detail transitions. The most recent survey of semi–regular approaches was shown by Pajarola and Gobbetti in [61].

While the irregular meshes minimize the amount of primitives that need to be rendered, the required approximation error is still concerned with the complexity of data management which has to produce accurate results but not expensive CPU usage. Therefore, a bunch of techniques were used to validate the desired advantages of irregular and semi–regular patterns.

2.2 Levels Of Detail (LOD) Techniques

View–dependency builds upon the concept of a level–of–detail (LOD) representation, where the underlying heightfield is represented at different resolutions. Thus, fulfilling the goal of producing a triangular mesh with required detail. Basically, the view–dependent concept is that for a given viewing point the distant objects are rendered in a coarser representation than objects close to viewing point. Hence, different levels of resolution of the rendered scene are required where one can store them in a hierarchy for LOD representation. The LOD concept is the fundamental medium to sustain the interaction over geoscience data exploration as introduced by Clark [10] and Luebke [15].

A variety of applications borrowed LOD as intrinsic part of their algorithm [3,5,6,9,35,45,54,55, 62,67,74,83]. Lindstrom et al. [48] utilized an LOD algorithm that is based on bottom–up/top–down strategies to alleviate the complexity of a terrain model by using one large triangle strip. While the bottom–up traverse of quadtree is made to produce fine triangles for generating more detailed terrain the top–down traverse is used to merge triangles if the reduction of the accuracy is not noticeable. Thereby, a decision to subdivide or to merge every mesh's node with its adjacent nodes is obtained. This LOD used a screen–space error metric to add more detail until the projected distance between adjacent vertices becomes smaller than a predefined pixel threshold. However, the large triangle strip that made the terrain rendering fast and easy meets difficulties as all levels of detail are created from one quadtree with a complex data structure.

Moreover, along the continuous changing of the viewing pose, the bottom–up strategy suffers from the perpetual tackling of the visual popping issue which occurs when changing the level of detail. Thus, a fast geo–morphing algorithm, which uses smoothly animated transitions, is needed as introduced by Hoppe [37, 38]. Surveys of LOD approaches that are based on regular and semi–regular geometrical data structures are introduced by [26, 61]. In addition, Tierny et al. [79] made use of LOD which based on the topological and geometrical analysis aspects to construct the desired 3D meshes.

2.2.1 LOD Transitions Issues

The introduced solutions of LOD strategies are still not enough to support the desired results of height fields rendering at run–time. Lindstrom et al. [47] proposed a solution, which is called Continuous Levels of Detail (CLOD), to smooth the transition between the used levels. Basically, the CLOD algorithm tackles the common issues of LOD based heightfield rendering in order to truncate the extra objects that result from granularity procedure. Furthermore, to avoid the exhaustive check for billion of vertices cases extra constraints were involved to focus on each patch solely. Indeed the patches borders are tackled for both: fast appropriate level of detail detection at run–time as well as the fidelity matching between adjacent patches whereupon the visual popping effects are eliminated between the consecutive frames just like in Hoppe's work [37, 38].

LOD and CLOD have yet to incorporate more improvements for avoiding the complexity at

run–time.

To simplify the rendering, Pajarola [60] captured the heightfield in a long triangle strip instead of organizing it in a tree that has to be traversed during rendering. However, this necessitates adding a lot of complexity to the computational aspect.

In comparison, the independency between surface approximation and heightfield representations is one of the major advantages of our approach in which it contributes to split the complexity of mesh and heightfield issues, and consequently a simplification of the interactive rendering process at run–time is achieved.

All view–dependent LOD algorithms focus on the design of the hierarchical structure and the way of selecting the LOD efficiently. Erikson et al. [24] investigated a way of multiple objects scene performing a hierarchical design of the scene globally where each object has its own traditional LOD structure. This approach gained the efficiency in which its static structure can be easily delivered to the rendering engine and the ability of choosing between high quality or a constant frame–rate.

However, the decision of switching between LODs is still a big challenge for the rendering algorithms as they should invoke either a simple solution for LOD switching and thus visual popping, T–junctions, and cracks issues will result, or a complex one that requires respectable efforts to avoid any probable bottlenecks cases.

Moreover, the work of Pajarola [60] employed as well LOD to various patches of the visible quota of the partitioned scene. Thus, a dynamic scene concept is implemented to the targeted data. In this approach, the heightfield is stored on hard disk as one file. Accordingly, a complex searching technique is used to retrieve on-demand data.

On the contrary, our approach prevents such time wasting by caching heightfield in textures that represent an immediate mediator for data rendering and data exploration.

2.3 Simplification Approaches

View–dependency was proposed as a fundamental feature in the hierarchical LOD design to simplify the complexity of rendered geometry. Basically, one of the most common works in this field was the imposters approach that was introduced by Maciel et al. [53]. They attempt to involve several factors underlying the view–dependency as intrinsic measurement to reduce the objects' number of

the rendered scene at a constant frame rate and in an adequate image quality.

The entire scene in the imposters approach is a compound of a set of objects which each has an LOD in association with texture maps of pre–defined viewpoints. The finite number of viewpoints are captured via a hemisphere around the entire objects of the scene. This approach depends on the orthographic projection technique. Thus, producing objects' images and imposters were achieved at each viewing direction (i.e. at each sample point). The accuracy was based on an image comparison and used as an important parameter to build the hierarchical LOD design.

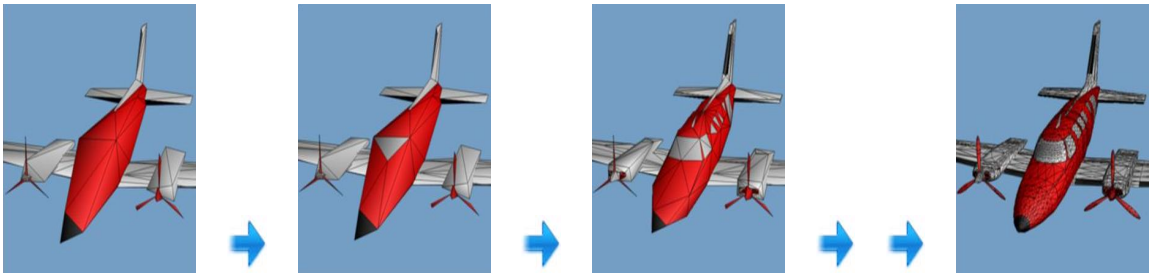
Losasso and Hoppe [50] showed the reliability of rendering large terrain in a given viewpoint upon a group of predefined clipmaps. The clipmaps are created simply utilizing the viewing distance as a world–space parameter so that the LOD can be selected over nested regular grids. Depending on the computation of the screen–space error metric introduced by Cohen [11] Dick et al. [19] create a threshold in order to choose the appropriate LOD at a specific view–port scene.

Basically, the simplification techniques that have been used in different approaches [1, 12, 15, 16, 23, 33, 40, 42, 43, 45, 51, 52, 55, 74, 80, 82] focus on irregular meshes to achieve the desired approximation without losing neither the targeted frame rate nor the best rendering quality. Hence, an auxiliary aid can be considered (e.g. texture objects such as in [4, 7, 53, 71, 77, 81]) to minimize the number of rendered mesh vertices.

With quadric error metrics validation, Garland and Heckbert [30] achieved worthy simplification of an irregular mesh model. As in the regular mesh approaches, semi–regular meshes exploit the parameterized maps to gain the desired adjustable surfaces (e.g. Guskov et al. [32]). Arbitrary meshes are aimed at in most exploration algorithms, e.g. [17, 22, 31]. These meshes use an acceptable error threshold for the coarse mesh pattern in order to speed up the final rendering model.

Progressive Meshes (PM) presented by Hoppe [37] describe a representation of an arbitrary mesh $\widehat{M}$ that results from coarser mesh M^0 along with a set of n detail records of information regarding the collapse operation of vertices, see Figure 2.6. PM could be achieved using one mesh transformation and its inverse instead of applying three mesh transformations: collapse, split, and swap were presented by Hoppe et al. [39] to minimize an energy function. For efficient visualization, PM representation used geomorphs to avoid the visual discontinuities between followed meshes. Afterward, an optimization of PM was introduced in Hoppe's work [40] to prove that the superiority

of irregular meshes can be maintained if extra criteria based on viewing (i.e. view frustum, surface orientation, and screen–space geometric error) are considered over the standard PM transformations. Moreover, Sander et al. [71] developed the standard PM with the share of a set of textures that are organized in a pre–defined atlas. Hence, the given mesh is partitioned into charts which are filled by 2D textures through an appropriate mapping function.



(a) (b) (c) (d)

Figure 2.6: Progressive mesh representation (PM) from a base mesh (a) to the finest mesh (d) [37].

2.4 Mip–mapping Technique

The motivation of several approaches highlighted caching media as an intrinsic challenge of out–of–core issues which is now mainly based on GPU architecture. Accordingly, texture objects are being used excessively for offering a straightforward mechanism to directly process the heightfield values on GPU with interactive manipulation ability. As a consequence, Losasso and Hoppe [50] organized the terrain in a hierarchical pyramid of textures with trivial power–of–two decimated maps that followed the LOD principle. Each map is an indicator for one level of detail that is generated as a uniform 2D grid that facilitates the fast rendering of the triangular mesh. The mesh vertices are indexed and cached in a strip that is based on an efficient structure for supporting GPU performance. To render the triangular mesh, a nesting of the pre–constructed maps is applied according to a given view–point, see Figure 2.7. However, to avoid the visual discontinuity effect, as in Figure 2.8 (a), between adjacent nested levels' borders, each map has to consider blending on both aspects: geometries, see Figure 2.8 (b), and textures, see Figure 2.8 (c), by using the L–Shape morphing method. The calculation of a view–dependent parameter is the required step to morph

vertices and texels on GPU. Yet, the algorithm suffers from extra triangles generation especially in flat areas due to the extreme utilization of uniform primitives through the error analysis step.

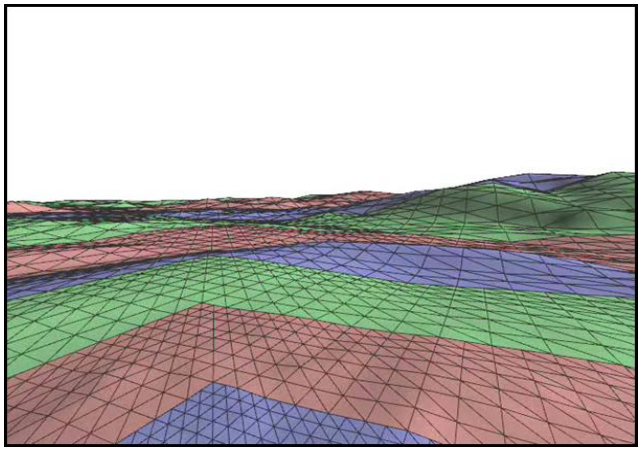
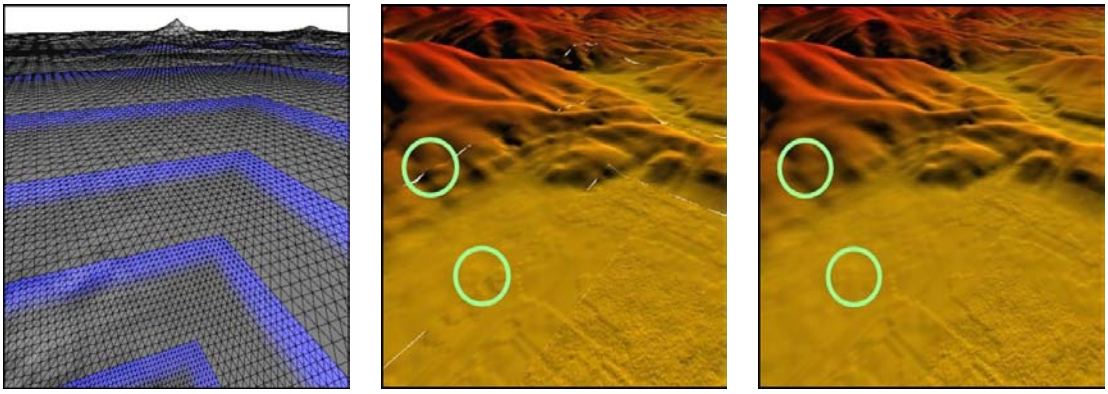


Figure 2.7: Clipmaps algorithm representing the advantages of textures on GPU with nested grides in a given view point [50].

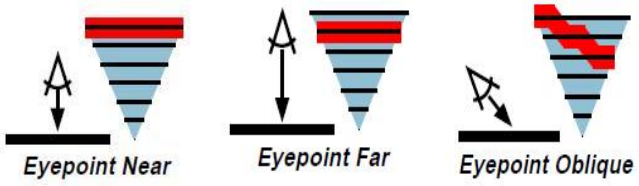


(a) (b) (c)

Figure 2.8: The problem of visual effects between levels (blue regions) (a). The solution for geometry discontinuity (b) is achieved by texture blending that uses the L–Shape method for a morphing process (c) [50].

Improvements of the clipmaps algorithm were defined as a set of heuristic parameters in the hybrid approach that was introduced by Amman et al. [2]. The major parameter is a viewing threshold to switch between a rendering using triangular mesh approximation and a rendering using a ray–casting approach.

Basically, such mapping concept had already been provoked by Tanner et al. [77] in a dynamic texture management system that addressed a collection of solutions for the common issues of earth surface rendering in real–time. Hence, the so–called mipmap textures were organized in a pre–defined hierarchical pyramid.



(a) (b) (c)

Figure 2.9: Different locations of view point over mipmap textures pyramid.

By considering different cases of viewing–point location including near, far, and oblique, see Figure 2.9, view–dependency technique was exploited to minimize the caching requirements as the number of the potential accessed texels is reduced when the clipping process truncates the pyramid according to a logical limit of memory size. Boer [7] exploited the standard mipmapping method to achieve the rendering of a fast 3–dimensional huge number of objects. However, the storage capacity is still insufficient against the enormous data in loading and updating processes even with the aid of GPU incorporations.

2.5 Tiling Technique

Tiling is the process of splitting terrain data to $2D$ patches. Those patches are similar and able to be reconnected with smooth appearance. Therefore, mipmaping approaches have been used along with the advantages of tiling technique where they could tackle each map independently. Hence, out–of–core data are made on–demand through spreading any huge size of heightfield maps over tiles that require then merely the algorithm's authorization to stich tiles in a seamless visual effect of continuous surface rendering at the real–time of data explorations.

Splitting terrains into independent tiles is used in most region–based approaches (e.g. the virtual GIS system introduced by Koller et al. [44]) to serve out–of–core data issues. The chunked LOD algorithm in [81] was introduced in pursuit of organizing the data in a set of LODs and hold them in

static textures to be reused for similar viewpoints. Various common techniques are also used to settle issues such as visual popping. Therefore, GPU computations were integrated in the context of using view–dependency to reduce the computations efforts over CPU and thus supporting the streaming of out–of–core data. However, the great deal of managing the data in a quadtree structure addressed the main challenge in the chunked LOD algorithm, see Figure 2.10.

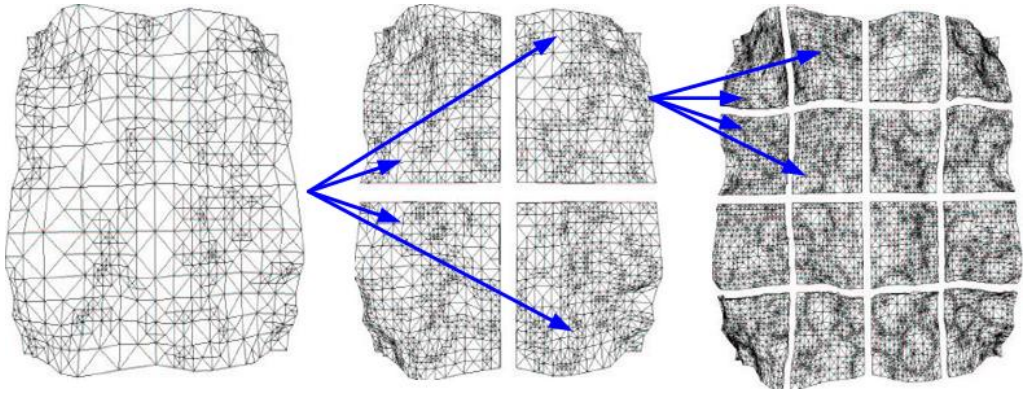


Figure 2.10: Chunked LOD algorithm is embodied in a hierarchy based on quadtree structure [81].

Bösch et al. in their RASTeR approach [8] introduced trade–offs of several common techniques such as the bintree, the hierarchical quadtree, and the view–dependent error metric computations for tiling terrains. They first exploit the contribution of the GPU features and second enable out–of–core data streaming. The fulfilling is achieved using two new concepts K–patches and M–blocks. K–patches stem from the generation of a bintree semi–regular triangulation as a set of triangle clusters that is independent from heightfield structure. M–blocks are created as a regular heightfield quadtree grid. The correlation between K–patches and M–blocks structures, see Figure 2.11, allows for the integration of GPU computations to use texture objects efficiently. However, the tackling of the T–junction issue between patches forced the approach to propagate undesirable extra triangles. Hence, the proposed solution required a consideration of view–dependent error metric computations.

Dick et al. [19] incorporated a GPU coding method to achieve tiling and to avoid the increment of triangles number and to solve mainly the problem of data streaming for hard disk and memory accesses. Essentially, the method is based on creating different strips of restricted quadtree meshes in a compression model that provides a 8–9 factor of reduction in the memory requirements. In

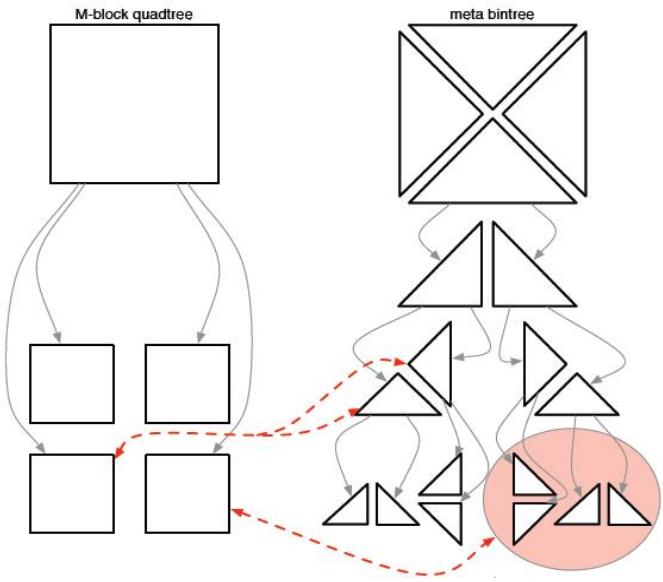


Figure 2.11: The matching between K–patches bintree structure and M–blocks quadtree heightfield structure.

a pre–processing step, the heightfield is organized in a hierarchical pyramid that simply contains power–of–two LODs. The entire domain is created by utilizing a tiling technique where each tile has a size of 513×513. However, each tile is meshed separately and thus skirts around the boundary in each tile are implemented to prevent the occurrence of T–junction cases, see Figure 2.12.

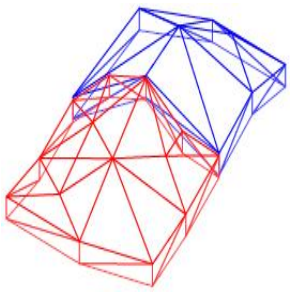


Figure 2.12: T–junction issue is solved using boundary skirts [19].

2.6 Ray Casting Approaches

The main advantage of ray casting approaches, that is the high quality rendering, becomes the major challenge for the interactive rendering of terrain data sets. Due to the need of huge efforts for tackling an accurate rendering and an interactive visualization of large terrain data sets the GPU becomes the means to accelerate the ray casting algorithms.

Recently Dick et. al. [18] had realized that utilizing an efficient LOD simplifies the combination of a view–dependency technique and the ray–casting algorithm for sampling the terrain that is represented as a set of tiled textures. The rays traverse every tile's bounding box where the rendering is performed for the back faces of the tile. They used back faces rendering to avoid dealing with special cases when the viewer stands inside the bounding box. All tiles are rendered in front–to–back order. Using textures for heightfield tiles allows them to perform the ray casting algorithm on the GPU. The intersections of the ray casting determine the sampling locations of the texture of the tile. This technique still suffers from the redundancy of the ray casting steps. Tevs et al. [78] introduced as well a method for GPU ray–casting intersections over a view–dependent mesh–based surface rendering. However, developments were added by Dick et al. [18] to implement terrain tiling in which a large–scale validation of data was performed.

To optimize the ray casting approach Dummer [21] presents a Cone Step Mapping (CSM) technique based on pre–computed circular cones for performing the heightfield intersections. The maximum angle of those cones guarantes that there will be no intersections with the heightfield. A cone is built for every pixel where the tip of the cone is touching the heightfield. Thus, the ray can skip several steps depending on the cone's size. The main disadvantage of CSM is using a huge storage of the hard disk. Therefore, Policarpo and Oliveira [65] replace the CSM technique by the Relaxed Cone Stepping (RCS) technique to relax the constrains of every cone using a cone map. The cone map is a depth map that caches the width/height ratios of the cones. The idea of RCS is that the radius of the cone is extended as long as the ray inside the cone is not intersecting the heightfield more than once.

However, in all ray casting approaches the complexity is either in the computations or in the huge storage due to the extreme dependency of the heightfield intersections. Such a problem is avoided in our work as we made the heightfield optimization independent from the surface approximation.

The ray casting technique is exploited in our work in the context of virtual mapping to generate an optimized mesh in both the rendering quality and the storage requirements. The main features of our approach are the high image quality and the high performance for heightfield rendering and the interactive visualizations.

2.7 Geo-scientific Data Visualization System

To render, visualize, and analyze geoscience data a set of systems are commonly used for terrain data and bathymetry data applications (e.g. Fledermaus software and a set of softwares that are produced by the Computer Aided Resource Information System (CARIS) company). These softwares are very important tools for geologists and hydrographic surveyors to map multibeams data. The software of CARIS is used to create charts and maps for specific hydrographic applications which are beyond the scope of our work and thus we focus on the Fledermaus software.

Fledermaus is a 3D data visualization software for a variety of applications including topographic and bathymetric data analysis and measurements. It provides a sophisticated tool for interactive data preparation, analysis, and presentation. This tool allows the user to add images, vertical imagery, ASCII points and lines, and Electronic Nautical Charts (ENCs), 3D models, Economic and Social Research Institute (ESRI) shape–files to create a nice scene. Users can navigate the scene using a 3–button mouse or a 3D navigation device (3DConnexion Space Navigator) in mono– or stereoscopic mode.

However, there are several limitations in Fledermaus that had been addressed in our work including the following:

I. The total performance is enhanced by transmitting most of the work to the GPU and exploit new graphics card features.

II. Integrating time–sampled bathymetric data (e.g. backscatter, water–column data) in the rendered height/depth scene and supporting interactive visual exploration of the data.

III. Interactive visualization of gas seeps using direct volume rendering.

Chapter 3

Mesh Generation

Our system presents a novel approach based on a decoupling of data management and mesh representation. We handle the two aspects individually and optimize both in preparation steps. At runtime, we bring the components together in an efficient manner.

A triangular mesh is generated relying on a view–dependent layout that is based on perspective projection. Such a virtual projection is presented by Livny et al. [49] but we do not have the persistent mapping at each frame. We achieve the projection merely once and at the view–dependent layout creation which provides one pre–computed mesh. Moreover, to gain a better rendering quality our approach targets equilateral triangles which are optimized due to circular shape of our layout instead of squared ones introduced by Losasso and Hoppe [50]. This chapter presents all requirements of the preprocessing step to generate a constant planar mesh. We show the generation of our view–dependent layout in detail. Depending on this layout we present the steps of creating a $2D$ triangular mesh with well–shaped triangles. Our objective of generating those triangles of (nearly) equilateral shape is to avoid stretched triangles which may lead to visual artifacts when employing linear interpolation, e.g., in the rasterization and shading step.

3.1 View–dependent Layout

Our contribution is targeted to generate a view–dependent spherical model in which a flat mesh–based surface is projected on a spherical surface with unified resolution. Hence, the triangles that

are built using the perspective projection scheme are projected to equilateral triangles of same size in screen space. As our model creation is done with respect to a flat model, it is independent of the heightfield data the created mesh does not need any regeneration at run–time. Thus, the exploration of all regions around viewer location is provided without performing a triangle projection continuously onto screen space.

3.2 Layout Model Creation

In our work we do the virtual projection that is based on perspective projection by building a view–dependent spherical model where the viewer is located at the center of this sphere. Hence, the entire surface of the lower hemisphere is covering all possible views that are surrounding the viewing point when assuming a flat surface. In this model for every view the screen is at the lower hemisphere surface. Thus, the generated triangular mesh of the entire landscape has approximately equilateral triangles when they are projected onto such screens.

To fulfill the layout model creation we suppose that a viewer is located at position V of a given world–coordinate system. The viewer will be located in the center of a virtual sphere in order to use the projection only for the lower hemisphere of the model. Thus, we can generate a set of concentric circles around point S with minimum inclination angle $= 0°$ to maximum inclination angle of $\approx 90°$, see Figure 3.1

The layout model has a parameter plane P that is located at a predefined distance X from the viewer and the projection plane P' is located at a distance x from the viewer, see Figure 3.2. Assuming that the viewer changes his/her viewing direction and that the distance to the plane P stays constant, the plane P is a tangential plane to a sphere with radius z around the viewer.

To mimic the perspective projection via two planes P and P' a set of viewing rays are sent from the view point V at constant spacing in spherical coordinates. Hence, we obtain rays separated by a constant angle $\Delta(\theta)$ in the inclination angle. In the azimuth direction, we have a continuous representation to obtain a circle. Later we sample this circle at certain positions when generating the triangles. Consequently, the rays hit the plane P' at the computed osculation points. Those points identify the concentric circles such that all vertices of the generated triangles lie on the circles

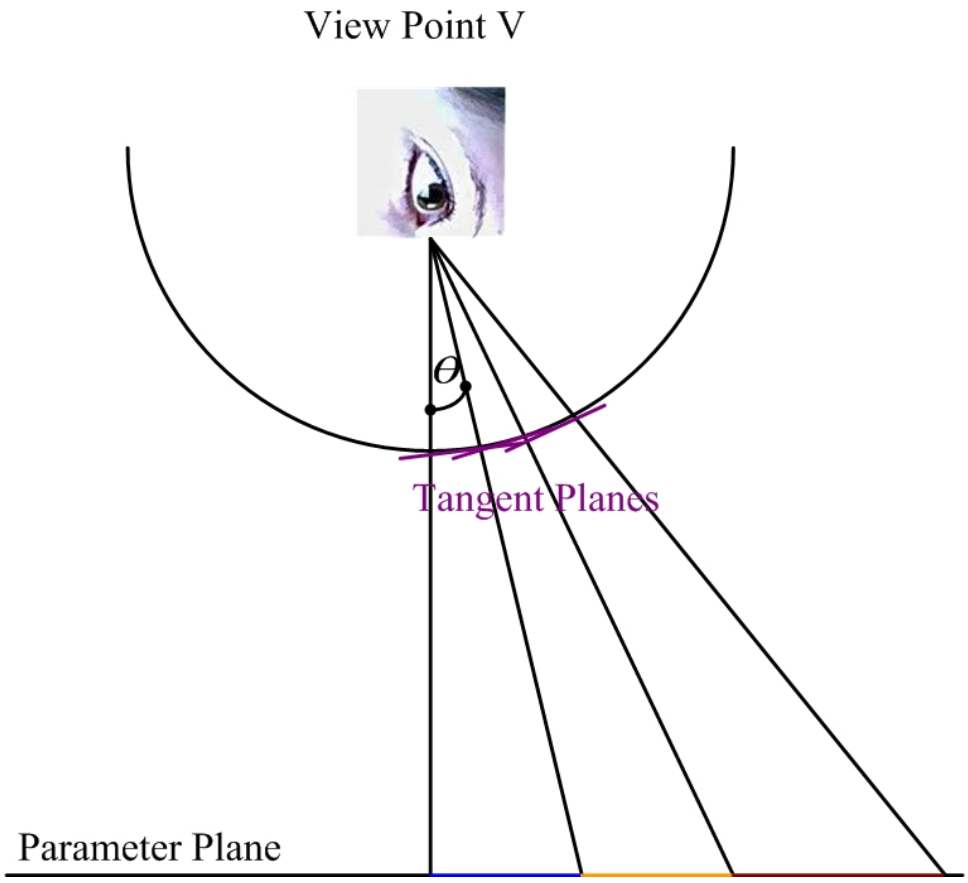


Figure 3.1: The emulation of perspective projection in our design: it starts with emitting viewing rays from a viewpoint V and are separated by constant angles θ.

that are perpendicular to the axis connecting the poles.

We assume that the axis connecting the poles is perpendicular to the parameter plane of the heightfield. When the circles are projected into the projection plane P', they generate concentric circles with radii $z = x \quad tan(\theta)$. Those circles generate rings of increasing size due to the distortion that is the result of the projection of spherical surface. In here, a set of different versions of triangular meshes can be created in advance on the parameter plane P depending on a simple perspective projection through the projection plane P' and according to the requirements of the targeted data set.

Note that the viewing rays are not sampling rays as in [49] but they are only used to produce

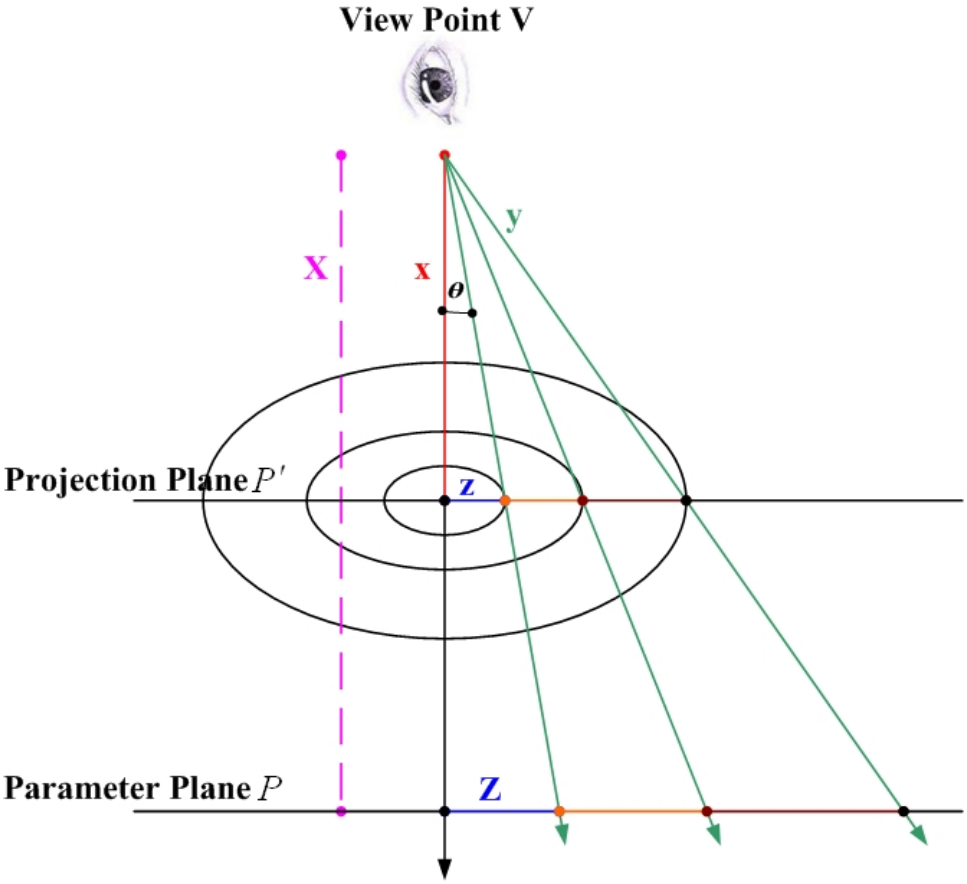


Figure 3.2: The concentric circles on the parameter plane maintain their characteristics when mapped onto the projected (or screen) plane that is parallel to the parameter plane.

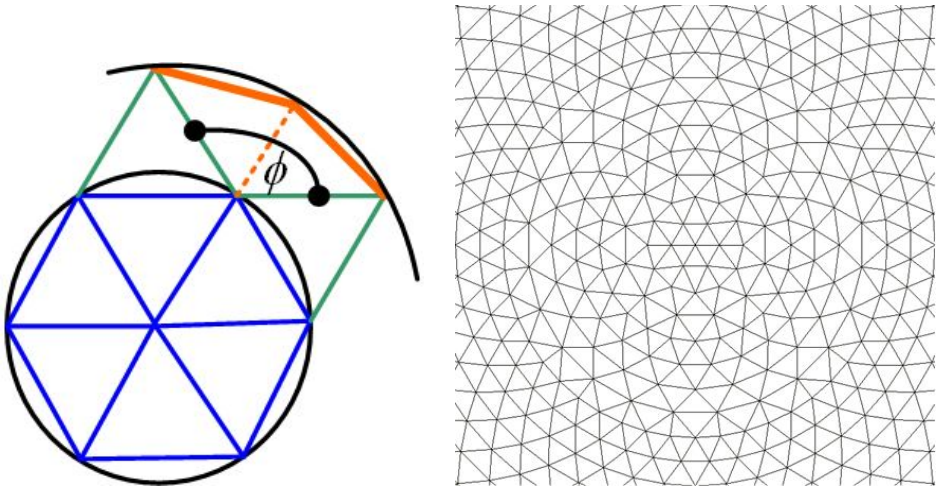
the projected circles of the view–dependent layout. The circumferences of those consecutive circles determine the given samples for our triangulation method.

3.3 Triangular Mesh Generation Using View–dependent Layout

What remains to be done, is to fill the rings with triangles. the rings are formed by two consecutive concentric circles. To generate approximately equilateral triangles, we start with the first degenerate one that consists of only one circle and its midpoint. This degenerate ring can be filled with six equilateral triangles as shown in Figure 3.3 (a) (blue).

The subsequent ring already has fix vertices on the inner circle, while the vertices on the outer circle can still be chosen freely. We use the concept of placing for each edge of the inner circle a vertex on the outer circle that has the same distance to both endpoints of the edge. Those endpoints of the edge are, then, connected to the new vertex on the outer circle to form a new triangle, see Figure 3.3 (a) (green). In this way, we generate the same amount of triangles on the current ring that we had on the preceding ring. As the rings are growing in size, it is necessary to successively increase the number of triangles per ring. Starting from the already generated triangles on the current ring, we use the criterion that a new vertex is created, if the angle ϕ between the edges of two neighboring newly generated triangles is larger than a given threshold λ. For our implementation, we chose $\lambda = 90°$.

Hence, each angle is only split once. The new vertex is placed on the outer circle of the current ring such that it has the same distance to the neighboring vertices on the outer circle. Subsequently, it is connected with its neighboring vertices on the outer circle and the corresponding vertex on the inner circle of the current ring, see Figure 3.3 (a) (orange). The resulting triangular mesh is watertight and uses triangles with a desirable shape, see Figure 3.3 (b).



(a) (b)

Figure 3.3: Triangular mesh construction: The innermost (degenerate) ring is filled with six equilateral triangles (blue) (a). For any other ring, we generate isosceles triangles that match the next–inner ring's triangles (green) and fill the space between those triangles with further triangles (orange). Whether one or two triangles are fitted, depends on an angle criterion: ϕ is checked versus a threshold λ. A watertight mesh with triangles of a desirable shape (b).

3.4 Discussion and Results

The view–dependent layout model guarantees the generation of triangular mesh versions with a set of advantages involving:

(a) The concept of view–dependent principle is integrated easily which allows us to generate the minimal number of rendered primitives.

(b) A watertight triangular mesh which is 100% free from any T–junction and cracks issues as the subdivision scheme is implemented on a 2D mesh version. Thus, no more simplifications are required at run–time process even after heightfield updates.

(c) The generated triangles (i.e. a well–shaped isosceles triangles) are always close to equilateral triangles and not producing long isosceles triangles and not even right triangles. A triangle quality parameter R_Q is computed to proof the achieved improvements against the right triangles that are basically used in quadtrees triangulation approaches.

$$R_Q = \frac{r_i}{r_o} \tag{3.4.1}$$

where r_i denotes the radius of the incircle and r_o the radius of the circumcircle of a triangle. Given a triangle Δ with sides a, b, and c and an area $A(\Delta)$, we derive that

$$\begin{aligned} A(\Delta) &= \frac{1}{4}\sqrt{4a^2b^2 - (a^2+b^2-c^2)^2}\ , \\ r_i &= \frac{2A(\Delta)}{a+b+c}\ , \\ r_o &= \frac{abc}{4A(\Delta)}\ , \end{aligned}$$

and, consequently, that

$$R_Q = \frac{r_i}{r_o} = \frac{8A(\Delta)^2}{(abc)(a+b+c)}\ .$$

The optimal ratio R_Q is obtained for an equilateral triangle with $a = b = c = x$. The optimal ratio is computed to

$$R_Q = \frac{8A(\Delta)^2}{3x^4} = 0.5\ .$$

For the 2D mesh, almost all triangles have a ratio R_Q that goes from $\approx$0.44 to 0.5.

To compare this finding to the state of the art, we consider semi–regular schemes using quadtrees or binary triangle subdivision. All those schemes produce (almost) exclusively right triangles. For a right triangle with $a = b = x$ and $c = \sqrt{2}x$, we obtain the ratio $R_Q \approx 0.41$. Hence, the triangles we generate have a better quality than the ones generated by state–of–the–art semi–regular schemes.

(d) A non–dynamic simple triangular mesh structure allows for an efficient caching of all vertices on GPU merely once for all frames at run–time instead of the continued manipulating of the mesh all over the entire frames as it is in Livny et al. work [49].

Chapter 4

Data Preparation

This chapter is presenting all steps for preparing the given data in a preprocessing stage.

So far, a triangular mesh has been generated independently of the data. It is mapped to the parameter plane P. Let the vertex position in plane P be defined by the x– and y–components. During runtime, the z–component of the mesh's vertices shall be updated using height values of the data.

4.1 Assumption

For our approach, we assume that the heightfield is given over a parameter plane that is sampled equidistantly, i.e., the sample locations form a uniform quadrilateral grid. If this is not the case, the data can be re–sampled to such a structure. As we support a multi–resolution representation, the re–sampling could be done at a high sampling rate.

4.2 Data Sampling

When achieving sampling process, one can distinguish two main types of the acquired heightfield data, the structured field which has been already sampled with a given resolution that will be the finest resolution our algorithm can use, and the unstructured field with raw data that will allow our algorithm to determine simply the finest resolution.

The sampling structure is provided in either a regular grid form or in form of irregular samples that imply the necessity of re–sampling stage over a regular grid that covers the data domain. The finest resolution grid is computed by applying a scattered interpolation method.

4.3 Scattered Data Interpolation

Basically, a variety of interpolation methods can be implemented over regularly gridded heightfield data and the purpose of choosing a specific method is based on the quality and the complexity requirements of any data exploration algorithm. Accordingly, our algorithm applied the implementation of high quality data interpolation in an efficient and coherent mechanism of a sophisticated interpolation technique as introduced by Park et al. [63].

4.3.1 Discrete Sibson Interpolation (DSI)

DSI is the discrete version of the Sibson Interpolation (SI). SI is used for scatter samples to interpolate any location utilizing the nearby samples, i.e., the natural neighbors. The natural neighbors of any sample are those samples that are created by Voronoi cells or the result of the connected sides of Delaunay triangles, more detail of Voronoi diagrams and Delaunay triangles had been presented by Fortune [28]. For example, in a 2D partition space the SI is computed over a set of sites $x_i \in X$ as follows:

$$F(x) = \frac{\sum_i a_i \quad F(x_i)}{\sum_i a_i}$$

where a_i is the shared area of the Voronoi cell $V(x_i)$ when computing the Voronoi diagram for X and the Voronoi cell $V(x)$ when computing the Voronoi diagram over $X \cup \{x\}$.

In DSI [63] the computations are simplified where the kdtree is used to get the closest site instead of using the complex structure of a Voronoi diagram. The efficient computations are done over a rasterized domain by using a sphere around a raster position i such that the radius of the sphere is the distance between i and the closest site. To compute the interpolation function value at i all values of every closest region are accumulated and averaged by the accumulated values number.

We used DSI to exploit the advantage of smooth surface interpolation using a kd–tree structure to store the data set. The DSI method is applied in our algorithm at every empty cell of the created grid. Hence, a set of overlapping circles are depicted with centers c_i at every empty cell p_i with radii r_i that are determined depending on the distance from the center c_i to the nearest filled cell value n_i using a kd–tree. Accordingly, n_i is then assigned to each cell within the related circle c_i. Let's p_{ni} be the accumulated value of the filled cell utilizing circle r_i and then the accumulated value is $p_{ni} = \frac{1}{N}\sum_{i=1}^{m} n_i$ where m is the number of overlapping n_i values of circle c_i and N is the number of accumulated values, see Figure 4.1.

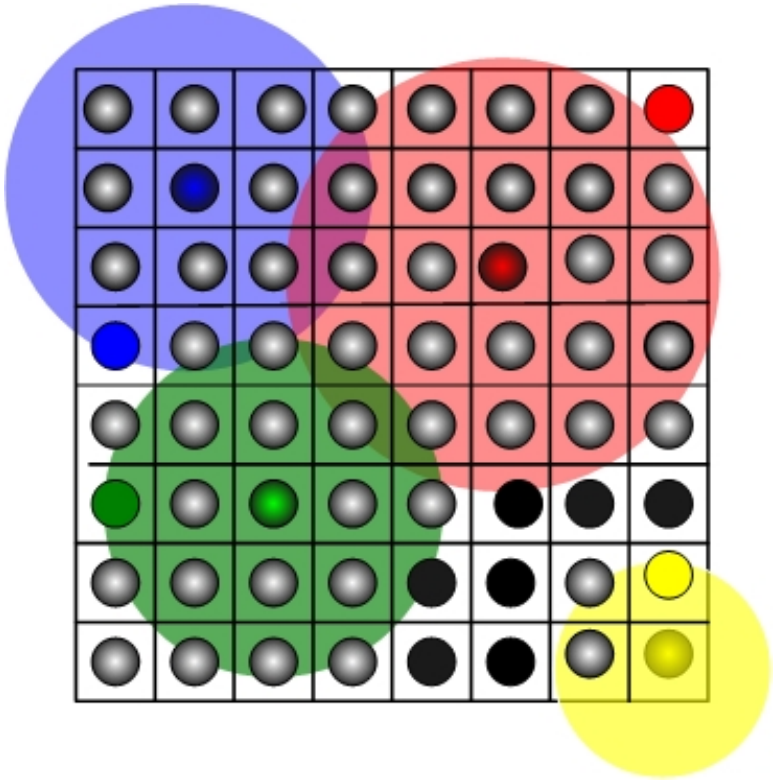


Figure 4.1: The implementation of DSI method on our algorithm where the colored disks are created with centers that are located at empty cells and radii that depict the distance from the empty cell to the nearest filled cell using kdtree search; non–gray dots are the filled cells, gray dots are the empty cells of the sampled grid; red, blue, green, and yellow dots on the circles' perimeters are the nearest filled cell.

Basically, in our work the data set is tiled such that the interpolation scheme is implemented on all tiles. For large number of tiles the interpolation is implemented on individual tiles. Thus,

we could get not well–stitched tiles. To avoid the occurrence of such problem one can apply the interpolation process not just to the tile but rather to the tile including all neighborhood tiles that are touching the tiles's borders. However, in this context other local simple interpolation methods are still required in this work.

4.3.2 Inverse Distance Weighting (IDW)

IDW interpolation method is vastly acknowledged for providing such local solution. Indeed, IDW interpolation method depends only on the influence of the distance parameter to compute the interpolation function, e.g. Shepard's method does the interpolation as follows:

$$F = \sum_{i=1}^{m} w_i \quad f_i \tag{4.3.1}$$

where m is the number of data points that are used to get the interpolation function value F at a point of interest, f_i are the function values at the data points, and w_i is the distance weighting function that is computed as follows:

$$w_i = \frac{{d_i}^{-2}}{\sum_{j=1}^{m} {d_j}^{-2}}$$

where d_i is the distance of the i–th data point to the point of interest from the point being interpolated such that

$$\sum_{i=1}^{m} w_i = 1$$

However, The IDW interpolation depends on one factor, i.e., the distance function whereas different other aspects can be involved within the interpolation process to get more confident impacts.

4.3.3 Kriging Interpolation

Kriging interpolation is one of the major techniques that has more sophisticated properties than those methods that are merely based upon the neighborhood distances from the point being interpolated such as in the IDW interpolation method. Essentially, kriging interpolation is synonymous

with optimal predication and it predicts the unknown value from an observed data at known locations. The supreme step in the kriging interpolation process is utilizing a variogram that reveals the spatial variation of a given data set. The predicted values are estimated via their own spatial distribution and subsequently any eventual error in the prediction operations is minimized.

In comparison to the IDW, kriging is:

1. Similar to (IDW) in considering the distance of the neighbors but with significant investigation involving:

   I) The orientation of the samples that surround the focused node.

   II) Releasing different weighting functions for perfect smoothing.

2. Different to IDW in which the samples can be in the manner of clustered data points.

On the one hand, kriging interpolation method exploits the distance property of the IDW method to emphasize that the influence of close points to the interpolated point is higher than the influence of far points. On the other hand, it covers the lack information in IDW methods that appears when a set of points fall on the same searching radius but with different influences based on their distribution around the target point.

The main step of kriging interpolation is using a variogram model in order to get the variogram function that will be used in the interpolation equation. The variogram model is chosen from a set of mathematical functions that describe the spatial distribution of a given data set. To select the model of best matching one should compare the shape of the curve of a generated experimental variogram and the shape of the curve of a mathematical function. Therefore, one needs to generate first an **Experimental Variogram (EV)**. The goal of creating an EV is measuring the similarity degree between two different variables in spatial context. As a trivial presentation of such a measurement to the variogram can not be provided, an auxiliary technique is incorporated in this step which is called the variance analysis process. For a given set of data points X defined with function values $F(X)$ the semi–variance is computed as:

$$\Gamma^*(h_j) = \frac{1}{2 \cdot N(h_j)} \sum_{i=1}^{N} (F(x_i + h_j) - F(x_i))^2 \tag{4.3.2}$$

where $N(h_j)$ is the number of observed data points at a delay time that is called lag where $lag = h_j$. The semi–variance $\Gamma^*(h_j)$ is actually half of the variance $2 \cdot \Gamma^*(h_j)$. However, the semi–variance at $lag = 0$ is solely the correlation since the variance will be computed for the point against itself.

The following step is the representation of the semi–variogram in a 2D plot with the X–axis depicting the $lags$ distances and the corresponding calculated semi–variances values on the Y–axis. The achieved EV characterizes a desired curve to fit the plotted variations with best smoothing parameters : sill=c, range=r, and nugget effect=n, that are the identifiers of the candidate model function of the variogram version that is used at an interpolation process. Sill is the location where the variogram leaves off the model, see Figure 4.2 (blue curve), and has overall scale from 0.0 to 1.0 and it is depicted as $1.0 - n$. Basically, the semi–variances start with $lag = 0$ at value zero. However, due to some error measurements a close to zero variance value results at lag distance $lag = 0$. Hence, a $nugget\ effect$ is often incorporated in the variogram parameters. The lag distance at value c is the range r which means the distance when the semi–variance becomes constant.

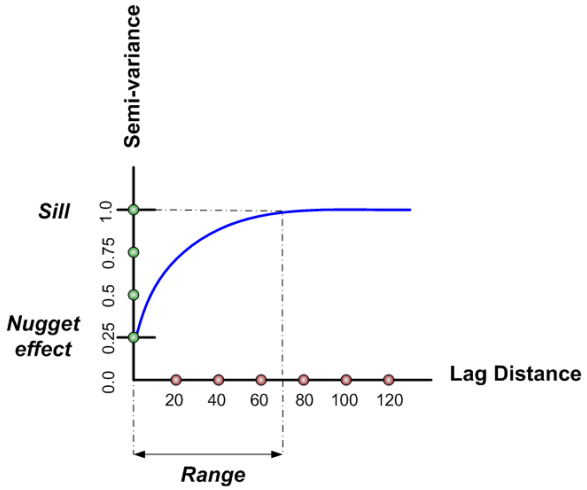


Figure 4.2: A 2D graph representing the variogram of spherical model with X–axis as the lag distances that uses meters as units and Y–axis as the semi–variance values depicted as the squared differences and it goes from 0.0 to 1.0 which expressed the parameter $Sill = c$. c is the value when the model (blue curve) leaves off and it could have 1.0 value or $1.0 - n$ where $Nugget\ effect = n$.

Variogram and Kriging Interpolation. The parameters r, c, and n are then used in the variogram model function that is concluded from the experimental variogram. However, the function is one of the following types:

Nugget:	$\Gamma^* h_j = \begin{cases} 0 & \text{if } \lvert h_j \rvert = 0 \\ c & \text{Otherwise} \end{cases}$
Exponential:	$\Gamma^* h_j = c \cdot \left(1 - e^{\frac{-3h_j}{r}}\right)$
Spherical:	$\Gamma^* h_j = \begin{cases} c \cdot \left(1.5\left(\frac{h_j}{r}\right) - 0.5\left(\frac{h_j}{3}\right)^3\right) & \text{if } \lvert h_j \rvert \leq r \\ c & \text{Otherwise} \end{cases}$
Gaussian:	$\Gamma^* h_j = c \cdot \left(1 - e^{\frac{-3h_j^2}{r^2}}\right)$
Power:	$\Gamma^* h_j = \ c \cdot {h^t}_j \qquad \text{where } 0 < t < 2$

Once the variogram is determined, kriging interpolation can be applied using the concluded $\Gamma^* h_j$ function for every data sample P_i as follows:

$$P_i = \sum_{j=1}^{N} (\Gamma^* h_j \cdot P_j) \tag{4.3.3}$$

where N is the number of samples that used for interpolating P_i sample and

$$\sum_{j=1}^{N} \Gamma^* h_j = 1 \tag{4.3.4}$$

4.4 Hierarchy Pyramid

We assume that the data are given in form of $2^{n \cdot k} \times 2^{n \cdot l}$ height values, where k, l, and n are natural numbers. The exponent n indicates the number of hierarchy levels. The numbers $k \times l$ is the size of the coarsest level.

Starting from the finest level with $2^{n \cdot k} \times 2^{n \cdot l}$ samples, we take groups of 2×2 samples' cells and compute the height values utilizing the numbers of real values in each cell as:

$H_k = \sum_{i=1,j=1}^{N,N}(n_{ij} \cdot H_{ij})$ where k is the level number; $N = 2^k$ is the number of the cells in one direction of the highest resolution grid; n_{ij} is the number of real values of the cell with index ij; and H_{ij} is a height value that is taken from the highest resolution level. The new height value is then used for the next–coarser level. The coarsening step is applied simultaneously to all samples such that we create $2^{(n-1) \cdot k} \times 2^{(n-1) \cdot l}$ samples. This procedure is iterated until we get to the coarsest level. Note that no adaptive data refinement is necessary for our approach. Moreover, the data representation is that of a piecewise constant height function.

4.5 Tiling Technique

The tiling technique is implemented in our approach in order to divide the huge heightfield domain to different tiles and store them on the hard disk. Our tiling model organizes the prepared tiles in a squared matrix in which the center of the tiles will be loaded to the GPU memory and the next frame of tiles will be loaded to the main memory. The rest of the tiles will stay on the hard disk until they are queried at run–time.

4.5.1 Tiling Model

The model is designed as a square matrix of tiles $\mathbb{T}(m, n)$ which is expressed as $\mathbb{T} = [t_{ij}]_{m \times n}$ where $i = 1, ..., m$ is the index of the tiles on rows and $j = 1, ..., n$ is the index of the tiles on columns and $m = n$. For instance a matrix $\mathbb{T}(m, n)$ with $m = 7$ and $n = 7$ is depicted as follows:

$$\mathbb{T} = \overbrace{\begin{bmatrix} t_{00} & t_{01} & t_{02} & t_{03} & t_{04} & t_{05} & t_{06} \\ t_{10} & t_{11} & t_{12} & t_{03} & t_{04} & t_{15} & t_{16} \\ t_{20} & t_{21} & t_{22} & t_{23} & t_{24} & t_{25} & t_{26} \\ t_{30} & t_{31} & t_{32} & t_{33} & t_{34} & t_{35} & t_{36} \\ t_{40} & t_{41} & t_{42} & t_{43} & t_{44} & t_{45} & t_{46} \\ t_{50} & t_{51} & t_{52} & t_{53} & t_{54} & t_{55} & t_{56} \\ t_{60} & t_{61} & t_{62} & t_{63} & t_{64} & t_{65} & t_{66} \end{bmatrix}}^{GPU+Main} \tag{4.5.1}$$

Matrix $\mathbb{T}$ consists of two main components; one is a matrix $\mathbb{T}_{GPU}[t_{ij}]_{3\times3}$ which is the inner part of the matrix $\mathbb{T}$ that is colored in green; and the other one is the outer frame of the matrix $\mathbb{T}$ which is colored in blue. Once the matrix $\mathbb{T}$ is loaded to the main memory, $\mathbb{T}_{GPU}$ is cached in the GPU memory leaving an opportunity for preceding loading of on–demand queried tiles from hard disk storage space. The default location of the viewing point P_v is in the center of the matrix $\mathbb{T}_{GPU}$ and in this context the element $\mathbb{T}_{GPU}[t_{ij}]_{3\times3}$ for $i = 3$ and $j = 3$ is used in Equation 4.5.1. For that reason, the viewing spot occupies an area within the tile t_{33} wherein the starting state is a vertical top–view above the rendered data. To allow for a fast navigation through the tiles and avoid stalling due to GPU–main memories transmissions, the set of tiles t_{ij} with $i, j \in [2, 4]$ that form a frame surrounding the tile t_{33}, are cached on the GPU memory at the start state as well.

4.6 Discussion and Results

As the targeted data could be provided in both types gridded and un–gridded, we can create in more flexible situation the desired resolution over a grid of un–gridded data and the work will be restricted to the given resolution over a gridded data. However, as the interpolation process is included in the preparation stage of our algorithm, different interpolation methods could be utilized either regardless the consumption time and focusing on the quality (e.g DSI interpolation technique), or medium quality (e.g IDW interpolation method), or acknowledging the efficiency in time and quality simultaneously (e.g kriging interpolation method). Figure 4.3 shows the result of the experimental variogram and different fitting curves to extract the best approximation function (in this example Gaussian curve) for the final variogram and kriging interpolation.

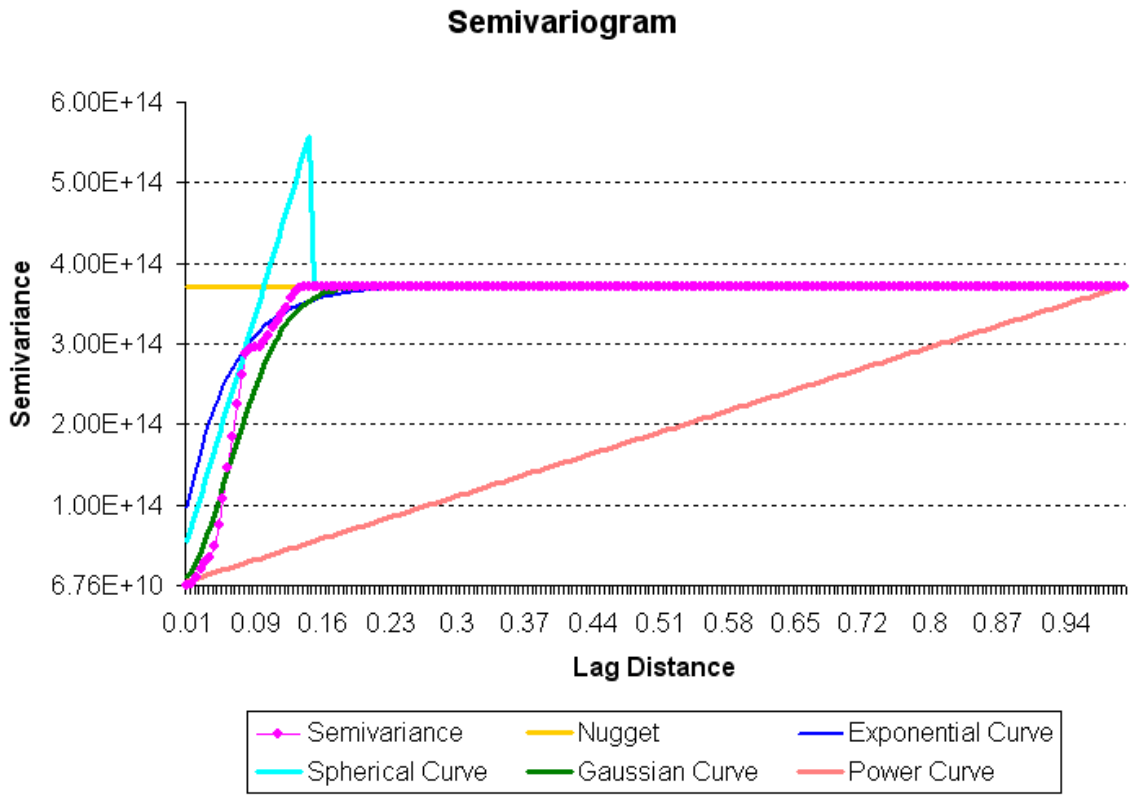


Figure 4.3: An experimental variogram of Black See Ereğli data (pink) as example with different approximation functions to show the best curve that fits the semivariance smoothly. Nugget effect is depicted in gold color, exponential function curve in blue color, spherical function curve in cyan color, Gaussian function curve in green, and power function curve in coral color.

Moreover, the layers in the hierarchy pyramids constructing stage are made up ever of the highest resolution layer with the real number of data points and hence any produced smoothing does not involve the effect from previous constructed layers.

Chapter 5

Interactive Visualization of Gridded Heightfield Using Pre–computed Mesh

In the runtime stage, the decoupled mesh and data representations are brought together. We always render the same mesh with varying height values. Hence, the x– and y–parameter plane positions of the vertices never change, nor changes the mesh connectivity. This approach entails various advantages.

I. We always have a continuous, watertight mesh representation.

II. The triangle shapes are well–behaved (unless we have sudden dramatic changes in the height values, where long, stretched triangles cannot be avoided when using a parameterization over a plane; this issue is discussed in more detail in Section 5.8).

III. The triangles can be stored on the GPU, effectively avoiding CPU–GPU transition.

IV. The translations in different viewer positions over the captured data in GPU memory are free from heightfield loading processes.

V. Multiple data sources are implemented in one rendering pass.

VI. Different tiles are generated for the given heightfield domain. Those tiles are stitched smoothly and one can navigate upon the view–dependent constant mesh with all possible views without the need of implementing subdivision schemes to achieve the desired heightfield adaptation.

In order to achieve heightfield updates, we start with organizing the heightfield domain in different tiles and generate a hierarchical pyramid for every tile. Those pyramids are well–suited for texture objects. Therefore, we cache the pyramids in texture memories and send few of them to GPU considering the limitation of the GPU storage. The stored height values on GPU are accessible utilizing shader programs. Moreover, the constant generated triangular mesh caches its vertices in a Vertex Buffer Object (VBO) that may be transmitted to GPU as well. Once the VBO is on the GPU, no CPU–GPU transformations are needed because we do not change the structure of our mesh at run–time and height updates are achieved by a Vertex Shader (VS) program using the Vertex Texture Fetch (VTF) technique. VTF is a shader feature of GPU and it allows the VS to read data from textures. Essentially, such a feature is very useful in our approach as our work is concern with the update of the heightfield per vertex. However, CPU–GPU transfers are executed for heightfield updates which require height values that are not available in GPU memory.

5.1 The Design of Hierarchy Pyramids Caching

To gain an optimized rendering process, pyramids of hierarchies are cached in memory entities of appropriate capability to produce fast and simple CPU–GPU transitions at run–time. Since levels of each pyramid can be represented by a 2D array, texture object memory is the most suitable media for capturing our pyramids.

Pyramid levels are 2D grids that include the information that could be cached in a texture memory. Therefore, in our algorithm the hierarchical levels are mapped to 2D textures exploiting their incredible benefits including first, the ability to retrieve data from textures at run–time without extra CPU–GPU invokes; second, the flexible manipulation of texture parameters and environment such as filtering approaches, border modes, etc. third, switching on/off the automatic mipmapping. The standard mipmapping technique is fast and free of the aliasing effects but every map has values that are the result of smoothing all values through the previous levels of the pyramid. To avoid such smoothing and build every level from only non–smoothed values we always create every level depending on the finest level of detail. In here, the built–in mipmapping is turned off and replaced by a manual handling in which we implement linear interpolation between different levels of detail in each pyramid.

Once the levels of the hierarchies are mapped to 2D textures and the height/normal and/or extra information at different levels of resolution are efficiently cached on the GPU, the pyramids are readily available as texture lookup tables. Those tables are used always for retrieving height values and are not reloaded unless the viewer enters new heightfield areas.

The only information that needs to be sent from the CPU to GPU is the viewing information unless tiles need to be swapped. For a given viewpoint, the distance to the parameter plane is derived. This distance determines what area is covered by each triangle, see Figures 3.2. Then, the appropriate level of the hierarchy is determined by comparing the triangle's size s_Δ against the screen pixel's size s_p as: If $s_\Delta \in [2^{i-1}s_p, 2^i s_p]$, we pick the data value from a level with resolution $2^{n-i} \times 2^{n-i}$, where $i \in \{0, \ldots, n\}$ and n denotes the highest resolution in the pyramid. In pursuance of selecting the levels of detail, our work incorporates a GPU–based technique to fulfill this task via integrating the graphics parallelism.

5.2 Vertex Texture Fetch (VTF)

In the VTF feature of the GPU the triangle sizes of the created triangular mesh is optimized against a given screen pixel size efficiently through considering a ratio factor

$$R = \frac{\#P}{\#T} \tag{5.2.1}$$

where $\#P$ is the number of screen pixel, and $\#T$ is the number of mesh's triangles. Factor R shows how many pixels are permissible to lie within one triangle which provides its height as a length to be compared to the level's pixel width for determining which level is requested. The decision of selecting which level of detail is required is done per triangle. The vertices of the triangle are updated following either the same level of detail if they lie on one circle in a ring of the view–dependent layout model or different levels for different circles per ring. Basically, the triangles are ordered starting with the center of the view–dependent layout outwards in the counterclockwise manner through the rings such that every triangle stores its size in its vertices before going to the next triangle. Once the vertex has been informed about one triangle size it always uses it for the update process. Since the provided triangular mesh is constant the triangle size information can be cached in the vertex attributes that are accessible to the Vertex Shader program (VS). In the VS 2D samplers are

performed using $tex2Dlod$ function to access the textures lookups.

5.3 Implementing Vertex Texture Fetch (VTF)

Basically, the major part when implementing the VTF is the lookup function $tex2Dlod(...)$ with a manual mipmapping property. The following snippet shows this function in the VS program.

```
(Vertex Shader Program )
sampler2D tex;
...
void main(void)
{
...
HeightUpdate =tex2Dlod( tex, t,0 );
% where t : refers to the selected level
% 0, referes to switching off the automatic mipmapping
...
}
```

The previous snippet example presents the lookup function as $HeightUpdate = tex2Dlod(tex, t, 0)$ which accepts three parameters $p_1 = tex, p_2 = t, p_3 = 0$ as inputs and returns one value $V = HeightUpdate$ as retrieved value. In general, a uniform variable is a global variable that is passed to the shader program and not changed within one rendering call. From the different types of uniform variables we used $Sampler2D$ for parameter p_1 that is used to pass a $2D$ texture. Parameter p_2 identifies the (x, y) location within texture tex to retrieve the height value and store it in V. Parameter p_3 is set to zero to switch off the automatic mipmapping and do our method to select the level of detail.

The following snippet shows the VS code for retrieving the height value:

(Vertex Shader Program)

```
attribute float h;
uniform sampler2D Texture0;
uniform mat4 ProjectionModelviewMatrix;
vec2 TexCoord;
float GetLevel(float Height);
void main()
{
vec4 texel, newVertex;
% Read the texture offset. Offset in the z direction only
l_i = GetLevel(h);
if( l_i ==0)
{
texel = texture2DLod(Texture0, TexCoord, 0.0);
}
% else is used for the rest of the levels
% such that l_i = 1...N
newVertex = gl_Vertex;
newVertex.z += texel.x;
gl_Position = ProjectionModelviewMatrix * newVertex;
}
```

To accomplish implementing the VTF a bunch of settings are required for both the VS program and textures on the *OpenGL* client.

5.3.1 Textures and VS Settings

The respective height values are obtained by a texture lookup of the texture that stores the determined resolution. The texture lookups are set on the *OpenGL* client as the following snippet:

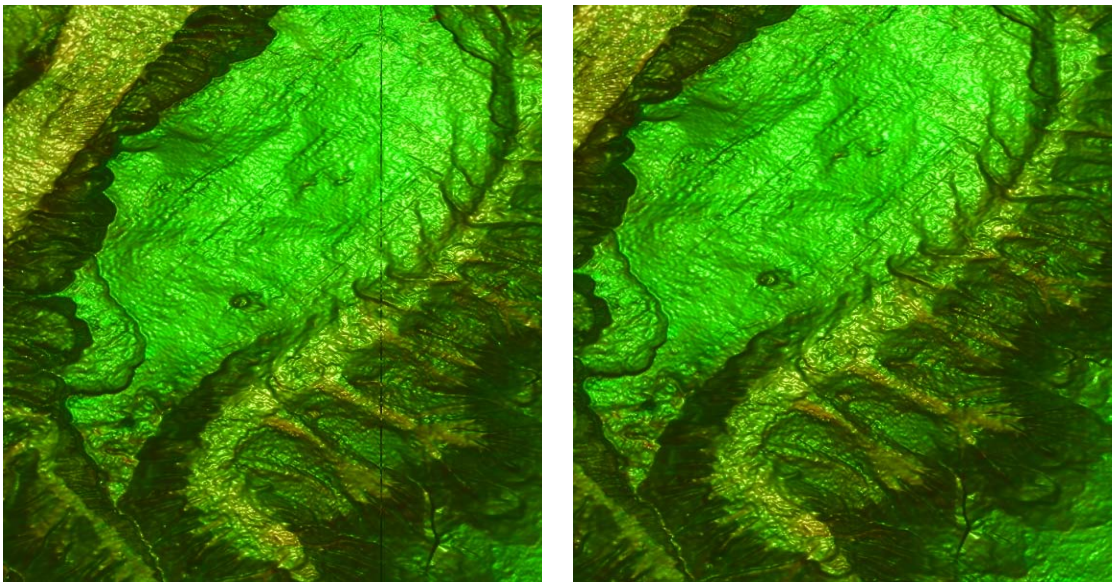
(OPENGL)

```
GLuint vertextexture;
glGenTextures(1, &vertex_texture);
glBindTexture(GL_TEXTURE_2D, vertex_texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_LUMINANCE32F_ARB,
width, height, 0,GL_LUMINANCE, GL_FLOAT, data);
```

As our data representation is a piecewise constant one, a nearest–neighbor interpolation is applied during the texture lookup. However, the issue of linear interpolation at texture borders that caused via incorporating invalid data from outside borders could be solved through extending each tile borders one pixel from surrounding tiles. Hence, no discontinuities result, see Figure 5.1, and it requires adding some commands for setting the suitable parameters as follows:

(OPENGL)

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
```



(a) (b)

Figure 5.1: The comparison between our results using Black See Ereğli data without edge extension showing the discontinuity between adjacent tiles (a), and with edge extension solution (b) that shows removing this problem.

The data tiles of the gazed regions have to be loaded to textures, transmitted to GPU memory, and put in the active mode to be accessible to the vertex shader program as follows:

```
(OPENGL )
glActiveTexture(GL_TEXTUREi);
glBindTexture(GL_TEXTURE_2D, vertex_texture);
```

Note that fetching values from textures within the vertex shader program imposes using a suitable NVIDIA graphics card of series Gf 8 or higher that allows the running mode to occur on GPU and not as software emulation mode. Therefore, there are a specific texture internal formats available for this task which are GL_LUMINANCE32F_ARB and G_RGBA32F_ARB.

5.4 Level of Detail Selection

Here, we derive the level of detail that is selected utilizing the optimized triangle's size at run–time.

Suppose that we have an optimized triangle size for a screen resolution $M \times N$ with pixel size

$P_s \times P_s$. By utilizing Equation 5.2.1 we can get the approximation number of screen pixels that are covered by an optimized triangle and represented as:

$$N_s = \lceil R \rceil \tag{5.4.1}$$

In any created triangular mesh our triangles have triangle shapes ranging from equilateral triangle to well–shaped isosceles triangle. Thus, the height h of the triangle Δ_i could be extracted from the distance between two consecutive rings of the predefined mesh layout. Each pixel side in the highest resolution level of detail P_{l_0} can be computed using Equation 5.4.1 as follows:

$$P_{l_0} = \lceil \sqrt{N_s} \rceil \cdot P_s \tag{5.4.2}$$

Hence, from Equation 5.4.2 we can determine the side length of the appropriate level of detail pixel P_{l_i} as following:

$$P_{l_i} = \begin{cases} = P_{l_0} & \text{if } F(h) \leq 1 \\ = \lceil F(h) \rceil \cdot P_{l_0} & \text{if } F(h) - \lfloor F(h) \rfloor \geq 0.5 \text{ and } F(h) > 1 \\ = \lfloor F(h) \rfloor \cdot P_{l_0} & \text{if } F(h) - \lfloor F(h) \rfloor < 0.5 \text{ and } F(h) > 1 \end{cases} \tag{5.4.3}$$

where $F(h) = \frac{h}{P_{l_0}}$. Eventually, which level l_i is calculated utilizing Equations 5.4.2 and 5.4.3 as follows:

$$l_i = \lfloor \frac{\lfloor \frac{P_{l_i}}{P_{l_0}} \rfloor}{2} \rfloor \tag{5.4.4}$$

5.5 Retrieving Height Values

Suppose that the created $2D$ mesh is centered at $v_c = (x_c, y_c, z_c)$ with a vertical pivot of the $2D$ mesh that passes through the viewing point P_v. Thus, the update is achieved at every mesh vertex $v_m = (x_m, y_m, z_m)$ by adding the retrieved height values to the z–coordinate of mesh vertices. Hence, x–coordinate and y–coordinate are used to calculate the corresponding coordinates within the candidate texture.

Depending on the layout model all mesh vertices lie on the centered circles and are depicted in terms of parametric variables of the circle equation as:

$$\begin{aligned} x_m(t) &= r_k \cdot \cos(t) + x_c \\ y_m(t) &= r_k \cdot \sin(t) + y_c \end{aligned} \tag{5.5.1}$$

with $k = 1, ..., M$, where M is the number of the farthest circle from v_c, radius r_k, and parameter $t \in [0, 2\pi]$.

The updated vertex belongs to a region within a tile such that tiles numbering starts with the upper left corner of the tile and it goes to rightwards/ downwards as it is presented in Section 4.5. The pixel indices within each tile are analogous the those of matrix $\mathbb{T}$.

Since the vertices of the $2D$ mesh start with the center $v_c(x_c, y_c, z_c)$ to outward radial directions we get the index of the candidate tile as:

$$\begin{aligned} I &= \tfrac{x_m}{N-1} + \tfrac{N-1}{2} \\ J &= \tfrac{y_m}{N-1} + \tfrac{N-1}{2} \end{aligned} \tag{5.5.2}$$

where $N \times N$ is the tile resolution and $M \times M$ matrix of tiles with indices start with the upper left corner of the tile as l_{ij} for $i, j = 0, 1, 2, ..., M \cdot N - 1$. Hence, the pixel index (P_i, P_j) within the tile of index (I, J) is:

$$\begin{aligned} P_i &= \tfrac{x_m - (l_{00} + I \cdot N)}{N} \\ P_j &= \tfrac{y_m - (l_{00} + J \cdot N)}{N} \end{aligned} \tag{5.5.3}$$

Equation 5.5.3 can be written as linear function:

$$\begin{aligned} P_i &= a_x \cdot x_m - b_x \\ P_j &= a_y \cdot y_m - b_y \end{aligned} \tag{5.5.4}$$

Where $a_x = a_y = \frac{1}{N}$ and $b_x = \frac{(l_{00} + I \cdot N)}{N}$ and $b_y = \frac{(l_{00} + J \cdot N)}{N}$

Basically, the pixel index in Equation 5.5.3 or 5.5.4 is the textures coordinates that define the location of the height value h that should be retrieved to be added to the z–coordinate z_m leading to vertex $v_m = (x_m, y_m, z_m + h)$.

5.6 Height Update

The update process is achieved on the GPU within the Vertex Shader (VS). The VS processor inputs a triangle size Z_i and one of the triangle's vertices $v_m = (x_m, y_m, z_m)$. Then, Z_i is tested against

the pixel's size of the projection screen using the given ratio. This request is sent to retrieve the appropriate height value h from the respective texture by implementing the VTF for VS 3.0 GPUs. Thus, h is added to the z–component z_m of the vertex v_m by the following look up function:

$$h = texture2DLod(TextureID, x_m, y_m, LoDID)$$

where $texture2DLod$ is a lookup function of the mipmapping, $TextureID$ is a texture sampler, and $LoDID$ is the index of the current level of detail. As an output, we obtain $v_m = (x_m, y_m, z_m)$ with $z_m = h$.

Due to the fact that the triangular mesh generation is based on the properties of perspective projection, view–dependency is already encoded in the mesh and the triangular mesh structure is static. Thus, the exploration processes of changing the viewpoint forth/back and left/right can be achieved without any modification of the triangle's size. Nevertheless, an adaptation is required when performing a zooming operation. In this case, we need to retrieve the height value from a different level of detail and need to adapt the viewer distance accordingly. The updated triangle's size is computed as

$$Z_i = \frac{X_j}{x_k} \cdot z_i$$

for $i = 0, \ldots, n$, where n is the number of triangles, $j = 0, \ldots, l$, where l is the number of viewing updates, and $k = 0, \ldots, p$, where p is a constant for a given mesh. Thus, a new triangle's size Z_i results from the previous size z_i when the viewer changes its location to distance X_j from the 2D mesh plane and the distance x_k is the default viewer distance from the 2D mesh plane.

For more efficient rendering, the static triangular mesh is cached on the GPU through a Vertex Object Buffer (VBO).

5.7 Translations and Rotations

When changing the viewing position, we distinguish between flying, rotation, and zooming operations. In a zooming operation, the distance to the parameter plane changes. Consequently, we need to re–estimate the distance to the parameter plane. When the viewpoint gets closer to the parameter plane, the triangle captures a smaller size of the terrain. A different level of resolution is

required for the height computation but we stay in the same region of the tile that resides on GPU which means we do not need any tile loading from CPU to GPU, see Figure 5.2.

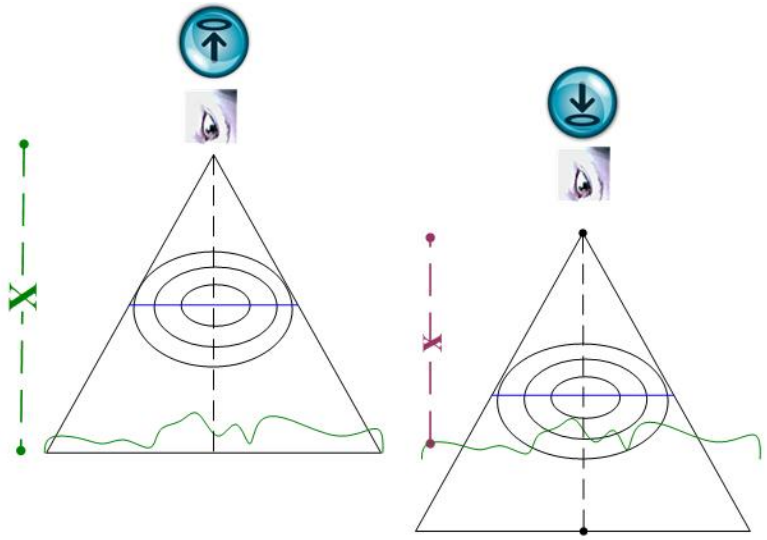


Figure 5.2: Zooming operation changes the distance of the terrain to the viewer. The mesh's triangles capture smaller terrain areas. The height lookups need to be performed on the finest level of detail.

A flying operation does not change the z–coordinate of the viewpoint. Hence, the levels of resolution stay the same. Still, all heights need to be updated, as each triangle captures a new area of the terrain, see Figure 5.3.

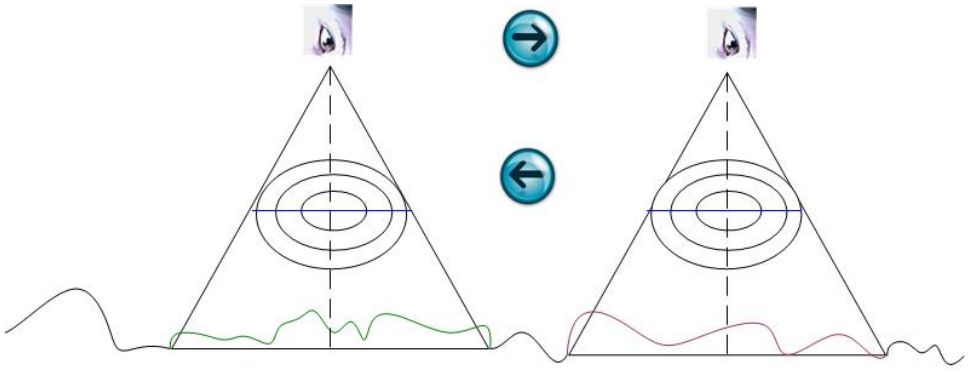


Figure 5.3: Flying transmissions require the updates of the heights with respect to a shifted terrain region. The used levels of resolution are the same.

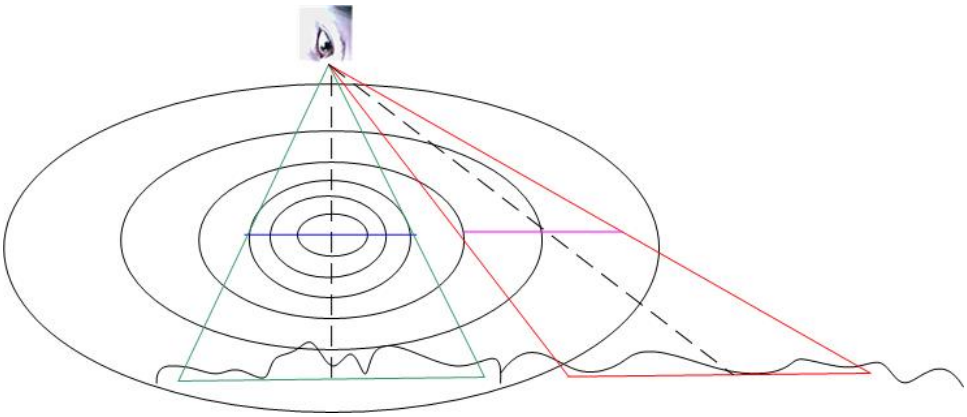


Figure 5.4: Rotation does not require any update.

When not changing the viewpoint but the viewing direction, the height values of the mesh's vertices do not change, see Figure 5.4. Hence, no update is necessary. Our approach targets the creation of a fixed mesh with geometries that are transmitted to the GPU once. Accordingly, we do not apply any view frustum considerations but rather update all vertex heights when zooming or flying. When applying no view frustum, a rotation does not need to update any heights. Alternatively, we also implemented a version where only vertices within the view frustum get a height update when zooming or flying. However, this did not improve the rendering performance.

5.8 Discussion and Results

The generation of the mesh optimized size and shape of triangles assuming a flat heightfield. Adjusting the heights according to the given heightfield data obviously affects the size and shapes of the triangles. Thus, an updated triangle may cover more than the anticipated number of pixels. However, this increase of triangle size can be determined beforehand by looking into the maximum height difference between neighboring samples. Then, the ratio of triangle size to pixel size can be adapted accordingly.

Suppose that $\mathbb{S} = \{s_i\}$ is a given data set over a grid with height values s_i where $i = 1, 2, ..., N$ and N is the number of samples of the entire data. For each sample with height value s_i we refer to $s_j \in \mathbb{S}$ as the set of height values of the neighboring samples. Accordingly, the Global Maximum

Difference $\mathbb{GMD}$ is defined by:

$$\mathbb{GMD} = \max_{i=1}^{N}(\max_{j=1}^{M}(|s_i - s_j|)) \tag{5.8.1}$$

$\mathbb{GMD}$ is then compared to the optimized size of the triangle. If it happens that $\mathbb{GMD}$ is too far from the triangle size of the current prepared mesh version a new mesh is created with smaller triangle size. The optimal frame rate is reduced when rendering more triangles but for all the data sets we considered it is still higher than the state–of–the–art considering the continuous static structure of the triangular mesh over all frames at run–time.

Moreover, when generating the grid hierarchy on the CPU, each level of a computed hierarchy is stored in a 2D array whose cells accommodate the data information that beforehand has been prepared in a preprocessing stage. The data preparation phase may include re–sampling to a grid (if necessary) or normal computation. For the examples provided in this work, each cell stores a height value and a surface normal. Color or surface texture values could be easily added, if provided.

We exploit the texture memory on the GPU, in which each hierarchy caches its levels in 2D textures whose texels capture the relevant data information.

However, one issue when utilizing the texture memory stems from the synchronous CPU–GPU execution of processes. While data are transferred to a texture, any CPU execution process is stopped, which leads to a stall every time the heightfield is updated. This becomes important in the context of tiling. To overcome this issue, we employed a Pixel Buffer Object (PBO) as an alternative memory storage concept utilizing the advantage of asynchronous CPU–GPU steps for efficient data storing and transferring.

The dependence on a fixed mesh and the decoupling from the heightfield optimization address our main contribution of presenting a fast method that produces a high rendering quality at run–time. In this method we can generate an optimal mesh using a ratio that performs a perfect quality with sub–pixel errors. The adjustment of such ratio can be achieved to speed up the whole approach when accepting a few larger triangles.

Chapter 6

Results and Discussion

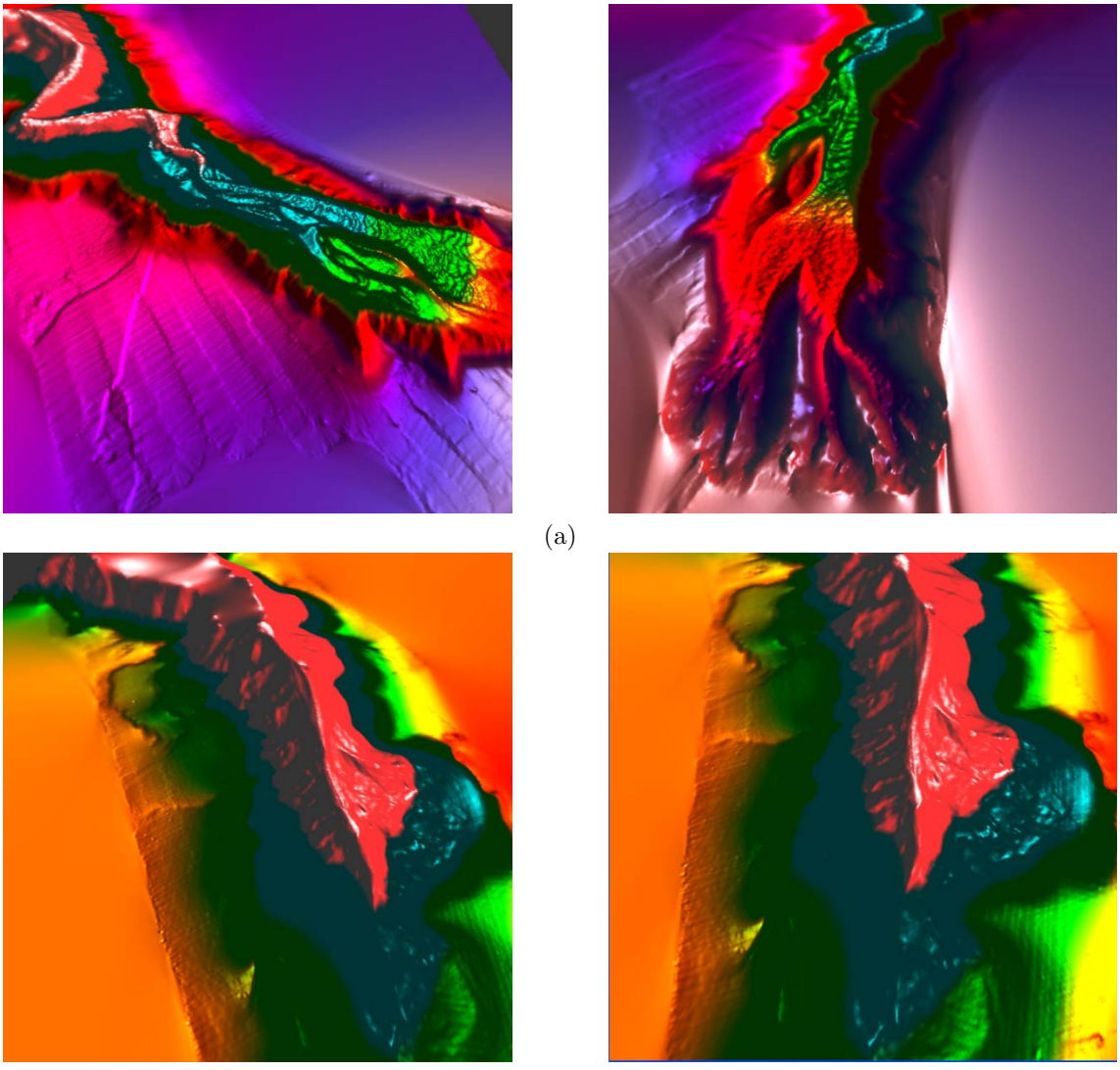
Our results are based on one synthetic data set and several real data sets. The synthetic data set is just a sampling of a repetitive pattern using trigonometric functions. This data set has mainly been used to create arbitrarily many tiles, which are then used to test the performance of our tiling algorithm. There are two of the real data sets that come from our motivating application to visualize ocean floor data and bathymetric data that are obtained using multibeam sonar scans. This scanning technique is a common one and provides unstructured data of a specific area of the ocean bed. Several real data sets are used one of them represents the Soquel Canyon with 1,457,778 sample points [68] and another is the Monterey Bay Canyon with 5,136,690 sample points [68]. Although the data represent depth values, it is not uncommon to visualize the data interpreting the values as heights [68]. The visualizations we present here follow this visual exploration concept. Since the obtained data is unstructured, we applied a data preparation step that re-sampled the data to a regular grid that consists of 50×50 tiles where each tile is of size 1024×1024. Next, we constructed the grid hierarchy levels and we cached them in 2D textures as described above.

The tested UTAH data is $790\,GB$ covering a landscape of $460\,km \times 600\,km$ with $5\,m$ heightfield resolution and $1\,m$ satellite image resolution and for screen resolution of $\approx 2\,megapixel$ our algorithm shows performance of $60\,fps$ when the satellite images are used and $90\,fps$ if they are not used. Dick et al. [18] are reporting high performance such that they used a better hardware of an NVIDIA GeForce GTX 280 graphics card with 1024 MB of local video memory. The achieved performance is high during flying operations as long as the viewer is flying over tiles of the heightfield that are already stored on GPU. When loading tiles from CPU to GPU the performance slightly drops to be

in the range between $\approx 57\,fps$ and $\approx 60\,fps$.

Notice that the 1024×1024 resolution governs the finest level for each hierarchical pyramid of one tile, i.e., the finest resolution of the entire data set equals $n \times 1024 \times 1024$, where n is the number of tiles. Moreover, the data set size does not affect the performance.

Figures 6.1 and 6.2 show some visualization results when applying our approach to the Monterey Bay Canyon (a) and the Soquel Canyon (b).



(a)

(b)

Figure 6.1: Our height field visualization of Monterey Bay Canyon (a), and Soquel Canyon (b).

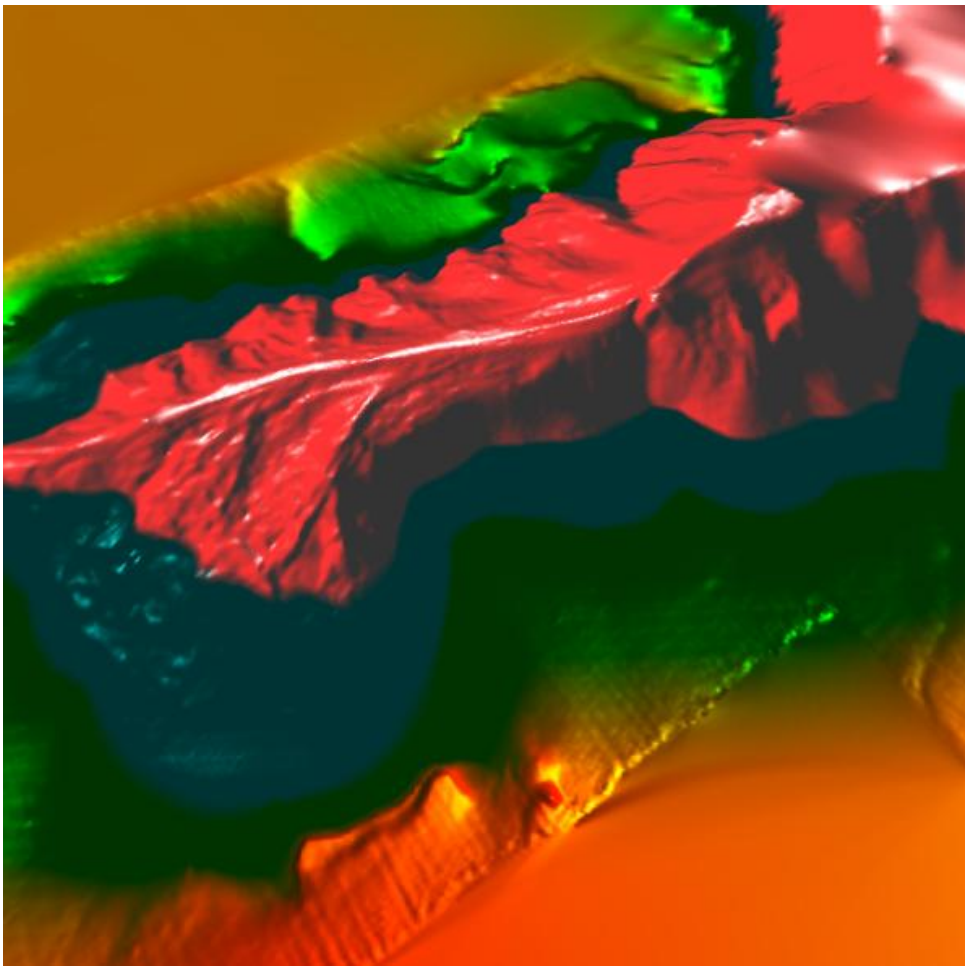


Figure 6.2: Zoomed–in view on the Soquel Canyon using our approach with a decoupling of mesh and data representations.

Figure 6.3 shows our results for synthetic data exploiting the new features of the NVIDIA Gf 8 graphic cards for implementing an anti–aliasing technique in which we can use the $16xQ$ $CSAA$ quality with tiny unnoticeable difference in performance and high quality. More detail on the $16xQ$ $CSAA$ method is presented in an NVIDIA white paper [58].

6.1 Ratio of Pixel Size to Triangle Size

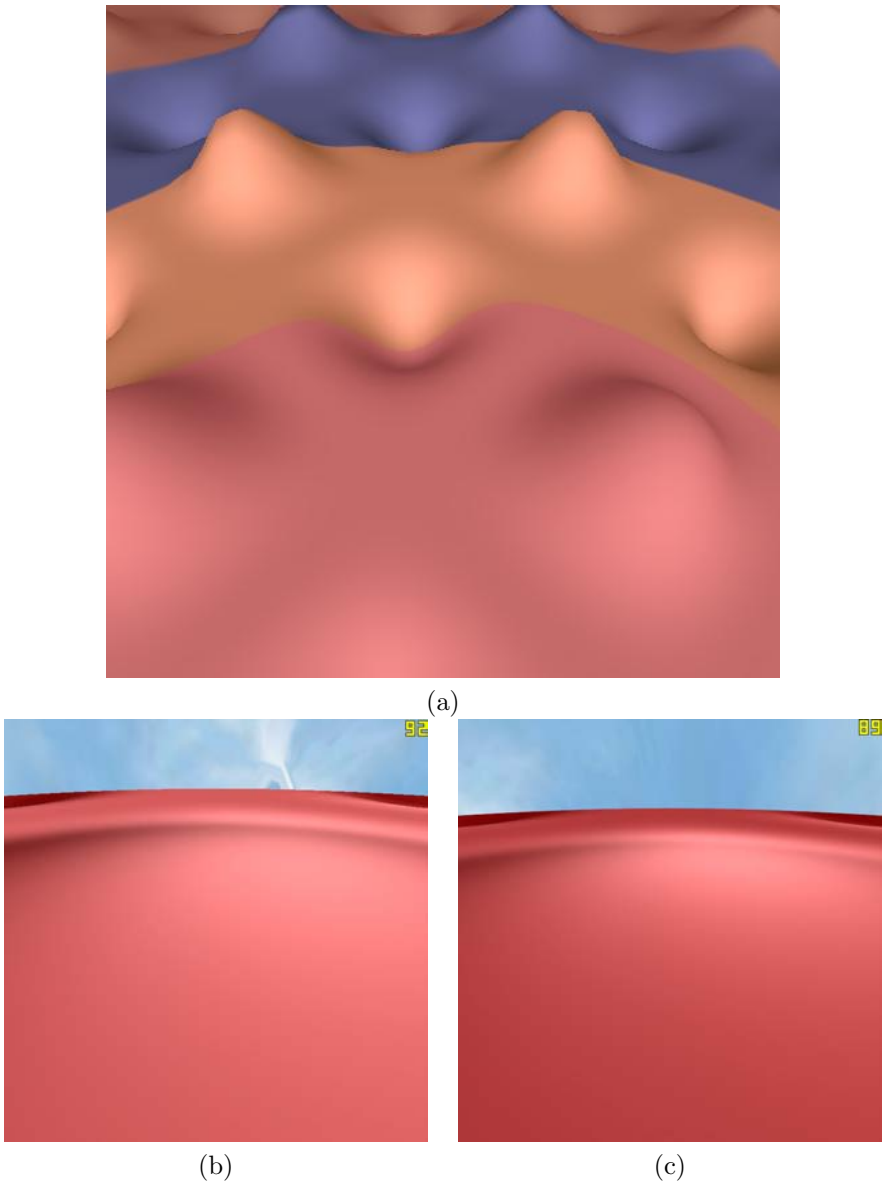
As all our triangles are supposed to be projected to the same size which is a global constant. However, the constant can be adjusted globally.

We optimized the triangular mesh against different display screens using the computation of the ratio factor R as

$$R = \frac{\#P}{\#T}$$

where $\#P$ is the number of pixels of the display screen and $\#T$ is the number of rendered triangles. Hence, this parameter describes how many pixels are covered, on average, by a triangle. Basically, we use the $\mathbb{GMD}$ that is presented in Section 5.8 to determine the bound of the triangles number in a triangular mesh. However, if the bound is unfeasible we can find a global R for generating an optimal triangular mesh. It is worth mentioning that the $\mathbb{GMD}$ may be very high, if there is a sudden jump in the heightfield (like a cliff). However, any heightfield visualization has a problem with rendering such cliffs which is an inherent problem with representing the data over a 2D parameter plane. In case of existence of such cliffs, it makes sense to adjust the R to a more suitable value.

Figure 6.4 shows a comparison of three visualizations that we obtained when applying it to a zoomed–in version of one of the canyon data sets and using the ratios $R \approx 2.8$, $R \approx 3.4$, and $R \approx 5.3$. As expected, the rendering quality decreases with increasing ratio R. In particular, it can be observed that the visualization for $R \approx 5.3$ does not reflect all the details that are visible in the visualization using ratios $R \approx 2.8$ and $R \approx 3.4$. However, at a certain point, the rendering quality does not increase further when decreasing ratio R. Moreover, the ratio R also influences the performance: The smaller the ratio, the more triangle need to be rendered, i.e., the lower is the performance. Hence, the aim is to pick the largest R that produces optimal renderings. We observe



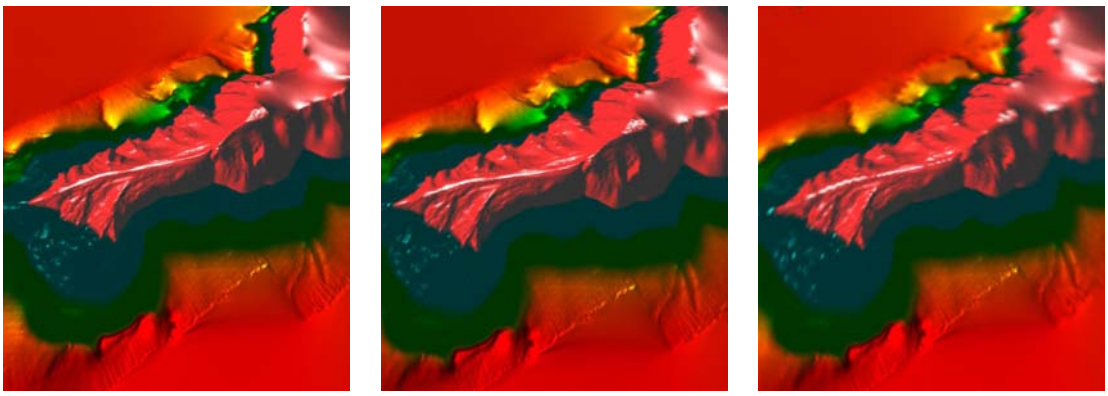
(a)

(b) (c)

Figure 6.3: Synthetic data with different levels of detail that are encoded in different colors and show up as concentric rings (a). Covering–Sampled Anti–Aliasing ($16xQ\,CSAA$) of NVIDIA graphics card with series $Gf\,8$ is implemented in our work to get rid of jagged silhouette as in (b) and to produce high quality smoothing within a desired performance (c).

that the rendering quality does not increase when going from $R \approx 3.4$ to $R \approx 2.8$. The ratio $R \approx 3.4$ seems to be the optimal choice, as reducing the ratio further does not increase visualization quality while increasing the ratio does significantly reduces the quality.

It is worth mentioning that ratio R is independent of the size of the rendering window. Hence, our findings can be applied generally.



R=2.8 R=3.4 R=5.3

Figure 6.4: The comparison between three different ratios to estimate the best number of triangle in preprocessing stage.

6.1.1 Triangle Shapes

Next, we want to evaluate the quality of our triangles with respect to their shape. As a quality measure we use the ratio as presented in Section 3.4. Figure 6.5 shows the distribution of the quality measures for all rendered triangles using our scheme. The figure compares the triangle qualities without the height update with the triangle quality after the height update. A resulting triangular mesh (before the height updates) was shown in Figure 3.3 (b). When updating the heights according to the synthetic dataset, the ratio stays more or less the same. This observation is mainly due to the smooth function that is being used for the synthetic data.

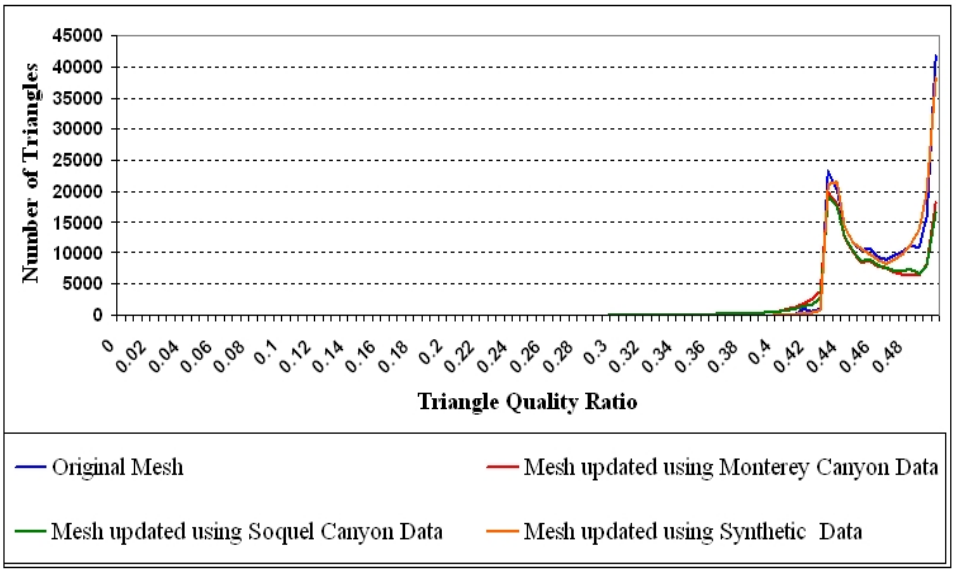


Figure 6.5: The evaluation of the triangular mesh quality. Three data sets are involved in this experiment, Monterey Bay Canyon Data (red), Soquel Canyon Data (green), and Synthetic Data (orange). Most of triangles have a quality ratio R_Q between ≈ 0.44 to 0.5. Our subdivision scheme survive triangles to capture the appropriate ratio again. View–dependent layout showed an optimal mesh before (blue) and after adaptation processes.

6.2 Performance

Table 6.1 reports the frame rates we achieved for different sizes of the rendering output window and different ratio R (directly affecting the number of rendered triangles). All experiments were processed on a PC with a 2.66 GHz Intel(R)Xeon(R) processor, 3.25 GB main memory, and an NVIDIA Geforce 8800 GTX graphics card with 768 MB video memory.

For the targeted ratio R, we achieve frame rates of 90 fps for rendering output windows of a size up to 1000×1000 pixels. When using an output size of 1600×1200, the frame rates are still at 60 fps. When using a larger ratio R, frame rates go up to 133 fps and 90 fps, respectively, at the expense of a lower–quality rendering. When using a smaller ratio R, frame rates go down to 60 fps and 30 fps, respectively, without any gain in quality for the considered data.

Next, we made experiments to figure out whether the texture lookups to update the height values of the vertices is a bottleneck of our pipeline. We implemented two versions of our approach: The first version updates the heights of all vertices at each frame, the second version updates only the

Ratio	$R \approx 2.8$			$R \approx 3.4$			$R \approx 5.3$		
#P	700 × 700	1000 × 1000	1600 × 1200	700 × 700	1000 × 1000	1600 × 1200	700 × 700	1000 × 1000	1600 × 1200
#T/s (K)	10,800	22,040	20,400	13,050	26,600	34,200	12,236	24,970	33,300
fps	60	60	30	90	90	60	133	133	90

Table 6.1: Frame rates (in fps) and triangles per second (#T/s) for different ratios R and different output sizes (number of pixels #P) when applied to the Soquel Canyon and Monterey Bay Canyon data sets. The optimal ratio is $R \approx 3.4$.

heights of the visible vertices (at the expense of checking visibility, which is a simple test though). Both versions reported the same frame rates. Even when not doing any height updates, the frame rates did not go up. Hence, the texture lookups are not a bottleneck.

The tiling approach is tested using synthetic data set and two real data sets. In our first experiments, we observed a short delay when loading new tiles. We have modified our implementation by spreading the loading procedures over a few frames.

Basically, when the pyramids are being cached on the texture memory of the GPU, an automatic mipmapping does the task of the interpolation between the levels of detail. However, as we are replacing our manual mapping by the automatic mipmapping we do linear interpolation between every two followed levels of detail within the VS program.

To summarize the findings, the comparison with the state of the art delivers the following insight. We considered the most recent advances of mesh-based approaches [18] and raycasting approaches [19]. Recent mesh–based approaches operate on regular grids and use compression for a

fast throughput of geometry submissions to the GPU. Regular meshes outperform irregular meshes, as the processing of the latter typically involves computations on the CPU. For the UTAH data set, compression of regular meshes was applied quite successfully and frame rates of about 200 fps on average were reported on a 1280×1024 viewport [18]. However, on higher–resolved surfaces scans, the compression rates are significantly lower and frame rates drop to about 20 fps on average, see [19]. The raycasting approach is, in general, slower and only gives frame rates of 40–50 fps for the UTAH data set (with a 1280×1024 viewport). However, the frame rates are independent of the complexity of the terrain, as just the heightfield values (and no geometry) are being sent to the GPU. The same is true for our approach, and we achieved higher frame rates on the UTAH data set when compared to the raycasting approach. We achieve frame rates of 90 fps when just rendering the heightfield and 60 fps when adding texture information (see next chapter) on older (and thus slower) hardware than the one used in [18, 19] with a viewport of 1000×1000. The triangle quality is higher for our approach (R_Q mainly between 0.44 and 0.5) when compared to regular grids (R_Q of about 0.41). For irregular meshes the triangle quality depends on the given sample distribution, while the criterion does not apply for the raycasting approach that is not using triangles. The screen–space error of all mentioned approaches have been chosen to be in the subpixel range for the reported performances.

Chapter 7

Applications to Terrain Data

In this chapter we will present the application of our algorithm to terrain data. Mainly, terrain information involve heightfield information plus additional attributes. Therefore, we propose a framework of several layers and initiate them in a preprocessing step.

To explain such a framework suppose that each sample consists of a range of values p_j for $j = 0, 1, .., N$ where N is the number of values, including height values and color values. The number N is also the number of layers in our framework such that $p_j \in \mathcal{R}^2$ for $j = 0$. p_0 indicates the location in parameter plane coordinates as (x, y) if p_j is undergone a GIS manipulation process that maps the earth location on a grid. Essentially, if the coordinates of the location are provided as $(longitude, latitude)$ and not mapped to a grid we apply the required transformations and map them onto our grid. The sampled parameter plane identifies the basic layer of our framework which is the parameterization layer. The rest of the information layers includes height and color values. Our algorithm considers the height values as the second layer and the aerial photography, i.e., color information as the third layer.

7.1 Aerial Imagery Data

Basically, imagery data is created by aerial photography techniques that rely on space–based cameras in remote sensing. The term space–based may correspond to taking images from a position on the landscape at a highly elevated surface via platforms that are fixed onto aircrafts, helicopters, balloons, blimps, etc. With the advancement in the world of the technology and the huge size of the

required surveyed data, satellite imagery, which is synonymous with ariel photography, is used to detect images in different types of resolution. However, the quality of the resulting images depends on the given resolution that is determined by both the used instrument and the altitude of the satellite orbit.

Because our interest is in spatial data and related attributes, satellite images are incorporated as a layer that captures the nature of the height surface. While the trend of our concerns is beyond presenting more detail in the detection process, we focus on UTAH data as it has been implemented in this work.

Our tiles used UTAH imagery data that were acquired within the National Agriculture Imagery Program (NAIP). NAIP is a program for providing the governmental agencies of the U.S.A with updated data in one year starting with the season of growing agricultures. All images are digital releases of ortho–photography with $1\,m$ sampling and color format as (R,G,B) natural color. The available formats are (R,G,B) natural color and (CIR) infrared color displayed in 4–band TIFF images as in Figure 7.1.

(RGB) (CIR)

Figure 7.1: UTAH imagery data color format for displaying data as TIFF images using natural (R,G,B) color (left), and (CIR) infrared color format (right).

Our algorithm receipts such data as RGB natural color format images for the third layer in which each sample values are composed of three color channels that are transferred to a texel on the texture memory where the format does not need any extra transformation. The VTF can access the color value from this layer and assign it to the correspond location on the triangular mesh. Notwithstanding, these images are not ready to be incorporated directly to our grid pyramids because these images are tiles in the format Digital Ortho Quarter Quad DOQQs tiles. In format DOQQs the areas correspond to U.S. Geological Survey (USGS) topographic quadrangles. Briefly,

each tile of DOQQs format has four quadrangles representing east, west, north, and south areas that are presented in the USGS topographic maps in which a set of longitude/latitude lines are created and through the intersections a location on a Universal Transverse Mercator (UTM) grid could be determined. Hence, we create our UTM grid by reading the header file parameters in each DOQQs format tile. To get the UTM geographic coordinates we use the following conversion equation with a, d, b, e, c and f parameters:

$$\begin{aligned} x_{utm} &= (a \cdot c_n) + (b \cdot r_n) + c \\ y_{utm} &= (d \cdot c_n) + (e \cdot r_n) + f \end{aligned} \tag{7.1.1}$$

where a is the pixel width in the x–direction; d is the rotation around the y–axis; b is the rotation around the x–axis; e is the pixel width in the y–direction; c is the x–coordinate of the pixel center (upper left); and f is the y–coordinate of the pixel center (upper left).

7.2 Discussion and Results

The layers of our algorithm framework are applied to UTAH data heights tiles and color imagery tiles. Regarding UTAH data imagery tiles, (2006, $Natural\,ColorImagery$) is selected with $1\,m$ resolution and every tile has parameters: $(a = 1.0, d = 0.0, b = 0.0, e = -1.0)$ and $(c, f) = (longitude_{ij}, latitude_{ij})$ at location ij of the given DOQ quadrangle. For height data we used UTAH tiles of (2006, $5\,Meter\,Auto-Correlated\,Elevation\,Model\,(DEM)$) with $5\,m$ resolution.

To use such data tiles we generate a UTM grid that covers landscape of $460\,km \times 600\,km$. The grid is sampled at $5\,m$ resolution for height tiles which is the same resolution as given by the DEM tiles. To get the same resolution in imagery tiles we do resolution down–sampling to gain imagery tiles with $5\,m$ resolution from given imagery tiles with $1\,m$ resolution. Every cell in UTM grid is computed using Equation 7.1.1. Every UTM grid's cell consists of a height value and a color value. Eventually, the UTM grid is tiled to tiles with 1024×1024 resolution that are loaded to two layers of textures. Tiles of both layers are made accessible to the VS program when they are on GPU memory space. It is worth mentioning that both height values and color values are loaded to textures in the same manner such that the tiles can be swapped easily for on–demand information. However, the high performance of rendering just heightfield will be affected by adding extra information,

i.e., reading height and color information instead of just height information per vertex (e.g. the performance of 90 fps in our experiment for UTAH data heightfield rendering becomes 60 fps if extra textures are used to capture the agriculture information). Figure 7.2 shows the final results of our algorithm using UTAH data where the output resolution is 1000×1000.

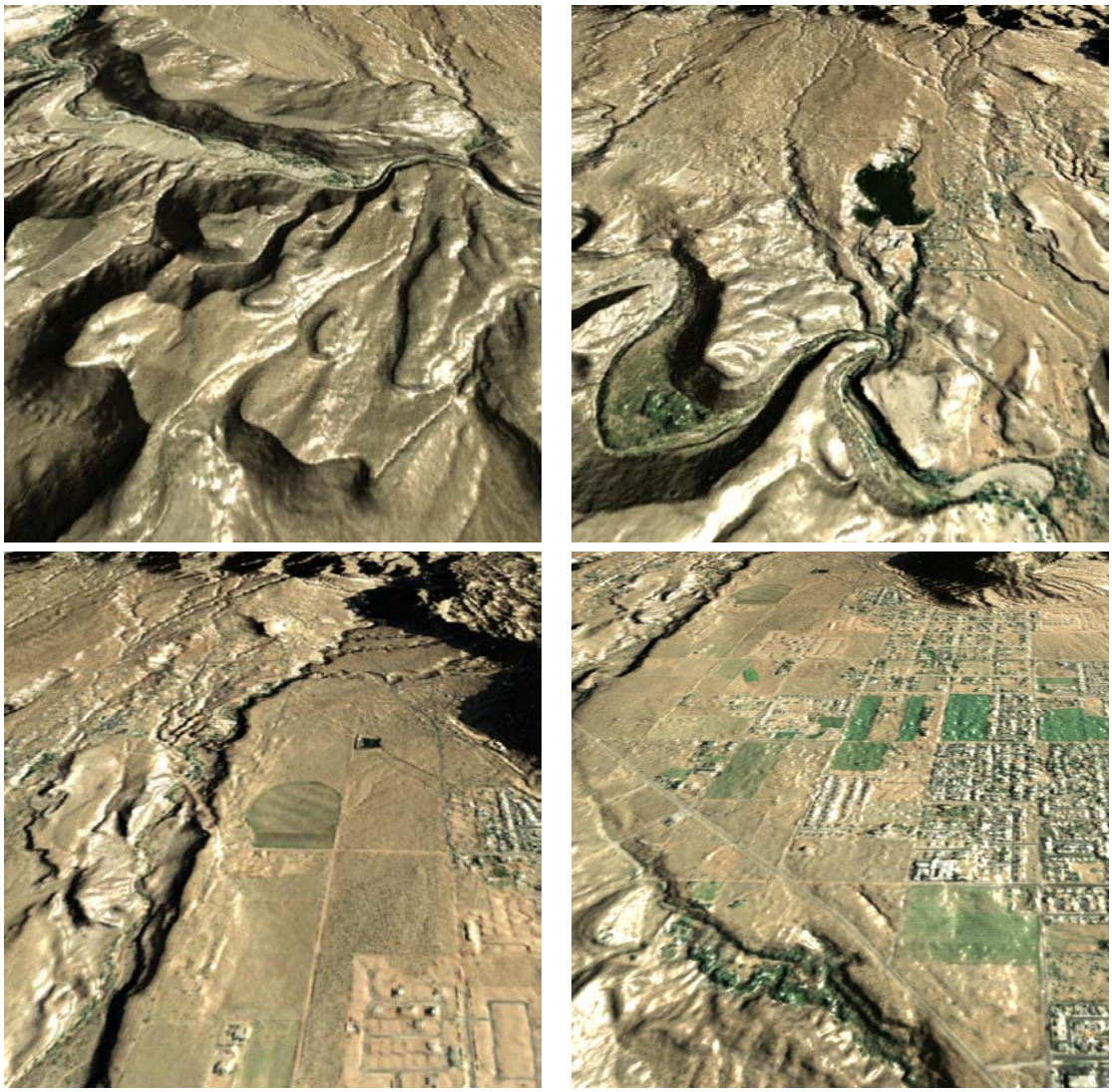


Figure 7.2: Four different views of our implementation on UTAH data using two layers of our framework, one for the height information and the other for satellite imagery data.

Chapter 8

Application to Bathymetry Data

We developed an interactive visual analysis system for bathymetry data, which is based on the heightfield visualization approach described in Chapter 5 which includes a number of additional features. Those features include backscatter data visualization, water column data visualization, and a number of useful interaction methods. However, our supposed layered framework as it is presented in Chapter 7 is similar with regards to the applied procedure to the first and second layers in which height information is replaced by depths.

Our system is also dealing with a specific data format that requires to be visualized in an efficient method to give the geologists an important information about the surface characteristics of the investigated ocean floor. Mainly, most popular ocean floor characteristics are backscatter data, water–column data, and subbottom–profile data.

The interactive visualization system consists of interactive visualization and exploration methods such as probing and coordinates views, a backscatter data visualization using color mapping, an uncertainty visualization using opacity mapping, and a visualization of gas seep data using an efficient density volume visualization. These methods provide the heightfield visualization with interesting and valuable tools that make use of some important analysis aspects of geo–spatial data.

8.1 Probing and Coordinated Views

Our developed system can explore and visualize bathymetry data interactively with the guide of intrinsic tools to probe the related information of every location within the current view. During

heightfield explorations our system allows the user to recognize the viewer location of the entire data within an overview rendering. In this rendering one can get a response of the user selection, i.e., a mouse click on the surface within the current view as a visual sign. All information that are related to the selected location are retrieved and displayed at run–time.

Once the user clicks on the surface the system detects the mouse position that is in the screen space as 2D position $X = (x, y)$ and then the position X is projected to 3D location in the object space as $X' = (x', y', z')$. Corresponding to the position X' the longitude and latitude of X' can be computed and the indices of the heightfield tile and its texel of X' can be determined where we can retrieve and display information such as longitude, latitude, height or depth, backscatter, and color values. Figure 8.1 shows our probing tool where the data rendering (left) is coordinated by a prepared view (right) interactively. The view is created by using a 2D image of a top–overview of the entire data. Such a view is telling the user over which area of the data set he/she is flying if he/she clicks on a location of the surface. Hence, the clicking is showing that location as a black dot on two sides.

8.2 Visualizing Backscatter Data

Recently, the advantages of multibeam sonar sensors are not limited to map only the wide high–resolution ocean floor area but it opens the door to study the seabed for different benthic researches as well. In addition to gain depth information of all multibeam acoustics reflections the energy density of the reflected waves can measure the fish habitats and the nature of the sediments.

To visualize the intensity of backscattered energy we map each given value in dB unit to an (R,G,B) color value in a smoothly varying cyan–to–red color map where a low backscatter intensity represents the more fine–grained sediments and a high backscatter intensity represents the more coarse–grained sediments.

Our result is shown in Figure 8.1 where the backscatter data are mapped to colors that go from cyan to red. We used HSV color model such that the luminance are constant and we set it to 0.8 and the saturation is set to 1.0 where the backscatter values are used for the channel H that is then mapped to RGB color map using the color transition from cyan to read.

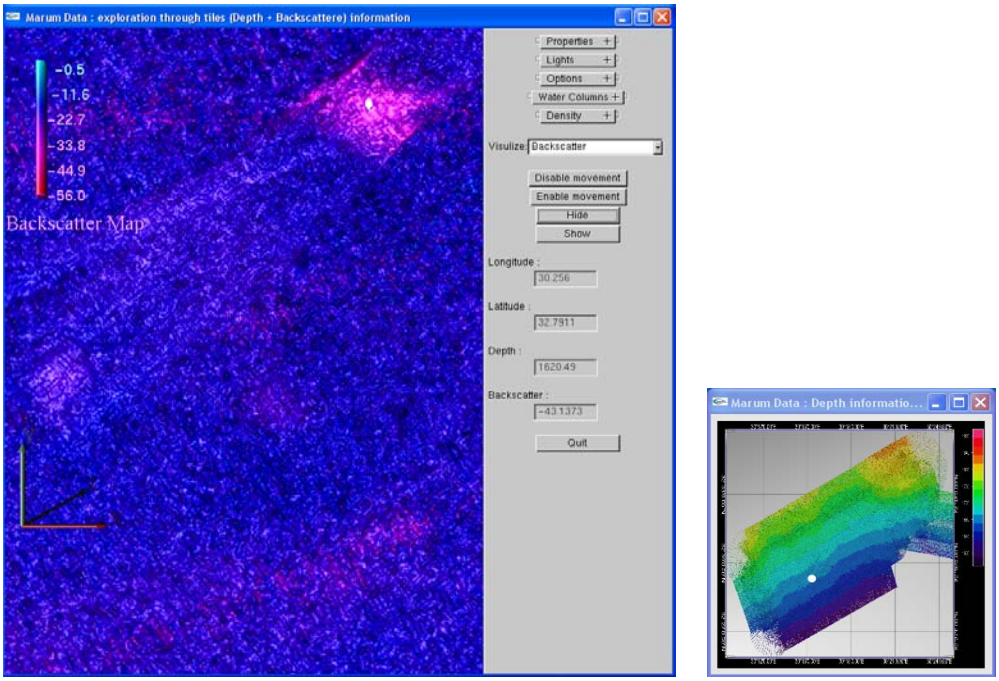


Figure 8.1: Backscatter data visualization of Ereğli: we are showing an example of mouse clicking on a surface (on left) and the same location on the entire top–view of coordinated view (on right). Different options of a toolbar allow us to get all information (longitude, latitude, depth, and backscatter) of the clicked location (white dot).

8.3 Uncertainty Visualization Using Opacity Mapping

An uncertainty visualization method is presented in our system in the context of real data distribution. The goal of this method is the interactive presentation of the real data cluttering in the interpolated surface. The visualization process is performed by combining the transparency and the color values of the rendered surface such that the user is allowed to control the transparency range via a slider that is created on a side toolbar.

In this technique every sample point of the heightfield has information about the distance to the nearest neighbor that is used to interpolate that sample point. We represent such distance to define the transparency of the color values. If the transparency of α is set to be a value between 0 and 1 we interpret mapping α to 0 as a non–visible color value of the interpolated sample points, i.e, we are getting the rendering of the surface as a set of the real sample points. When α is mapped to 1 we are getting the rendering of the entire surface as the interpolated samples are having fully opaque color. For any value of α that is happened to be between 0 and 1 the rendered surface is showing a translucent color values of the regions that contain only the interpolated sample points. The transparency of the color value is high if the nearest neighbor that is used for interpolating this sample point is far and vice versa.

The desired range of the transparency of α value can be set by utilizing a slider that is created on the side toolbar, see Figure 8.2.

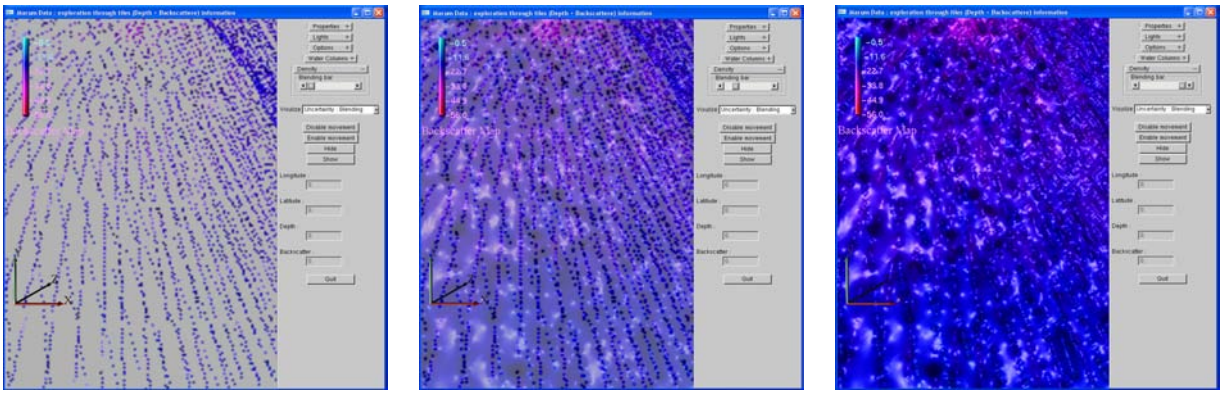


Figure 8.2: Three different α values are applied to visualize the uncertainty in data points. When $\alpha = 0$ we get only the point rendering of the real data set (left), $\alpha = 1$ we get the surface rendering (right), and for α is between 0 and 1 we get a transparency according to the cluttering of the points in the interpolated surface (middle).

8.4 Visualizing Gas Seeps Data

Through bathymetry data acquisition a set of water columns are specified to generate a gas map of the investigated ocean floor area. In this context, one can show the distribution of water–column data points by rendering them as point objects. This rendering gives unclear presentation of the data set without visual understanding of their distribution due to visual clutter. Therefore, visualizing the density distribution of the points would be a better option for an appropriate comprehension of the data.

To visualize the density we developed a simple yet effective direct volume rendering technique where the density is mapped to the opacity. Our approach is inspired by the splatting technique of direct volume rendering, where each sample is convoluted with 3D kernel using a Gaussian function. The volumetric contributions of all kernels need to be integrated in the viewing direction. Integrating a 3D Gaussian kernel leads to a 2D Gaussian kernel which is referred to as a splat, see Figure 8.3.

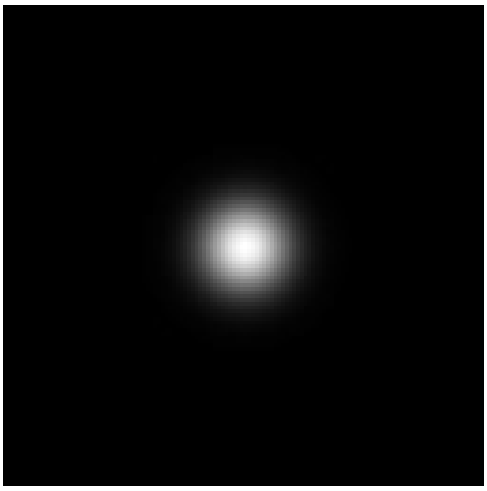


Figure 8.3: Example of the used $2D$ image with 128×128 resolution of an integrated 3D Gaussian kernel leading to a 2D Gaussian kernel to visualize the density of the points in a simple and a fast way. The $2D$ image is stored in a texture and used for all points.

We use these splats to render water–column data points where the data point is located in the center of the splat.

To render those splats, we use the pre–computed 2D footprint of the 3D kernel and store it in a 2D texture. Several oriented quadrilaterals are used to which we map the 2D footprint textures. The quadrilaterals are oriented orthogonal to the viewing direction, i.e., orthogonal to the ray from the viewpoint to the respective data point that is convoluted with the kernel. The 2D textures define opacity values while the RGB color information can be defined arbitrarily.

To achieve the RGBA values accumulation we render the oriented quadrilaterals in a front–to–back order using alpha blending. Alpha blending allows us to understand the distribution of the data points in each water column in which the blending is opaque when the points are grouped tightly and the blending becomes transparent when the points spread further away. The degree of transparency expresses the points density.

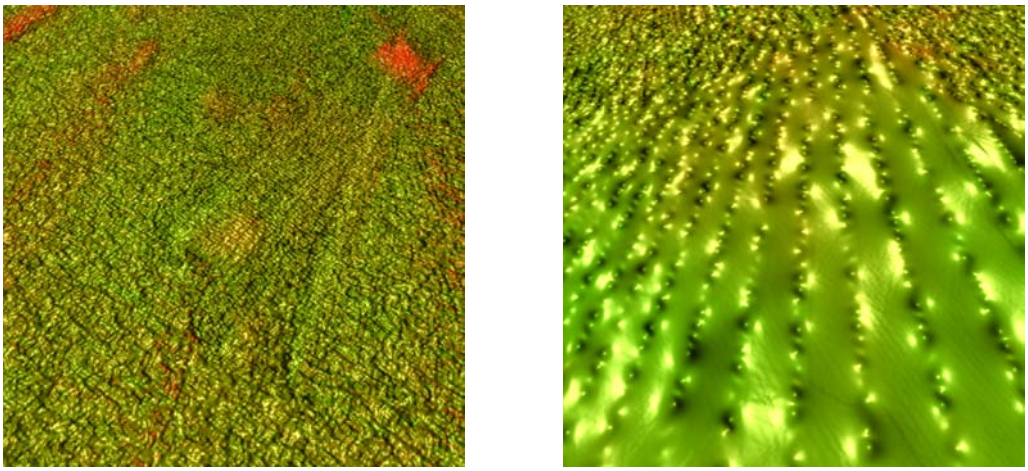
For multiple water–columns we use different color for every column. However, during interaction in the exploration process viewing directions are changed and the data point sets need to be resorted to keep the order manner front–to–back. To eliminate the need of resorting data sets at run–time we store three copies of sorted data set pointers in x–, y–, and z–direction. Thus, when we change our viewing position we change just the pointer direction.

Resorting the points at run–time depends on the number of points and costs time which is ≈ 12 sec for resorting about $95,000$ points of three water–columns in x– and y–directions using heap sorting technique. To avoid such time costs we prepare three copies for every water–column, i.e., a copy for sorting points in x–direction, a copy for sorting the points in y–direction, and a copy for sorting the points in z–direction. Thus, the resorting is done free of time cost by just reversing the reading order of the pointers using some extra memory storage.

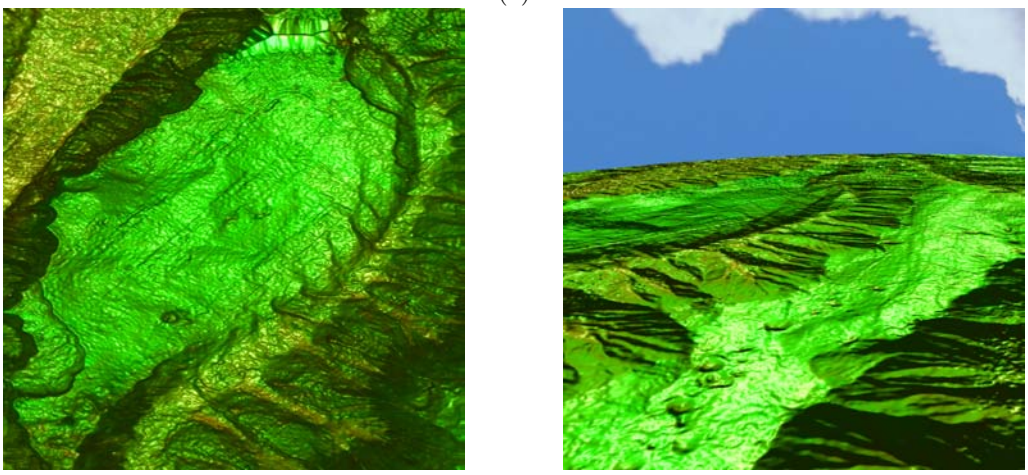
8.5 Discussion and Results

Our application to bathymetry data focused on ocean floor data rendering and integrates the backscatter data visualization and the water–column data. The integration of sub–bottom profile is left for future work. For backscatter data we were able to visualize the backscatter reflections using color mapping. Our approach allows the user to select any location on the rendered surface

interactively. Backscatter, longitude, and latitude information of every selected location is shown on a side toolbar. We used a coordinated view to render a top–view of the entire data with a color map. Data from the Black Sea close to Ereğli in Turkey is used to show our results for backscatter data visualization with interactive selection and displaying the retrieved values on a side toolbar, see Figure 8.1. Figure 8.4 shows also our results using two different data sets from Black Sea close to Ereğli in Turkey.



(a)



(b)

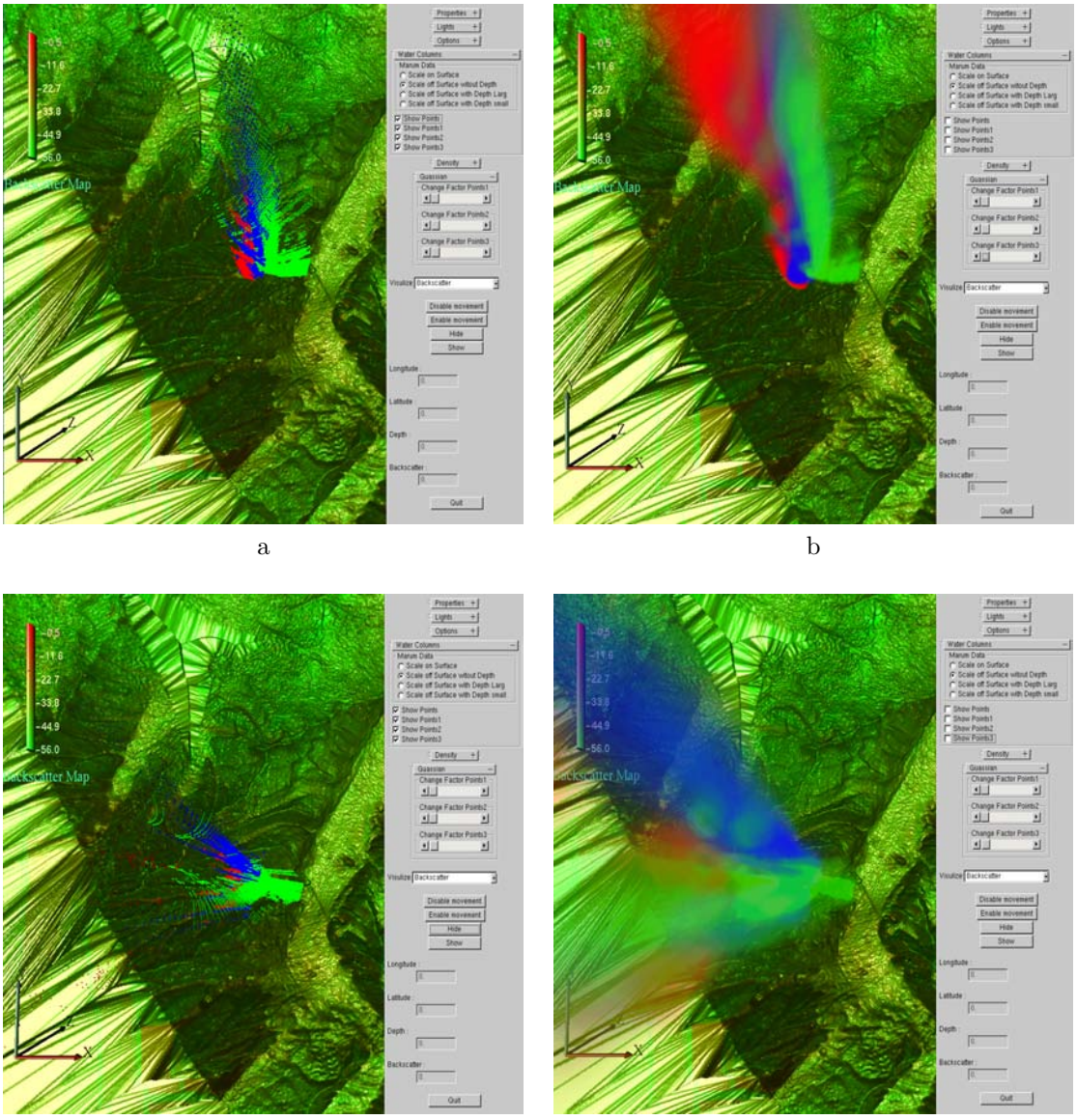
Figure 8.4: Backscatter visualization of two different Black See Ereğli data sets.

The Ereğli data set is also used to show our result for the uncertainty visualization method using the opacity mapping as shown in Figure 8.2.

We introduced another integration for water–column data set visualization. Our results used three columns of gas seeps that are attached to the Ereğli data set. The first column consists of $23,106$ data points, the second column consists of $47,174$ data points, and the third column consists of $23,921$ data points. Our approach uses Gaussian function to visualize the columns' point density that is accumulated in viewing direction. One $2D$ texture image of Gaussian function is used for all points in the three columns. However, three different constant colors are used to distinguish the overlapping of those columns' points. Furthermore, different Gaussian functions can be implemented at run–time using a slider on the side toolbar. Thus, the user is allowed to change width and height of the applied Gaussian function interactively. Figure 8.5 shows our application results of gas seeps visualization.

To do points sorting we choose Heap sorting algorithm because it provides cheap time complexity in the order of $O(n\log(n))$ concerning both Heap building and Heap sorting. In addition we exploit more advantages such as stability, adaptivity, and complete binary shape.

However, water–column data visualization is missing the shading effect which may be supported in the future plan for presenting better depth perception.



a b

c d

Figure 8.5: Three water columns distinguished in red, green, and blue colors. Gas seeps density is visualized by a novel fast direct volume rendering approach where every data point is rendered as a splat with Gaussian kernel for transparency. All points are sorted from close location to the viewing point further away. Thus, blending all points shows the density of all gas seeps for different viewing poses and compare the points rendering (a and c) with our visualization method (b and d).

Chapter 9

Conclusion and Future Work

We presented a visualization approach for heightfield rendering that decouples the mesh representation from the data representation. The data are stored in a grid–based multiresolution hierarchy that is loaded to texture memory of the GPU. We used a tiling strategy to replace the hierarchies in texture memory on demand. The mesh is an irregular triangular mesh that is precomputed using considerations of view–dependency (triangle sizes) and rendering quality (triangle shape). The precomputed mesh also resides on the GPU. During the runtime stage, the two representations are combined by doing texture lookups for all vertices of the mesh to update their height values in each frame. Consequently, no geometry needs to be sent from the CPU to the GPU. We made experiments to determine the optimal ratio of (projected) triangle size to screen pixel size. For the optimal ratio, we reported frame rates of 90 fps for a rendering output window of size 1000×1000 and 60 fps for size 1600×1200. Moreover, we have shown that the triangles' quality is higher than that of state–of–the–art semi–regular approaches. Our decoupling strategy assures that we always generate watertight meshes without any special handling of cracks (T–junctions) or tile stitching.

Several optional layers are added to the framework to visualize information of both terrain data (i.e. heights) and bathymetry data (i.e. depths). Concerning terrains we had achieved landscape exploration utilizing UTAH data size of 790 GB with textures of 1024×1024 resolution for each tile at finest level of detail and high performance of 60 fps.

Regarding bathymetric data we implemented several visualization techniques for depth data

accompanied with backscatter, density, and water–column information. We showed our backscatter visualization that is based on a color map by rendering the scene with the coordinated views to probe and display the information of any selected location of the rendered surface. The other advantage of the coordinated views is also providing a top–view of the entire data depths that are visualized in a separated color map. Moreover, a set of interactions is provided using a side toolbar. On this toolbar we can display all selected information and there are more options related to the density information and the water–column data.

For water–column data we showed the ability of our algorithm of incorporating density visualization of volumetric data sets in the same scene using Gaussian function as a kernel. Users can easily change the used Gaussian function via a slider on the side toolbar.

However few limitations of our work could be presented as follows:

I. One of the major limitations of our work is texture memory capacity of the GPU. Therefore, we limited the number of levels of detail in every pyramid to three levels starting with the level of 1024×1024 resolution as the finest resolution.

II. Another limitation addresses the size of the water–column data set where our approach is rendering water–column data distribution on the CPU. Thus, the frame rate of the rendered scene drops down as the number of water–column data sets increases. However, we had feasible results using $\approx 95,000$ data points for three gas seeps columns.

Visualizing sub–bottom profile information of bathymetric data field will provide the geologists with precious analysis of natural resources underneath the ocean floor. The major challenge in this task is rendering the information of the sub–bottom profiles on the surface interactively with an effective visualization method. Such task is still one of the interesting challenges for our work that is left to the future work.

We achieved a beneficial visualization approach for water–column data density using the splatting technique of direct volume rendering. Our future extension to this approach involves the integration of data illuminations such as the approach that is presented by Sanftmann and Weiskopf [72].

Bibliography

[1] Maria-Elena Algorri and Francis Schmitt, *Mesh simplification*, Proceedings of EuroGraphics'96, Computer Graphics Forum (Futuroscope, Poitiers, France), no. 3, 1996, pp. 77–86.

[2] Lucas Ammann, Olivier Génevaux, and Jean-Michel Dischler, *Hybrid rendering of dynamic heightfields using ray-casting and mesh rasterization*, Proceedings of Graphics Interface 2010 (Toronto, Ont., Canada, Canada), GI '10, Canadian Information Processing Society, 2010, pp. 161–168.

[3] Peter Astheimer and Maria-Luise Pöche, *Level-of-detail generation and its applications in virtual reality*, Virtual Reality Software and Technology (Proceedings of VRST'94, August 23-26, 1994, Singapore) (Singapore), World Scientific Publishing, August 1994, pp. 299–312.

[4] Michel Beigbeder and Ghassan Jahami, *Managing levels of detail with textured polygons*, COMPUGRAPHICS '91, First International Conference on Computational Graphics and Visualization Techniques, 1991, pp. 479–489.

[5] Salim Belblidia, *Generating various levels of detail of architectural objects for image-quality and frame-rate control rendering.*, Computer Graphics International, 1996, pp. 84–89.

[6] Salim Belblidia, Jean-Pierre Perrin, and Jean-Claude Paul, *Multi-resolution rendering of architectural models*, International Conference on Computer-Aided Architectural Design Futures, 1995.

[7] Willem H. De Boer, *Fast Terrain Rendering Using Geometrical Mipmapping*, 2000, pp. 1–7.

[8] Jonas Bösch, Prashant Goswami, and Renato Pajarola, *RASTeR: Simple and efficient terrain rendering on the GPU*, Proceedings EUROGRAPHICS Areas Papers, Scientific Visualization, 2009, pp. 35–42.

[9] Bradford Chamberlain, Tony DeRose, Dani Lischinski, David Salesin, and John Snyder, *Fast rendering of complex environments using a spatial hierarchy*, Proceedings of the conference on Graphics interface '96 (Toronto, Ont., Canada, Canada), Canadian Information Processing Society, 1996, pp. 132–141.

[10] James H. Clark, *Hierarchical geometric models for visible surface algorithms*, vol. 19, ACM, October 1976, pp. 547–554.

[11] Jonathan Cohen, Marc Olano, and Dinesh Manocha, *Appearance-preserving simplification*, IN PROC. SIGGRAPH98, The University of North Carolina at Chapel Hill, 1998, pp. 115–122.

[12] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright, *Simplification envelopes*, vol. 30, 1996, pp. 119–128.

[13] Daniel Cohen-Or and Yishay Levanoni, *Temporal continuity of levels of detail in delaunay triangulated terrain*, VIS '96: Proceedings of the 7th conference on Visualization '96 (Los Alamitos, CA, USA), IEEE Computer Society Press, 1996, pp. 37–42.

[14] Oceanic Imaging Consultans, *White a Brief History of Sonar Development*, Oceanic Imaging Consultans, `http://www.oicinc.com/history_sonars.html`.

[15] Luebke David, *A survey of polygonal simplification algorithms*, 1997.

[16] Michael J. DeHaemer Jr. and Michael J. Zyda, *Simplification of objects rendered by Polygonal Approximations*, vol. 15, 1991, pp. 175–184.

[17] Tony D. DeRose, Michael Lounsbery, and Joe Warren, *Multiresolution analysis for surface of arbitrary topological type*, Report 93-10-05, Department of Computer Science, University of Washington, Seattle, WA, 1993.

[18] Christian Dick, Jens Krüger, and Rüdiger Westermann, *GPU ray-casting for scalable terrain rendering*, Proceedings of Eurographics 2009 - Areas Papers, 2009, pp. 43–50.

[19] Christian Dick, Jens Schneider, and Rüdiger Westermann, *Efficient geometry compression for GPU-based decoding in realtime terrain rendering*, vol. 28, 2009, pp. 67–83.

[20] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Miller Mark C., Charles Aldrich, and Mark B. Mineev-Weinstein, *ROAMing terrain: real-time optimally adapting meshes*, IEEE Visualization, 1997, pp. 81–88.

[21] Jonathan LoneSock Dummer, *Cone step mapping: An iterative ray–heightfield intersection algorithm*, 2006, `http://www.lonesock.net/files/ConeStepMapping.pdf`.

[22] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle, *Multiresolution analysis of arbitrary meshes*, Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '95, ACM, 1995, pp. 173–182.

[23] Carl Erikson, *Polygonal Simplification : An Overview*, Tech. Report TR96-016, Department of Computer Science, UNC-Chapel Hill, January 1996.

[24] Carl Erikson, Dinesh Manocha, and William V. Baxter III, *Hlods for faster display of large static and dynamic environments*, Proceedings of the 2001 symposium on Interactive 3D graphics (New York, NY, USA), I3D '01, ACM, 2001, pp. 111–120.

[25] Raphael A. Finkel and Jon Louis Bentley, *Quad trees a data structure for retrieval on composite keys*, vol. 4, March 1974, pp. 1–9.

[26] Leila Floriani, Leif Kobbelt, and Enrico Puppo, *A Survey on Data Structures for Level-of-Detail Models*, Advances in Multiresolution for Geometric Modelling (Neil A. Dodgson, Michael S. Floater, and Malcolm A. Sabin, eds.), Mathematics and Visualization, Springer-Verlag, Berlin/Heidelberg, 2005, pp. 49–74.

[27] Leila De Floriani, Enrico Puppo, and Paola Magillo, *A formal approach to multiresolution modeling*, Geometric Modeling: Theory and Practice (R. Klein, W. Straßer, and R.Rau, eds.), Springer, 1997, pp. 302–323.

[28] Steven Fortune, *Voronoi diagrams and delaunay triangulations*, Computing in Euclidean Geometry (Lecture Notes on Computing 1) (F. K. Hwang and D.-Z. Du, eds.), World Scientific, 1992.

[29] Michael Garland and Paul S. Heckbert, *Fast polygonal approximation of terrains and height fields*, Tech. Report CMU-CS-95-181, Computer Science Department, Carnegie Mellon University, 1995.

[30] Michael Garland and Paul S. Heckbert, *Surface simplification using quadric error metrics*, Proceedings of the 24th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.

[31] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe, *Geometry images*, SIGGRAPH, 2002, pp. 355–361.

[32] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder, *Normal meshes*, SIGGRAPH, 2000, pp. 95–102.

[33] Bernd Hamann, *A data reduction scheme for triangulated surfaces*, vol. 11, 1994, pp. 197–214.

[34] Jonathan C. Hardwick, *Implementation and evaluation of an efficient parallel delaunay triangulation algorithm*, in Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 1997, pp. 23–25.

[35] Paul Heckbert and Michael Garland, *Multiresolution modeling for fast rendering*, Proceedings of Graphics Interface '94 (Banff, Alberta, Canada), Canadian Information Processing Society, May 1994, pp. 43–50.

[36] Paul Hinker and Charles Hansen, *Geometric optimization*, Visualization '93, 1993, pp. 189–195.

[37] Hugues Hoppe, *Progressive meshes*, vol. 30, 1996, pp. 99–108.

[38] Hugues Hoppe, *Smooth view-dependent level-of-detail control and its application to terrain rendering*, Proceedings of the conference on Visualization '98 (Los Alamitos, CA, USA), VIS '98, IEEE Computer Society Press, 1998, pp. 35–42.

[39] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle, *Mesh optimization*, vol. 27, 1993, pp. 19–26.

[40] Hoppe Hugues, *View-dependent refinement of progressive meshes*, SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM Press/Addison-Wesley Publishing Co., 1997, pp. 189–198.

[41] Martin Isenburg, Yuanxin Liu, Jonathan Richard Shewchuk, and Jack Snoeyink, *Illustrating the streaming construction of 2d delaunay triangulations*, Proceedings of the twenty-second annual symposium on Computational geometry (New York, NY, USA), SCG '06, ACM, 2006, pp. 481–482.

[42] Alan D. Kalvin and Russel H. Taylor, *Superfaces: Polyhedral approximation with bounded error*, SPIE Proceedings, Medical Imaging : Image Capture, Formatting, and Display (Yongmin Kim, ed.), May 1994, pp. 2–13.

[43] Alan D. Kalvin and Russell H. Taylor, *Superfaces: Polygonal mesh simplification with bounded error*, vol. 16, IEEE Computer Society Press, May 1996, pp. 64–77.

[44] David Koller, Peter Lindstrom, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory Turner, *Virtual gis: A real-time 3d geographic information system*, Proceedings of the 6th conference on Visualization '95 (Washington, DC, USA), VIS '95, IEEE Computer Society, 1995, pp. 94–100.

[45] Mike Krus, Patrick Bourdot, Françoise Guisnel, and Gullaume Thibault, *Levels of detail & polygonal simplification*, vol. 3, ACM, May 1997, pp. 13–19.

[46] Aaron Lee, Henry Moreton, and Hugues Hoppe, *Displaced subdivision surfaces*, Proceedings of ACM SIGGRAPH 2000, July 2000, pp. 85–94.

[47] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner, *Real-time, continuous level of detail rendering of height fields*, SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1996, pp. 109–118.

[48] Peter Lindstrom and Valerio Pascucci, *Visualization of large terrain made easy*, 2001, pp. 363–370.

[49] Yotam Livny, Neta Sokolovsky, Tal Grinshpoun, and Jihad El-Sana, *A gpu persistent grid mapping for terrain rendering*, vol. 24, 2008, pp. 139–153.

[50] Frank Losasso and Hugues Hoppe, *Geometry clipmaps: terrain rendering using nested regular grids*, vol. 23, 2004, pp. 769–776.

[51] David Luebke, *Hierarchical structures for dynamic polygonal simplification*, no. TR 96-006, 1996.

[52] David Luebke and Carl Erikson, *View-dependent simplification of arbitrary polygonal environments*, Proceedings of the 24th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., 1997, pp. 199–208.

[53] Paulo W. C. Maciel and Peter Shirley, *Visual navigation of large environments using textured clusters*, In 1995 Symposium on Interactive 3D Graphics, 1995, pp. 95–102.

[54] Reddy Martin, *A survey of level of detail support in current virtual reality solutions.*, 1995, pp. 95–98.

[55] Reddy Martin, *Perceptually modulated level of detail for virtual environments*, Ph.D. thesis, University of Edinburgh, 1997.

[56] NOAA National Ocean and Atmospheric Administration, *Noaa and partners demonstrate success of multibeam sonar to detect and map deep-sea gas seeps*, NOAA National Ocean and Atmospheric Administration United States Department of Commerce, `http://www.noaanews.noaa.gov/stories2011/20110915_okeanosexplorer.html`.

[57] Southampton : University of Southampton National Oceanography Center and GeoAcoustics : A Kongsberg Company Natural Environment Research Council, *Marine Seismic Reflection*, 3D Chirp Decimetre-Resolution Sub-bottom Profiling, `http://www.noc.soton.ac.uk/soes/research/groups/3dchirp/background.htm`.

[58] NVIDIA, *White Paper Coverage-Sampled Antialiasing*, NVIDIA White Paper, `http://developer.download.nvidia.com/SDK/10/direct3d/Source/CSAATutorial/doc/CSAATutorial.pdf`.

[59] AML Oceanographic, *Multibeam System Overview*, AML Oceanographic, `http://www.amloceanographic.com/Home/Multibeam-Overvew`.

[60] Renato Pajarola, *Large scale terrain visualization using the restricted quadtree triangulation*, vol. 0, IEEE Computer Society, 1998, pp. 19–26.

[61] Renato Pajarola and Enrico Gobbetti, *Survey of semi-regular multiresolution models for interactive terrain rendering*, vol. 23, Springer-Verlag New York, Inc., July 2007, pp. 583–605.

[62] Jean L. Pajon, Yves Collenot, Xavier Lhomme, Nicolas Tsingos, François X. Sillion, Pascal Guilloteau, Pascal Vuyslteker, G. Grillon, and D. David, *Building and Exploiting Levels of Detail: An Overview and some VRML Experiments*, VRML '95 First Annual Symposium on the Virtual Reailty Modeling Language, December 1995, pp. 117–122.

[63] Sung W. Park, Lars Linsen, Oliver Kreylos, John D. Owens, and Bernd Hamann, *Discrete Sibson interpolation*, vol. 12, 2006, pp. 243–253.

[64] Thomas K. Peucker, Robert J. Fowler, James J. Little, and David M. Mark, *The triangulated irregular network*, vol. 516, 1978, pp. 96–103.

[65] Fabio Policarpo and Manuel M. Oliveira, *Relaxed cone stepping for relief mapping*, GPU Gems 3 (Hubert Nguyen, ed.), Addison-Wesley, 2008, pp. 409–428.

[66] Extended Continental Shelf Project, *2010 u.s.-Canada Arctic Continental Shelf Survey*, Extended Continental Shelf Project, `http://continentalshelf.gov/missions/10arctic/background/plan.html#skip`.

[67] Martin Reddy, *Reducing Lags in Virtual Reality Systems using Motion-Sensitive Level of Detail*, 1994, pp. 25–31.

[68] Repository, *California State University, Monterey Bay, See Floor Mapping Lab*, Data Library Website, `http://seafloor.csumb.edu/SFMLwebDATA_mb.htm`.

[69] Jarek R. Rossignac and Paul Borrel, *Multi-resolution 3D approximations for rendering complex scenes*, Geometric Modeling in Computer Graphics (Genova, Italy) (B. Falcidieno and T. L. Kunii, eds.), Springer-Verlag, 1993, Also published as technical report RC 17697 (#77951), IBM Research Division, T. J. Watson Research Center, 1992, pp. 455–465.

[70] Stefan Rttger, Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel, *Real-time generation of continuous levels of detail for height fields*, Winter School in Computer Graphics (WSCG Proceedings), 1998, pp. 315–322.

[71] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe, *Texture mapping progressive meshes*, Proceedings of the 28th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '01, ACM, 2001, pp. 409–416.

[72] Harald Sanftmann and Daniel Weiskopf, *Illuminated 3D Scatterplots*, vol. 28, Blackwell Publishing Ltd, 2009, pp. 751–758.

[73] Gernot Schaufler, *Exploiting frame to frame coherence in a virtual reality system*, VRAIS '96, April 1996, pp. 92–102.

[74] Gernot Schaufler and Wolfgang Stuerzlinger, *Generating multiple levels of detail for polygonal geometry models*, Virtual Environments (Monte Carlo, MC), January 1995, pp. 53–62.

[75] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen, *Decimation of triangle meshes*, SIGGRAPH '92, vol. 26, July 1992, pp. 65–70.

[76] Byeong-Seok Shin and Ei-Kyu Choi, *An efficient clod method for large-scale terrain visualization*, Entertainment Computing ICEC 2004 (Matthias Rauterberg, ed.), Lecture Notes in Computer Science, vol. 3166, Springer Berlin / Heidelberg, 2004, pp. 3–23.

[77] Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones, *The clipmap: a virtual mipmap*, Proceedings of the 25th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '98, ACM, 1998, pp. 151–158.

[78] Art Tevs, Ivo Ihrke, and Hans-Peter Seidel, *Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering*, Symposium on Interactive 3D Graphics and Games (i3D'08), 2008, pp. 183–190.

[79] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi, *3d Mesh Skeleton Extraction Using Topological and Geometrical Analyses*, 14th Pacific Conference on Computer Graphics and Applications (Pacific Graphics 2006) (Taipei, Taiwan), 2006, pp. 85–94.

[80] Greg Turk, *Re-tiling polygonal surfaces*, Computer Graphics (SIGGRAPH '92 Proceedings), vol. 26, July 1992, Also available a TR92-006, Department of Computer Science, University of North Carolina at Chapel Hill, pp. 55–64.

[81] Thatcher Ulrich, *Rendering massive terrains using chunked level of detail control*, SIGGRAPH Course Notes **3** (2002).

[82] Philippe Véron and Jean-Claude Léon, *Static polyhedron simplification using error measurements*, vol. 29, 1997, pp. 287–298.

[83] Douglas J. Wiley and Allen N. Duckworth, *Adaptive level of detail technique for real-time display in virtual reality*, Proceedings of 24th Applied Imagery Pattern Recogition Workshop, Tools and Techniques for Modeling and Simulation, SPIE, 1996, pp. 192–203.